1.7 table - Module

Glish interface to table system
include "table.g"

Description

AIPS++ stores all data inside AIPS++ tables which are stored on disk. An AIPS++ table consists of an unlimited number of columns of data, with optional column keywords and optional table keywords. Columns are named and rows are numbered (starting at 1). The columns hold data, such as visibilities and uv coordinates, while the keywords hold general information such as units or revision numbers or table author or even other tables.
To make this concrete, examples of columns might be:

```
  U       V        W         TIME        ANT1   ANT2      VISIBILITY
124.011 54560.0  3477.1  43456789.0990    1      2        4.327 -0.1132
34561.0 45629.3  3900.5  43456789.0990    1      3        5.398 0.4521
....
....
....
```

and examples of keywords might be:

```
REVISION=2.01
AUTHOR="Tim Cornwell"
INSTRUMENT="VLA"
```

Everything in an AIPS++ table (and thus all data stored in AIPS++) is potentially accessible and changable from Glish. The table module provides a convenient way of accessing and changing AIPS++ tables from inside Glish. To do this one creates table tools inside Glish. The resulting tools can be operated on in various ways:

- Opening, copying, renaming of existing tables, (using table, tablecopy, tablerename),
- Get and put of table cells, columns and keywords,
- Selection and sorting with TaQL (using the query tool function or calc tool function or tablecommand),
- Iteration by subtables (using tableiterator),
- Get and put of table information strings,
- Access via table columns (using tablecolumn),

- Access via table rows (using tablerow),
- Indexed access (using tableindex),
- Browsing of tables (using the browse tool function),
- Printing of a summary of a table (using summary tool function),
- Determine if a Glish value is a valid table (using is_table),
- Writing or reading a table to or from an ASCII format (using tablefromascii, and toascii),
- Reading a table from binary FITS format (using tablefromfits)

There is no `tofits` function available. It might get available in the future.
In addition this module contains a number of global functions related to tables, such as determining if a table exists (tableexists).
All operations are done inside the Glish client and are not written to disk until an explicit flush command is performed.
The most typical operation on an AIPS++ table is to open it by creating a table tool inside Glish, load a column from the table into Glish, alter it using Glish capabilities, and then write it back to the table. For this only a few commands are relevant: see the example below.
Sorting and selecting of tables is possible. The table thus produced is a *reference* table, and points back to the original table.
The Glish client handling all table events and operations is the `tableserver`. At startup of `table.g` the `defaulttableserver` is started which runs until the glish session ends. This server is by default used. It is, however, possible to start another one like:

```
mytableserver:=tableserver();
```

and use that in the various constructors and functions taking a `tableserver` argument. In a similar way the `defaulttableserver` can be restarted in case it crashes in one way or another.

Example

```
include "table.g"
vis:=table("3C273XC1.MS", readonly=F);
vis.summary();
uvw:=vis.getcol("UVW");
spw:=table("3C273XC1.MS/SPECTRAL_WINDOW", readonly=T);
freq:=spw.getcell("REFERENCE_FREQUENCY", 1);
uvw*:=(1.420E9/freq);
vis.putcol("UVW", uvw);
vis.close();
```

See Also
tablecolumn tablerow tableiterator tableindex

Aipsrc Variables

table.relinquish.reqautolocks.interval	nr of seconds to wait before relinquishing autolocks requested in another process Allowed:float Default: 5
table.relinquish.allautolocks.interval	nr of seconds to wait before relinquishing all autolocks Allowed:float Default: 60
table.endianformat	endian format to be used for storing data in new tables Allowed:big,little,local Default: big

Tools

table	Access tables from Glish
tablecolumn	Allows access to a column in a table
tablecomparer	Compare various aspects of two tables to see if they are equivalent
tableindex	Allows value based table access
tableiterator	Allows iteration through a table tool
tableplot	Plot data from tables via Glish
tablerow	Allows access to rows of a table tool

Functions

is_table	Is this a valid table tool?
tablebrowser	"Table browser"
tablecloseall	close all open table tool objects
tablecommand	Execute a table command
tablecopy	copy a table
tablecreatearraycoldesc	create descriptor of an array column
tablecreatedesc	create a table descriptor
tablecreatescalarcoldesc	create descriptor of a scalar column
tabledefinehypercolumn	define a hypercolumn in a table descriptor
tabledelete	delete a table
tabledemo	Perform a demonstration: Not yet implemented
tableexists	does this table exist on disk?
tableinfo	return the info record
tableiswritable	is this table writable?
tableopentables	return names of open tables
tablerename	rename a table
tabletest	Perform a test

1.7.1 table - Tool

Package utility
Module table

Access tables from Glish

Description
table is the tool that contains all the tool functions relevant for table handling.
To make a table tool, use the constructor.

Constructors

table	Construct table tool
tablefromascii	Create an AIPS++ table from an ASCII format
tablefromfits	Create an AIPS++ table from binary FITS

Functions

addcols	add one or more columns
addreadmeline	add a readme line to the info record
addrows	add a specified number of rows
browse	browse a table using a graphical browser
calc	Calculate an expression on a table
close	close the table tool
colarraytype	return the column array type
coldatatype	return the column data type
colfieldnames	get the names of fields in a keyword in a column
colkeywordnames	get the names of all keywords in a column
colnames	return the names of the columns
copy	copy a table
copyrows	copy rows from this table to another
datachanged	has data changed in table?
done	end the table tool
endianformat	get the endian format used for this table
fieldnames	get the names of fields in a table keyword
flush	flush the current contents to disk
getcell	get a specific cell
getcellslice	get a slice from a specific cell
getcol	get a specific column
getcoldesc	get the description of a specific column
getcolkeyword	get value of specific column keyword
getcolkeywords	get values of all keywords for a column
getcolshapestring	get shape of arrays in a specific column
getcolslice	get a slice from a specific column

getdesc	get the table description
getdminfo	get the info about data managers
getkeyword	get value of specific table keyword
getkeywords	get values of all table keywords
getvarcol	get a specific column (for variable arrays)
haslock	has this process a lock on the table?
info	get the info record
iscelldefined	test if a specific cell contains a value
ismultiused	is the table in use in another process?
isscalarcol	is the specified column scalar?
isvarcol	tell if column contains variable shaped arrays
iswritable	is the table writable?
keywordnames	get the names of all table keywords
lock	acquire a lock on the table
lockoptions	get the lock options used for this table
name	return name on disk
ncols	return number of columns
nrows	return number of rows
ok	Is the table tool ok?
putcell	put a specific cell
putcellslice	put a slice into a specific cell
putcol	put a specific column
putcolkeyword	put a specific keyword for a column
putcolkeywords	put multiple keywords for a column
putcolslice	put a slice into a specific column
putinfo	set the info record
putkeyword	put a specific table keyword
putkeywords	put multiple table keywords
putvarcol	put a specific column (for variable arrays)
query	Make a table from a query
removecolkeyword	remove a specific keyword for a column
removecols	remove one or more columns
removekeyword	remove a specific table keyword
removerows	remove the specified rows
renamecol	rename a column
resync	resync the table tool with table file
rownumbers	return the row numbers in the (reference) table
selectrows	Make a table from a selection of rows
setmaxcachesize	set maximum cache size for column in the table
summary	summarize the contents of the table
toascii	Write AIPS++ table into an ASCII format
unlock	unlock and flush the table

table - Constructor

Package utility

Module **table**
Tool **table**

Construct table tool

Synopsis
table(tablename, tabledesc, nrow, readonly, lockoptions, ack, dminfo, endian, memorytable, tableserver, tablenote, handle)

Description
Use this constructor to construct a table tool inside Glish from the name of a disk file containing an AIPS++ Table. A new table may also be created from a table descriptor (see tablecreatedesc). When creating a new table, detailed data manager information can be given using the `dminfo` argument. This is a record as returned by the getdminfo function.
Most of the arguments are rarely used: most of the time, you'll just need to use the tablename, and perhaps readonly.
A table can be shared by multiple processes by using the appropriate locking options. The possible options are:
- auto: let the system take care of locking. At regular time intervals these autolocks are released to give other processes the opportunity to access the table. The aipsrc variable
`table.relinquish.reqautolocks.interval` defines the number of seconds between releasing autolocks on tables needed in another process.
`table.relinquish.allautolocks.interval` defines the number of seconds between releasing all autolocks.
- autonoread: as auto, but no read locking is needed. This must be used with care, because it means that reading can be done while the table tool is not synchronized with the table file (as is normally done when a lock is acquired). The function `resync` can be used to explicitly synchronize the table tool
- user: the user takes care by explicit calls to lock and unlock
- usernoread: as user and the no readlocking behaviour of autonoread.
- permanent: use a permanent lock; the constructor fails when the table is already in use in another process
- permanentwait: as above, but wait until the other process releases its lock
- default: this is the default option. If the given table is already open, the locking option in use is not changed. Otherwise it reverts to auto.
When auto locking is used, it is possible to give a record containing the fields option, interval, and/or maxwait. In this way advanced users have full control over the locking options. In practice this is hardly ever needed.
When creating a new table, the endian format in which the data should be stored, can be specified. The possible values are:
- big: big endian format (as used on e.g. SUN)

- little: little endian format (as used on e.g. PC)
- local: use the endian format of the machine being used
- aipsrc: use the endian format specified in aipsrc variable table.endianformat (which defaults to big).

The default is aipsrc.
Note that usually it is best to store data in local endian format, because that requires the least amount of byte swapping. However, if the table must be accessible with AIPS++ version 1.7 or before, big endian should be used.

When creating a new table, the table will normally reside on disk. It is, however, possible to specify that the table should be held in memory. In such a case the table is process specific, thus cannot be seen by other processes. Note that a memory table uses the MemoryStMan storage manager for all its stored columns, but it is still possible to use virtual columns as well.

Arguments

tablename	in	Name of table on disk Allowed:String—Bool Default: F
tabledesc	in	Table descriptor Allowed:Descriptor—Bool Default: F
nrow	in	Number of rows Allowed:Int Default: 0
readonly	in	Open Read-only? Allowed:Bool Default: T
lockoptions	in	locking to be used Allowed:String—Record Default: default
ack	in	Acknowledge creations, etc Allowed:Bool Default: T
dminfo	in	Data manager info for new table Allowed:Record Default: F
endian	in	Endian format of new table Allowed:String Default: aipsrc
memorytable	in	Hold new table in memory? Allowed:Bool Default: F
tableserver	in	Table handler to be used Allowed:Any tableserver Default: defaulttableserver
tablenote	in	note function to be used Allowed:Any note function Default: note
handle	in	attach to this table handle Allowed:for internal use only Default: F

Returns
table tool

Example

```
table1:=table("3C273XC1.MS");
table1.browse();
```

The first line opens an existing table 3C273XC1.MS, the second browses it using the browse tool function.

```
table2:=table("name", readonly=F, lockoptions='user', endian='local');
table2.lock();
table2.addRows();
table2.unlock();
```

In this example explicit user locking is used. The function lock is needed to acquire a (write) lock before the addRows is done. Thereafter the lock is released to give other processes the chance to operate on the table.
Note that releasing a lock implies flushing the table, so doing that very often can be quite expensive.

---

tablefromfits - Constructor

Package	**utility**
Module	**table**
Tool	**table**

Create an AIPS++ table from binary FITS

Synopsis
tablefromfits(tablename, fitsfile, whichhdu, storage, convention, readonly, lockoptions, ack, tableserver)

Description
Create a table from binary FITS format. This generates an AIPS table from the binary FITS table in the given HDU (header unit) of the FITS file. Note that other FITS formats (*e.g.* Image FITS and UVFITS) are read by other means.
It is possible to specify the storage manager to use for the table:
`standard` is the default storage manager.
`incremental` is efficient for slowly varying data.

Arguments

tablename	in	Name of table to be created Allowed:String
fitsfile	in	Name of FITS file to be read Allowed:String
whichhdu	in	Which HDU to read (0-relative to primary HDU i.e. 1 is the smallest valid value) Allowed:Int Default: 1
storage	in	Storage manager to use (standard or incremental) Allowed:String Default: standard
convention	in	Convention to use (SDFITS or none) Allowed:String Default: sdfits
readonly	in	Open Read-only? Allowed:Bool Default: T
lockoptions	in	locking to be used Allowed:String—Record Default: default
ack	in	Acknowledge creations, etc Allowed:Bool Default: T
tableserver	in	Table server to be used Allowed:Any tableserver Default: defaulttableserver

Returns
table tool

---

tablefromascii - Constructor

Package	**utility**
Module	**table**
Tool	**table**

Create an AIPS++ table from an ASCII format

Synopsis
tablefromascii(tablename, asciifile, headerfile, autoheader, autoshape, sep, commentmarker, firstline, lastline, readonly, lockoptions, ack, tableserver)

Description
Create a table from a file in ASCII format. Columnar data as well as table and column keywords may be specified.
Once the table is created from the ASCII data, it is opened in the specified mode and a table tool is returned.
The table columns are filled from a file containing the data values separated by a separator (one line per table row). The default separator is a blank. Blanks after the separator are ignored.
If a non-blank separator is used, values can be empty. Such values default to 0, empty string, or F depending on the data type. E.g. 1,,2, has 4 values of which the 2nd and 4th are empty and default to 0. Similarly if fewer values are given than needed, the missing values get the default value.
Either the data format can be explicitly specified or it can be found automatically. The former gives more control in ambiguous situations. Both scalar and array columns can be generated from the ASCII input. The format string determines the type and optional shape.
In automatic mode (`autoheader=T`) the first line of the ASCII data is analyzed to deduce the data types. Only the types I, D, and A can be recognized. A number without decimal point or exponent is I (integer), otherwise it is D (double). Any other string is A (string). Note that a number may contain a leading sign (+ or -). The `autoshape` argument can be used to specify if the input should be stored as multiple scalars (the default) or as a single array. In the latter case one axis in the shape can be defined as variable length by giving it the value 0. It means that the actual array shape in a row is determined by the number of values in the corresponding input line.
Columns get the names `Column1`, `Column2`, etc..
For example:

1. `autoshape=[]` (which is the default) means that all values are to be stored as scalar columns.

2. `autoshape=0` means that all values in a row are to be stored as a variable length vector.

3. `autoshape=10` defines a fixed length vector. If an input line contains less than 10 values, the vector is filled with default values. If more than 10 values, the latter values are ignored.

4. `autoshape=[5,0` defines a 2-dim array of which the 2nd axis is variable. Note that if an input line does not contain a multiple of 5 values, the array is filled with default values.

If the format of the table is explicitly specified, it has to be done either in the first two lines of the data file (named by the argument filename), or in a separate header file (named by the argument headerfile). In both forms, table keywords may also be specified before the column definitions. The column names and types can be described by two lines:

1. The first line contains the names of the columns. These names may be enclosed in quotes (either single or double).

2. The second line contains the data type and optionally the shape of each column. Valid types are:

   - S for Short data
   - I for Integer data
   - R for Real data
   - D for Double Precision data
   - X for Complex data (Real followed by Imaginary)
   - Z for Complex data (Amplitude then Phase)
   - DX for Double Precision Complex data (Real followed by Imaginary)
   - DZ for Double Precision Complex data (Amplitude then Phase)
   - A for ASCII data (a value must be enclosed in single or double quotes if it contains whitespace)
   - B for Boolean data (False are empty string, 0, or any string starting with F, f, N, or n).

If a column is an array, the shape has to be given after the data type without any whitespace. E.g. `I10` defines an integer vector of length 10. `A2,5` defines a 2-dim string array with shape [2,5]. Note that `I` is not the same as `I1` as the first one defines a scalar and the other one a vector with length 1. The last column can have one variable length axis denoted by the value 0. It "consumes" the remainder of the input line.
If the argument headerfile is set then the header information is read from that file instead of the first lines of the data file.
To give a simple example of the form where the header information is located at the top of the data file:

```
COLI   COLF   COLD       COLX        COLZ       COLS
 I      R      D          X           Z          A
1      1.1    1.11       1.12 1.13   1.14 1.15  Str1
10     11     12         13   14     15   16    ""
```

Note that a complex number consists of 2 numbers.
Also note that an empty string can be given.
Let us now give an example of a separate header file that one might use to get interferometer data into AIPS++:

```
U     V      W         TIME        ANT1       ANT2      DATA
R     R      R          D           I          I        X1,0
```

The data file would then look like:

```
124.011 54560.0  3477.1  43456789.0990    1      2        4.327 -0.1132
34561.0 45629.3  3900.5  43456789.0990    1      3        5.398 0.4521
```

Note that the DATA column is defined as a 2-dim array of 1 correlation and a variable number of channels, so the actual number of channels is determined by the input. In this example both rows will have 1 channel (note that a complex value contains 2 values).
Tables may have keywords in addition to the columns. The keywords are useful for holding information that is global to the entire table (such as author, revision, history, *etc,*).
The keywords in the header definitions must preceed the column descriptions. They must be enclosed between a line that starts with ".key..." and a line that starts with ".endkey..." (where ... can be anything). Between these two lines each line should contain the following as listed below. A table keywordset and column keywordsets can be specified. The latter can be specified by specifying the column name after the .keywords string.

- The keyword name, e.g., ANYKEY
- The datatype and optional shape of the keyword (cf. list of valid types above)
- The value or values for the keyword (the keyword may contain a scalar or an array of values). e.g., 3.14159 21.78945

Thus to continue the example above, one might wish to add keywords as follows:

```
.keywords
DATE        A  "97/1/16"
REVISION    D 2.01
AUTHOR      A "Tim Cornwell"
INSTRUMENT  A "VLA"
.endkeywords
.keywords TIME
UNIT A "s"
.endkeywords
U     V      W         TIME        ANT1       ANT2      DATA
R     R      R          D           I          I        X1,0
```

Similarly to the column format string, the keyword formats can also contain shape information. The only difference is that if no shape is given, a keyword can have multiple values (making it a vector).
It is possible to ignore comment lines in the header and data file by giving the `commentmarker`. It indicates that lines starting with the given marker are ignored. Note that the marker can be a regular expression (e.g. texttt' *//' tells that lines starting with // and optionally preceeded by blanks have to be ignored).

With the arguments `firstline` and `lastline` one can specify which lines have to be taken from the input file. A negative value means 1 for `firstline` or end-of-file for `lastline`. Note that if the headers and data are combined in one file, these line arguments apply to the whole file. If headers and data are in separate files, these line arguments apply to the data file only.

Also note that ignored comment lines are counted, thus are used to determine which lines are in the line range.

The number of rows is determined by the number of lines read from the data file.

Arguments

tablename	in	Name of table to be created Allowed:String
asciifile	in	Name of ASCII file to be read Allowed:String
headerfile	in	Name of an optional file defining the format Allowed:String Default: ”
autoheader	in	Determine header information automatically Allowed:Boolean Default: F
autoshape	in	Shape to be used if autoheader=T Allowed:Vector of Ints Default: []
sep	in	Value separator Allowed:String Default: ’ ’
commentmarker	in	Regex indicating comment line Allowed:String Default: ”
firstline	in	First line to use Allowed:Int Default: 1
lastline	in	Last line to use Allowed:Int Default: -1
readonly	in	Open Read-only? Allowed:Bool Default: T
lockoptions	in	locking to be used Allowed:String—Record Default: default
ack	in	Acknowledge creations, etc Allowed:Bool Default: T
tableserver	in	Table server to be used Allowed:Any tableserver Default: defaulttableserver

Returns
table tool

---

table.flush - Function

Package	**utility**
Module	**table**

Tool table

flush the current contents to disk

Synopsis
flush(recursive)

Description
Until a flush is performed, the results of all operations are not reflected in any change to the disk file. Hence you *must* do a flush to write the changes to disk.

Arguments

recursive	in	Flush all subtables recursively? Allowed:Bool Default: F

Returns
Bool

---

table.resync - Function

Package	**utility**
Module	**table**
Tool	**table**

resync the table tool with table file

Synopsis
resync()

Description
Acquiring a read or write lock automatically synchronizes the internals of the table tool with the actual contents of the table files. In this way different processes accessing the same table always use the same table data.
However, a table can be used without read locking. In that case the table tool internals are not synchronized automatically. The resync function offers a way

to do explicit synchronization. It is only useful if the table is opened with locking mode `autonoread` or `usernoread`.

Returns
Bool

---

table.close - Function

Package	**utility**
Module	**table**
Tool	**table**

close the table tool

Synopsis
close()

Description
First a flush is done, then the table is closed inside Glish and is no longer available for use.

Returns
Bool

---

table.copy - Function

Package	**utility**
Module	**table**
Tool	**table**

copy a table

Synopsis
copy(newtablename, deep, valuecopy, dminfo, endian, memorytable, returnobject, copynorows)

Description

Copy the table. All subtables are also copied. References to another table are preserved.

The argument `deep` determines how a reference table (i.e. the result of a query) is copied. By default a file copy is made, thus the resulting table still contains references and no actual data. If, however, `deep=T` is given, a deep copy is made which means that the actual data are copied. Also all subtables are copied.
Normally a plain table is copied by copying the files. However, if `deep=T` and `valuecopy=T` are given, a plain table is copied by copying all its values and subtables. This is useful to reorganize the tables, i.e. to regain file space that is wasted by frequent updates to a table.
The argument `dminfo` can be used to specify explicit data manager info for the columns in the new plain table. It can be used to change, for example, a storage manager from IncrStMan to StandardStMan. The `dminfo` is a record as returned by the getdminfo If `dminfo` is a non-empty record, it forces `valuecopy=T`.

By default all data are copied when a table and its subtables are copied. However, by giving `copynorows=T` the rows are not copied, thus only the table structure and keywords are copied.

The standard operation is make the copy to a plain table. It is, however, possible to copy to a memory table by giving `memorytable=T`.

The endian format for the newly created table can be specified. This is only meaningful if a deep copy is made to a plain table. The possible values are:
- big: big endian format (as used on e.g. SUN)
- little: little endian format (as used on e.g. PC)
- local: use the endian format of the machine being used
- aipsrc: use the endian format specified in aipsrc variable table.endianformat (which defaults to big).
The default is aipsrc.

Normally the `copy` function only copies the table and does not create a new table tool object. The user can do that by opening the newly created table in the standard way. However, it is possible to get an object back by using `returnobject=T`. An object is always returned if the copy is made to a memory table.

Arguments

newtablename	in	Name of newtable on disk Allowed:String
deep	in	Make a deep copy of a reference table? Allowed:Bool Default: F
valuecopy	in	Make a deep copy of any table? Allowed:Bool Default: F
dminfo	in	Data manager info for new table Allowed:Record Default: [=]
endian	in	Endian format of new table Allowed:String Default: aipsrc
memorytable	in	Hold new table in memory? Allowed:Bool Default: F
returnobject	in	Return a tool object for the new table Allowed:Bool Default: F
copynorows	in	Make no copy of the rows? Allowed:Bool Default: F

Returns
Bool

---

table.copyrows - Function

Package	**utility**
Module	**table**
Tool	**table**

copy rows from this table to another

Synopsis
copyrows(outtable, startrowin, startrowout, nrow)

Description
Copy rows from this table to another. By default all rows of this table are appended to the output table. It is possible though to control which rows are copied.

Rows are added to the output table as needed. Because no rows can be added to a reference table, it is only possible to overwrite existing rows in such tables. Only the data of columns existing in both tables will be copied. Thus by making a reference table consisting of a few columns, it is possible to copy those columns only.

Arguments

outtable	in	table object of output table Allowed:String
startrowin	in	First row to take from input table Allowed:Int Default: 1
startrowout	in	First row to write in output table Allowed:Int Default: -1 (=end)
nrow	in	Nr of rows to copy Allowed:Int Default: -1 (=all)

Returns
Bool

Example
This example appends rows to the table itself, thus doubles the number of rows.

```
t:=table('test.ms',readonly=F)
t.copyrows(t)
```

This example copies 10 rows of the selected subset of the MS to the beginning of the output MS.

```
t:=table("3C273XC1.MS")
tout:=table('test.ms',readonly=F)
t1:=t.query('ANTENNA1==0')
t1.copyrows(tout,nrow=10,startrowout=1)
```

---

table.done - Function

Package	**utility**
Module	**table**
Tool	**table**

end the table tool

Synopsis
done()

Description
End the table tool and free up all memory. It cannot be reused anymore.

---

table.iswritable - Function

Package	**utility**
Module	**table**
Tool	**table**

is the table writable?

Synopsis
iswritable()

Description
Test if the table is opened for write.
Note that the global function `tableiswritable(tablename)` tests if a table with the given name is writable.

Returns
Bool

---

table.endianformat - Function

Package	**utility**
Module	**table**
Tool	**table**

get the endian format used for this table

Synopsis
endianformat()

Description
Get the endian format used for this table. It returns a string with value 'big' or 'little'.

Returns
String

---

table.lock - Function

Package	**utility**
Module	**table**
Tool	**table**

acquire a lock on the table

Synopsis
lock(write, nattempts)

Description
Try to acquire a read or write lock on the table. Nothing will be done if the table is already correctly locked by this process. It is only needed when user locking is used. When the lock is acquired, the internal caches will be synchronized with the (possible changed) contents of the table.
It is possible to specify the number of attempts to do (1 per second) in case the table is locked by another process. The default 0 is trying indefinitely.

Arguments

write	in	Write lock? (F=read lock) Allowed:Bool Default: T
nattempts	in	Nr of attempts Allowed:Int Default: 0

Returns
Bool

---

table.unlock - Function

Package	**utility**
Module	**table**
Tool	**table**

unlock and flush the table

Synopsis
unlock()

Description
The table is flushed and the lock on the table is released. This function is only needed when user locking is used. However, it is also possible to use it with auto locking. In that case the lock will automatically be re-acquired before the next table operation.

Returns
Bool

---

table.datachanged - Function

Package	**utility**
Module	**table**
Tool	**table**

has data changed in table?

Synopsis
datachanged()

Description
This function tests if data in the table have changed (by another process) since the last call to this function.

Returns
Bool

---

table.haslock - Function

Package	**utility**
Module	**table**
Tool	**table**

has this process a lock on the table?

Synopsis
haslock(write)

Description
Has this process a read or write lock on the table?

Arguments

write	in	Has it a write lock? (F=read lock) Allowed:Bool Default: T

Returns
Bool

---

table.lockoptions - Function

Package	**utility**
Module	**table**
Tool	**table**

get the lock options used for this table

Synopsis
lockoptions()

Description
Get the lock options used for this table. It returns a record with the fields option, interval and maxwait. The record can be used as the lockoptions argument in the table constructor.

Returns
Record

---

table.ismultiused - Function

Package	**utility**
Module	**table**
Tool	**table**

is the table in use in another process?

Synopsis
ismultiused(checksubtables)

Description
Is the table still in use in another process? If so, the table cannot be deleted.

Arguments

checksubtables	in	check if subtables are multiused?) Allowed:Bool Default: F

Returns
Bool

---

table.browse - Function

Package	**utility**
Module	**table**
Tool	**table**

browse a table using a graphical browser

Synopsis
browse()

Description

To start the browser, the environment variable DISPLAY must be set.

Returns
Bool

---

table.name - Function

Package	**utility**
Module	**table**
Tool	**table**

return name on disk

Synopsis
name()

Description
This gives the name of the AIPS++ table on disk that this table tool corresponds to *i.e.* this is the name that the table was constructed from.

Returns
String

Example

```
  t:=table("3C273XC1.MS")
  t.name()
# 3C273XC1.MS
```

---

table.toascii - Function

Package	**utility**
Module	**table**
Tool	**table**

Write AIPS++ table into an ASCII format

Synopsis
toascii(asciifile, headerfile, columns, sep)

Description
Write a table into an ASCII format. Both columnar data (scalar) and table keywords may be written. The separator between values can be specified and defaults to a blank. Note that columns containing arrays will be ignored (after showing a warning).
The output format is that read by the constructor tablefromascii.
If the argument headerfile is set then the header information is written to that file instead of the first two lines of the data file.

Arguments

asciifile	in	Name of ASCII file to be written Allowed:String
headerfile	in	Name of an optional file defining the format Allowed:String Default: ''
columns	in	Names of columns to be written Allowed:Vector of strings Default: All
sep	in	Value separator Allowed:String Default: ' '

Returns
Bool

---

table.query - Function

Package	**utility**
Module	**table**
Tool	**table**

Make a table from a query

Synopsis
query(query, name, sortlist, columns)

Description
Make a table from a query applied to the current table. The query is the qualification part of the tablecommand. It is also possible to specify column(s) and/or expressions to sort on and to specify the columns to be contained in the output table. See the example below. A new table tool is returned. The new (reference) table can be given a name and will then be written to disk. Note that the resulting table is just a reference to the original table. One can make a deep copy of the query result using the copy function (see example).
If the arguments `query`, `sortlist`, and `columns` are empty strings (their default values), the taqlwidget will be started (if a GUI is available) before the query function is executed. In that way it is possible to let the user form the query interactively.

Arguments

query	in	Query string Allowed:String Default: ”
name	in	Name of resulting reference table Allowed:String Default: ”
sortlist	in	Sort string (one or more expressions separated by commas) Allowed:String Default: ”
columns	in	List of column names separated by commas Allowed:String Default: ”

Returns
table tool

Example

```
  t:=table("3C273XC1.MS");
  subt:=t.query("SPECTRAL_WINDOW_ID==1",
                sortlist="ARRAY_ID", columns="TIME, DATA, UVW");
  print subt.ncols();
# 3
  t.close();
  subt.copy ("3C273XC1_spw1.MS", T);
  subt.close();
```

From the original table corresponding to the disk file 3C273XC1.MS, only rows with SPECTRAL_WINDOW_ID equal to 1 are selected and sorted by ARRAY_ID. Only the columns TIME DATA UVW are written. Thereafter a deep copy of the result is made. The table query command is equivalent to:

```
subt:=tablecommand(paste('select TIME, DATA, UVW from 3C273XC1.MS',
                         'where SPECTRAL_WINDOW_ID==1',
                         'orderby ARRAY_ID'))
```

The sortlist argument can be used to sort in ascending or descending order (or a mix of them) on one or more columns. Default is ascending. It is also possible to remove duplicate values using the word NODUPLICATES at the beginning. E.g.:

```
sortlist='TIME desc'
sortlist='noduplicates ANTENNA1,ANTENNA2'
sortlist='ANTENNA1 desc, ANTENNA2 asc'
sortlist='desc ANTENNA1, ANTENNA2, TIME'
```

See Also
tablecommand

---

table.calc - Function

Package	**utility**
Module	**table**
Tool	**table**

Calculate an expression on a table

Synopsis
calc(expr)

Description
Get the result from the calculation of an expression on a table. using a CALC command given to the tablecommand function. The expression can be any expression that can be given in the WHERE clause of a SELECT expression (thus including subqueries). The given expression determines if the result is a scalar, a vector, or a record containing arrays. See the examples below.

Arguments

expr	in	Expression string Allowed:String

Returns
record, vector, or scalar

Example

```
t:=table("3C273XC1.MS");
t.calc ('DATA+1');
```

returns a (potentially enormous) record where a field contains the value of the expression for the row with that number. Note that it returns a record because for each row the expression results in an array. It should be clear that this example is useless. However, something like this could be useful for a column with (very) small arrays.

```
t:=table("3C273XC1.MS");
t.calc ('ntrue(FLAG)');
```

returns for each row the number of flags set. The result is a vector, because for each row the expression results in a scalar.

```
t:=table("3C273XC1.MS");
t.calc ('sum([select from $1 giving [ntrue(FLAG)]])');
```

returns the total number of flags set in the table (in a single scalar). It is the same as

```
sum(t.calc ('ntrue(FLAG)'));
```

The latter is easier to write, but sends (much) more data over the glish bus.

See Also
tablecommand

---

table.selectrows - Function

Package	**utility**
Module	**table**
Tool	**table**

Make a table from a selection of rows

Synopsis
selectrows(rownrs, name)

Description
Create a (reference) table containing a given subset of rows. It is, for instance, useful when a selection is done on another table containing the row numbers in the main table. It can be useful to apply the Glish function unique to those row numbers, otherwise the same row might be included multiple times (see example).
Note that usually such referencing row numbers are 0-based, while the selectrows function expects 1-based row numbers. It means that in such cases 1 should be added (see example).
It is possible to give a name to the resulting table. If given, the resulting table is made persistent with that table name. Otherwise the table is transient and disappears when closed or when Glish exits.
The rownumbers function returns a vector containing the row number in the main table for each row in the selection table. Thus given a row number vector `rownrs`, the following is always true.

```
rownrs == maintable.selectrows(rownrs).rownumbers()
```

However, it is not true when selectrows is used on a selection table. because `rownumbers` does not return the row number in that selection table but in the main table.
It means that one has to take great care when using `selectrows` on a selection table.

Arguments

rownrs	in	1-based Row Numbers Allowed:Vector of Ints
name	in	Name of resulting table Allowed:Any String Default: ''

Example

```
#   Do the query on the main table.
  maintable := table('SOMENAME');
  scantable := maintable.query(command);
```

```
#   Get the column containing the 0-based row numbers in the BACKEND table.
#   Make the row numbers unique and make them 1-based.
  backrows := unique(scantable.getcol('NS_GBT_BACKEND_ID')) + 1;
#   Form the table subset of the BACKEND table containing those rows.
  backtable := table('SOMENAME/GBT_BACKEND');
  scanback := backtable.selectrows(backrows);
#   Do something with that table.
  print scanback.nrows();
```

The last statement is equivalent to the query command:

```
  scanback := backtable.query("rownumber() in $backrows");
```

However, the query takes much, much, much more time to execute, especially for larger tables and larger row number vectors. Furthermore, the `backrows` variable need to be global to be able to substitute it in the query command.

Returns
table tool

---

table.putinfo - Function

Package	**utility**
Module	**table**
Tool	**table**

set the info record

Synopsis
putinfo(value)

Description
The info record contains information on the table. It is written by applications, and used by, for example, the catalog directory browser to determine what type of information is stored in a table.

Arguments

value	in	Info record Allowed:info record

Returns
Bool

Example

```
  include 'catalog.g'
  dc.whatis("MS")
# [type=Measurement Set, istable=T, subType=observed]
  dc.whatis("tcal")
# [type=Calibration, istable=T, subType=T Jones]
```

---

table.info - Function

Package	**utility**
Module	**table**
Tool	**table**

get the info record

Synopsis
info()

Description
The info record contains information on the table. It is written by applications, and used by, for example, the catalog directory browser to determine what type of information is stored in a table.

Arguments

Returns
Record

---

table.addreadmeline - Function

Package	**utility**
Module	**table**
Tool	**table**

add a readme line to the info record

Synopsis
addreadmeline(value)

Description
A readme line is part of the info record associated with a table. It is to inform the user, and is not used by any application directly.

Arguments

value	in	readme line Allowed:String

Returns
Bool

---

table.summary - Function

Package	**utility**
Module	**table**
Tool	**table**

summarize the contents of the table

Synopsis
summary(recurse)

Description
A (terse) summary of the table contents is sent to the defaultlogger.

Arguments

recurse	in	Summarize subtables recursively Allowed:Bool Default: F

Returns
Bool

Example

```
  table("tcal").summary()
# successful readonly open of table  tcal :  9 columns, 11 rows
# Table summary: tcal
# Shape: 9 columns by 11 rows
# Info: [type=Calibration, subType=T Jones, readme=]
# Table keywords: [Type=T Jones, Interval=30, DeltaT=1]
# Columns: StartTime StopTime Gain SolutionOK Fit FitWeight
# iSolutionOK iFit iFitWeight
```

---

table.colnames - Function

Package	**utility**
Module	**table**
Tool	**table**

return the names of the columns

Synopsis
colnames()

Description
The names of the columns in the table are returned as a vector of Strings.

Returns
Vector of Strings

Example

```
  table("tcal").colnames()
# StartTime StopTime Gain SolutionOK Fit FitWeight iSolutionOK iFit iFitWeight
```

---

table.rownumbers - Function

Package	**utility**
Module	**table**
Tool	**table**

return the row numbers in the (reference) table

Synopsis
rownumbers(tab, nbytes)

Description
This function can be useful after a selection or a sort. It returns the row numbers of the rows in this table with respect to the given table. If no table is given, the original table is used.
For example:

```
  t:=table('W53.MS')
  t1:=t.selectrows([1,3,5,7,9])
  t1.rownumbers(t)
# [1 3 5 7 9]
  t2:=t1.selectrows([2,5])
  t2.rownumbers(t1)
# [2 5]
  t2.rownumbers(t)
# [3 9]
  t2.rownumbers()
# [3 9]
```

The last statements show that the function returns the row numbers referring to the given table. Table t2 contains rows 2 and 5 in table t1, which are rows 3 and 9 in table t.
Note that when a table is opened using its name, that table can be a reference table. Thus in the example above the last 2 statements may give different results depending on the fact if W53.MS is a reference table or not.
The function should always be called with a table argument. The ability of omitting the argument is only present for backward compatibility.
The function can be useful to get the correct values from the result of a getcol or getcolslice on the original table.

Arguments

tab	in	Table to which the row numbers refer Allowed:Record Default: F
nbytes	in	Maximum cache size in bytes Allowed:Int

Returns
Vector of Ints

Example

```
  t:=table("W53.MS")
  t.nrows()
# 1920
  data:=t.getcolslice("Data", [1,1], [1,1])
  data::shape
# [1 1 1920]
  selt:=t.query("ANTENNA1==1")
  selt.nrows()
# 300
  subdata:=data[,,selt.rownumbers(t)]
  subdata::shape
# [300]
```

---

table.setmaxcachesize - Function

Package	**utility**
Module	**table**
Tool	**table**

set maximum cache size for column in the table

Synopsis
setmaxcachesize(columnname, nbytes)

Description
It can sometimes be useful to limit the size of the cache used by a column stored with the tiled storage manager. This function requires some more knowledge about the table system and is not meant for the casual user.

Arguments

columnname	in	Name of column Allowed:String
nbytes	in	Maximum cache size in bytes Allowed:Int

Returns
Bool

Example

```
  t:=table("W53.MS")
  t.nrows()
# 1920
  t.setmaxcachesize ("DATA", 4*1024*1024);
# T
```

---

table.isscalarcol - Function

Package	**utility**
Module	**table**
Tool	**table**

is the specified column scalar?

Synopsis
isscalarcol(columnname)

Description
A column may contain either scalars or arrays in each cell. This tool function tests if the specified column has scalar contents.

Arguments

columnname	in	Name of column Allowed:String

Returns
Bool

Example

```
  table("tcal").isscalarcol("StartTime")
# T
  table("tcal").isscalarcol("Gain")
# F
```

---

table.isvarcol - Function

Package	**utility**
Module	**table**
Tool	**table**

tell if column contains variable shaped arrays

Synopsis
isvarcol(columnname)

Description
This functions tells if the column contains variable shaped arrays. If so, the function `getvarcol` should be used to get the entire column. Otherwise `getcol` can be used.

Arguments

columnname	in	Name of column Allowed:String

Returns
Bool

---

table.coldatatype - Function

Package	**utility**
Module	**table**
Tool	**table**

return the column data type

Synopsis
coldatatype(columnname)

Description
A column may contain various data types. This tool function returns the type of the column as a string.

Arguments

columnname	in	Name of column Allowed:String

Returns
String

Example

```
  table("tcal").coldatatype("StartTime")
# double
  table("tcal").coldatatype("Gain")
# complex
```

---

table.colarraytype - Function

Package	**utility**
Module	**table**
Tool	**table**

return the column array type

Synopsis
colarraytype(columnname)

Description
The possible column array types are defined as:

FixedShape FixedShape means that the shape of the array must be the same in each cell of the column. If not given, the array shape may vary. Option Direct forces FixedShape.

Direct Direct means that the data is directly stored in the table. Direct forces option FixedShape. If not given, the array is indirect, which implies that the data will be stored in a separate file.

Arguments

columnname	in	Name of column Allowed:String

Returns
String

Example

```
  table("tcal").colarraytype("Gain")
# Direct,FixedShape
```

---

table.ncols - Function

Package	**utility**
Module	**table**
Tool	**table**

return number of columns

Synopsis
ncols()

Returns
Int

Example

```
  table("tcal").ncols()
# 9
```

---

table.nrows - Function

Package	**utility**
Module	**table**
Tool	**table**

return number of rows

Synopsis
nrows()

Description
Note that rows are numbered starting at 1.

Returns
Int

Example

```
  table("tcal").nrows()
# 11
```

---

table.addrows - Function

Package	**utility**
Module	**table**
Tool	**table**

add a specified number of rows

Synopsis
addrows(nrow)

Description
Rows can be added to the end of a table that was opened readonly=F. The new rows are empty.

Arguments

nrow	in	Number of rows to add Allowed:Int Default: 1

Returns
Bool

---

table.removerows - Function

Package	**utility**
Module	**table**
Tool	**table**

remove the specified rows

Synopsis
removerows(rownrs)

Description
Remove the row numbers specified in the vector from the table. It fails when the table does not support row removal.

Arguments

rownrs	in	Row numbers to remove Allowed:Int

Returns
Bool

---

table.addcols - Function

Package	**utility**
Module	**table**
Tool	**table**

add one or more columns

Synopsis
addcols(desc, dminfo)

Description
Columns can be added to a table that was opened readonly=F. The new columns will be filled with a default value (0 or blank).
For each column to be added a column description has to be setup using function tablecreatescalarcoldesc or tablecreatearraycoldesc. When multiple columns are used, they have to be combined in a single record using tablecreatedesc.
It is possible to specify data manager info in order to define a data manager (storage manager or virtual column engine) for the columns to be added.

Arguments

desc	in	Description of one or more columns Allowed:Descriptor
dminfo	in	Optional description data manager to use Allowed:Record Default: F

Returns
Bool

Example

```
  t:=table("mytable", readonly=F)
  dc3:=tablecreatescalarcoldesc('C3', 'a')
  dc4:=tablecreatescalarcoldesc('C4', as_float(0))
  dc5:=tablecreatearraycoldesc('C5', as_double(0), 2, [10,20])
  t.addcols(dc3)
# T
  t.addcols(tablecreatedesc(dc4, dc5))
# T
```

A single column can be added as such, but multiple columns have to be combined.

---

table.renamecol - Function

Package	**utility**
Module	**table**
Tool	**table**

rename a column

Synopsis
renamecol(oldname, newname)

Description
A column can be renamed in a table that was opened readonly=F.
However, renaming is not possible in a (reference) table resulting from a select or sort operation.

Arguments

oldname	in	name of column to be renamed Allowed:String
newname	in	new name of column Allowed:String

Returns
Bool

Example

```
  t:=table("mytable", readonly=F)
  t.renamecol ('colold', 'colnew')
# T
```

Column `colold` is renamed to `colnew`.

---

table.removecols - Function

Package	**utility**
Module	**table**
Tool	**table**

remove one or more columns

Synopsis
removecols(columnnames)

Description
Columns can be removed from a table that was opened readonly=F.
It may not always be possible to remove a column, because some data managers do not support column removal. However, if all columns of a data manager are removed, it will always succeed. It results in the removal of the entire data manager (and its possible files).
Note that function getdminfo can be used to find which columns are served by which data manager.

Arguments

columnnames	in	names of columns to be removed Allowed:Vector of Strings

Returns
Bool

Example

```
  t:=table("mytable", readonly=F)
  t.removecols ("col1 col2")
# T
```

Two columns are removed.

---

table.iscelldefined - Function

Package	**utility**
Module	**table**
Tool	**table**

test if a specific cell contains a value

Synopsis

iscelldefined(columnname, rownr)

Description
A column containing variable shaped arrays can have an empty cell (if no array has been put into it). This function tests if a cell is defined (thus is not empty). Note that a scalar column and a fixed shape array column cannot have empty cells.

Arguments

columnname	in	Name of column Allowed:String
rownr	in	Row number, starting at 1 Allowed:Int

Returns
Bool

---

table.getcell - Function

Package	**utility**
Module	**table**
Tool	**table**

get a specific cell

Synopsis
getcell(columnname, rownr)

Description
A cell is the value at one row in one column. It may be a scalar or an array.

Arguments

columnname	in	Name of column Allowed:String
rownr	in	Row number, starting at 1 Allowed:Int

Returns
Value

---

table.getcellslice - Function

Package	**utility**
Module	**table**
Tool	**table**

get a slice from a specific cell

Synopsis
getcellslice(columnname, rownr, blc, trc, incr)

Description
A cell is the value at one row in one column. It must be an array. The slice must be specified as blc, trc with an optional stride.
In blc and trc -1 can be used to indicate all values for a dimension (-1 in blc is equivalent to 1, so -1 is especially useful for trc).

Arguments

columnname	in	Name of column Allowed:String
rownr	in	Row number, starting at 1 Allowed:Int
blc	in	Bottom left corner (e.g. [1,1,1] is start of 3D array) Allowed:Vector of Ints
trc	in	Top right corner Allowed:Vector of Ints
incr	in	Stride (defaults to 1 for all axes) Allowed:Vector of Ints

Returns
Array

Example

```
  t:=table("W53.MS")
  data:=t.getcellslice("DATA", 1, [1,1], [2,1])
  data::shape
# [2 1]
```

---

table.getcol - Function

Package	**utility**
Module	**table**
Tool	**table**

get a specific column

Synopsis
getcol(columnname, startrow, nrow, rowincr)

Description
The entire column (or part of it) is returned. Warning: it might be big! The functions can only be used if all arrays in the column have the same shape. That is guaranteed for columns containing scalars or fixed shaped arrays. For columns containing variable shaped arrays it only succeeds if all those arrays happen to have the same shape.
Note that function `getvarcol` can be used to get a column of arbitrary shaped arrays, which also handles empty cells correctly. Function `isvarcol` tells if a column contains variable shaped arrays. shaped

Arguments

columnname	in	Name of column Allowed:String
startrow	in	First row to read (default 1) Allowed:Int
nrow	in	Number of rows to read (default -1 means till the end) Allowed:Int
rowincr	in	Increment in rows to read (default 1) Allowed:Int

Returns
Array

Example

```
  t:=table("tcal",readonly=F)
# successful read/write open of table  tcal :  9 columns, 11 rows
  gain:=t.getcol("Gain")
  gain::shape
# [2 2 28 1 11]
```

---

table.getvarcol - Function

Package	**utility**
Module	**table**
Tool	**table**

get a specific column (for variable arrays)

Synopsis
getvarcol(columnname, startrow, nrow, rowincr)

Description
Function `getcol` can only used if values in the column cells to get have the same shape. Function `getvarcol` addresses this limitation by returning the values as a record instead of an array. Each field in the record contains the value for a column cell. If the value is undefined (i.e. the cell does not contain a value), the unset value is put in the record. Each field name is the letter r followed by the row number. The length of the record is the number of rows to get.
Note that the function `isvarcol` tells if a column contains variable shaped arrays.

Arguments

columnname	in	Name of column Allowed:String
startrow	in	First row to read (default 1) Allowed:Int
nrow	in	Number of rows to read (default -1 means till the end) Allowed:Int
rowincr	in	Increment in rows to read (default 1) Allowed:Int

Returns
Record

Example

```
  t:=table("tcal",readonly=F)
# successful read/write open of table  tcal :  9 columns, 11 rows
  gain:=t.getvarcol("Gain")
  length(gain)
# 11
```

---

table.getcolslice - Function

Package	**utility**
Module	**table**
Tool	**table**

get a slice from a specific column

Synopsis
getcolslice(columnname, blc, trc, incr, startrow, nrow, rowincr)

Description
A slice from the entire column (or part of it) is returned. Warning: it might be big!
In blc and trc -1 can be used to indicate all values for a dimension (-1 in blc is equivalent to 1, so -1 is especially useful for trc). Note that blc and trc should not contain the row number, only the blc and trc of the arrays in the column.

Arguments

columnname	in	Name of column Allowed:String
blc	in	Bottom left corner (e.g. [1,1,1] is start of 3D array) Allowed:Vector of Ints
trc	in	Top right corner Allowed:Vector of Ints
incr	in	Stride (defaults to 1 for all axes) Allowed:Vector of Ints
startrow	in	First row to read (default 1) Allowed:Int
nrow	in	Number of rows to read (default -1 means till the end) Allowed:Int
rowincr	in	Increment in rows to read (default 1) Allowed:Int

Returns
Array

Example

```
  t:=table("W53.MS")
  data:=t.getcolslice("DATA", [1,1], [2,1])
  data::shape
# [2 1 1920]
```

---

table.putcell - Function

Package	**utility**
Module	**table**
Tool	**table**

put a specific cell

Synopsis
putcell(columnname, rownr, value)

Description
A cell is the the value at one row in one column. It may be a scalar or an array.

Arguments

columnname	in	Name of column Allowed:String
rownr	in	Row number(s) (1-relative) Allowed:Vector of Ints
value	in	Value

Returns
Bool

---

table.putcellslice - Function

Package	**utility**
Module	**table**
Tool	**table**

put a slice into a specific cell

Synopsis
putcellslice(columnname, rownr, value, blc, trc, incr)

Description
A cell is the value at one row in one column. It must be an array. The slice must be specified as blc, trc with an optional stride.
In blc and trc -1 can be used to indicate all values for a dimension (-1 in blc is equivalent to 1, so -1 is especially useful for trc).

Arguments

columnname	in	Name of column Allowed:String
rownr	in	Row number, starting at 1 Allowed:Int
value	in	Value
blc	in	Bottom left corner (e.g. [1,1,1] is start of 3D array) Allowed:Vector of Ints
trc	in	Top right corner Allowed:Vector of Ints
incr	in	Stride (defaults to 1 for all axes) Allowed:Vector of Ints

Returns
Bool

---

table.putcol - Function

Package	**utility**
Module	**table**
Tool	**table**

put a specific column

Synopsis
putcol(columnname, value, startrow, nrow, rowincr)

Arguments

columnname	in	Name of column Allowed:String
value	in	Array
startrow	in	First row to put (default 1) Allowed:Int
nrow	in	Number of rows to put (default -1 means till the end) Allowed:Int
rowincr	in	Increment in rows to put (default 1) Allowed:Int

Returns
Bool

Example

```
  t:=table("tcal",readonly=F)
# successful read/write open of table  tcal :  9 columns, 11 rows
  gain:=t.getcol("Gain")
  gain[1,1,1,6,]:=1+0i
  t.putcol("Gain", gain)
  t.flush()
```

---

table.putvarcol - Function

Package	**utility**
Module	**table**
Tool	**table**

put a specific column (for variable arrays)

Synopsis
putvarcol(columnname, value, startrow, nrow, rowincr)

Description

`putcol` can only used if values in the column cells to put have the same shape. `putvarcol` addresses this limitation by passing the values as a record instead of an array. Each field in the record contains the value for a column cell. So the length of the record has to match the number of rows to put. If a value is the unset value, no put is done for that row.

Arguments

columnname	in	Name of column Allowed:String
value	in	Record with values
startrow	in	First row to put (default 1) Allowed:Int
nrow	in	Number of rows to put (default -1 means till the end) Allowed:Int
rowincr	in	Increment in rows to put (default 1) Allowed:Int

Returns
Bool

Example

```
  t:=table("tcal",readonly=F)
# successful read/write open of table  tcal :  9 columns, 11 rows
  gain:=t.getvarcol("DATA", 1, 10)
  t.putvarcol("Gain", gain, 11, 10)
  t.flush()
```

This example copies the values from row 1-10 to row 11-20.

---

table.putcolslice - Function

Package	**utility**
Module	**table**
Tool	**table**

put a slice into a specific column

Synopsis
putcolslice(columnname, value, blc, trc, incr, startrow, nrow, rowincr)

Description

In blc and trc -1 can be used to indicate all values for a dimension (-1 in blc is equivalent to 1, so -1 is especially useful for trc). Note that blc and trc should not contain the row number, only the blc and trc of the arrays in the column.

Arguments

columnname	in	Name of column Allowed:String
value	in	Array
blc	in	Bottom left corner (e.g. [1,1,1] is start of 3D array) Allowed:Vector of Ints
trc	in	Top right corner Allowed:Vector of Ints
incr	in	Stride (defaults to 1 for all axes) Allowed:Vector of Ints
startrow	in	First row to put (default 1) Allowed:Int
nrow	in	Number of rows to put (default -1 means till the end) Allowed:Int
rowincr	in	Increment in rows to put (default 1) Allowed:Int

Returns
Bool

Example

```
  t:=table("tcal",readonly=F)
# successful read/write open of table  tcal :  9 columns, 11 rows
  gain:=t.getcolslice("Gain", [1,1,1,6,1], [1,1,1,6,20])
  gain:=1+0i
  t.putcolslice("Gain", gain, [1,1,1,6,1], [1,1,1,6,20])
  t.flush()
```

---

table.getcolshapestring - Function

Package	**utility**
Module	**table**
Tool	**table**

get shape of arrays in a specific column

Synopsis
getcolshapestring(columnname, startrow, nrow, rowincr)

Description
The shapes of the arrays in the entire column (or part of it) are returned as strings like [20,3]. When the column contains fixed shaped arrays, a single string is returned. Otherwise a vector of strings is returned.

Arguments

columnname	in	Name of column Allowed:String
startrow	in	First row to read (default 1) Allowed:Int
nrow	in	Number of rows to read (default -1 means till the end) Allowed:Int
rowincr	in	Increment in rows to read (default 1) Allowed:Int

Returns
Array of Strings

Example

```
  t:=table("tcal",readonly=F)
# successful read/write open of table  tcal :  9 columns, 11 rows
  shapes:=t.getcolshapestring("Gain")
```

---

table.getkeyword - Function

Package	**utility**
Module	**table**
Tool	**table**

get value of specific table keyword

Synopsis
getkeyword(keyword)

Description
The value of the given table keyword is returned. The value can be of any type, including a record and a table.
If a keyword is a table, its value is returned as a string containing the table name prefixed by 'Table: '. That prefix is automatically removed by table functions like `open` and `tableexists`.
It is possible that the value of a keyword is a record itself (arbitrarily deeply nested). A field in such a subrecord can be read by separating the name with dots.

Arguments

keyword	in	Name or seqnr of keyword Allowed:String—Int

Returns
Value

Example

```
  t:=table('tcal')
# successful readonly open of table  tcal :  9 columns, 33 rows
  t.getkeyword('Interval')
# 30
  t.getkeyword('rec.fld')     # get field from a record
# 3.14
```

---

table.getkeywords - Function

Package	**utility**
Module	**table**
Tool	**table**

get values of all table keywords

Synopsis
getkeywords()

Description
The values of all table keywords are returned. The values can be of any type, including a record and a table.
If a keyword is a table, its value is returned as a string containing the table name prefixed by 'Table: '. That prefix is automatically removed by table functions like `open` and `tableexists`.

Returns
Record

Example

```
  t:=table('tcal')
# successful readonly open of table  tcal :  9 columns, 33 rows
  t.getkeywords()
# [Type=T Jones, Interval=30, DeltaT=1]
```

---

table.getcolkeyword - Function

Package	**utility**
Module	**table**
Tool	**table**

get value of specific column keyword

Synopsis
getcolkeyword(columnname, keyword)

Description
The value of the given column keyword is returned. The value can be of any type, including a record and a table.
If a keyword is a table, its value is returned as a string containing the table name prefixed by 'Table: '. That prefix is automatically removed by table functions like `open` and `tableexists`.
It is possible that the value of a keyword is a record itself (arbitrarily deeply nested). A field in such a subrecord can be read by separating the name with dots.

Arguments

columnname	in	Name of column Allowed:String
keyword	in	Name or seqnr of keyword Allowed:String—Int

Returns
Value

Example

```
  t:=table("MS")
# successful readonly open of table  MS :  22 columns, 7669 rows
  t.getcolkeyword("UVW", "UNIT")
# m
```

---

table.getcolkeywords - Function

Package	**utility**
Module	**table**
Tool	**table**

get values of all keywords for a column

Synopsis
getcolkeywords(columnname)

Description
The values of all keywords for the given column are returned. The values can be of any type, including a record and a table.
If a keyword is a table, its value is returned as a string containing the table name prefixed by 'Table: '. That prefix is automatically removed by table functions like `open` and `tableexists`.

Arguments

columnname	in	Name of column Allowed:String

Returns
Record

Example

```
  t:=table("MS")
# successful readonly open of table  MS :  22 columns, 7669 rows
  t.getcolkeywords("UVW")
# [UNIT=m, MEASURE_TYPE=UVW, MEASURE_REFERENCE=, _ForwardColumn_TableName=MS]
```

---

table.putkeyword - Function

Package	**utility**
Module	**table**
Tool	**table**

put a specific table keyword

Synopsis
putkeyword(keyword, value, makesubrecord)

Description
Put a table keyword. The value of the keyword can be a scalar or an array of any type or it can be a record.
It is possible to define a keyword holding a subtable. In that case a special string containing the name of the subtable will be passed to the table client.
It is possible that the value of a keyword is a record itself (arbitrarily deeply nested). A field in such a subrecord can be written by separating the name with dots. If a subrecord does not exist, an error is returned unless `makesubrecord=T` is given. In such a case intermediate records are created when needed.

Arguments

keyword	in	Name or seqnr of keyword Allowed:String—Int
value	in	Value of keyword
makesubrecord	in	Create intermediate records Allowed:Bool Default: F

Returns
Bool

Example

```
  t:=table("MS", readonly=F)
  t.putkeyword("VERSION", "1.66")
# T
#      define ANTENNA subtable
  t.putkeyword("ANTENNA", 'Table: MS/ANTENNA')
  t.flush()
# T
#      write a field in a record and create subrecords when needed
  t.putkeyword("REC.SUB.FLD", "val", T)
# T
#      write a keyword with a record value
  t.putkeyword("REC", [SUB=[FLD="val"]])
# T
```

Note that the last example does the same as the previous one (assuming that `REC` does not exist yet with other fields).

---

table.putkeywords - Function

Package	**utility**
Module	**table**
Tool	**table**

put multiple table keywords

Synopsis
putkeywords(value)

Description
Put multiple table keywords. All fields in the given record are put as table keywords. The value of each field can be a scalar or an array of any type or it can be a record.
It is also possible to define a keyword holding a subtable. This can be done by giving the keyword a string value consisting of the subtable name prefixed by 'Table: '. Remember to use single quotes, otherwise glish turns it into a string vector.

Arguments

value	in	Record of keyword=value pairs

Returns
Bool

Example

```
  t:=table('tcal', readonly=F)
  kw:=t.getkeywords()
  print kw
# [Type=T Jones, Interval=30, DeltaT=1]
  kw.Interval:=45
  t.putkeywords(kw)
# T
  t.flush()
# T
```

---

table.putcolkeyword - Function

Package	**utility**
Module	**table**
Tool	**table**

put a specific keyword for a column

Synopsis
putcolkeyword(columnname, keyword, value)

Description
Put a keyword in the given column. The value of the keyword can be a scalar or an array of any type or it can be a record.
It is possible to define a keyword holding a subtable. In that case a special string containing the name of the subtable will be passed to the table client.
It is possible that the value of a keyword is a record itself (arbitrarily deeply nested). A field in such a subrecord can be written by separating the name with dots. If a subrecord does not exist, an error is returned unless `makesubrecord=T` is given. In such a case intermediate records are created when needed.

Arguments

columnname	in	Name of column Allowed:String
keyword	in	Name or seqnr of keyword Allowed:String—Int
value	in	Value of keyword

Returns
Bool

Example

```
  t:=table("MS", readonly=F)
  t.putcolkeyword("UVW", "MEASURE_UNIT", "m")
# T
  t.flush()
# T
```

---

table.putcolkeywords - Function

Package	**utility**
Module	**table**
Tool	**table**

put multiple keywords for a column

Synopsis
putcolkeywords(columnname, value)

Description
Put multiple keywords in the given column. All fields in the given record are put as column keywords. The value of each field can be a scalar or an array of any type or it can be a record.
It is also possible to define a keyword holding a subtable. This can be done by giving the keyword a string value consisting of the subtable name prefixed by 'Table: '. Remember to use single quotes, otherwise glish turns it into a string vector.

Arguments

columnname	in	Name of column Allowed:String
value	in	Record of keyword=value pairs

Returns
Bool

Example

```
  t:=table("MS", readonly=F)
  t.getcolkeywords("UVW")
  kw:=[UNIT=m, MEASURE_TYPE=UVW, MEASURE_REFERENCE=, _ForwardColumn_TableName=MS]
  kw.MEASURE_REFERENCE='B1950'
  t.putcolkeywords(kw)
# T
```

table.removekeyword - Function

Package	**utility**
Module	**table**
Tool	**table**

remove a specific table keyword

Synopsis
removekeyword(keyword)

Arguments

keyword	in	Name or seqnr of keyword Allowed:String—Int

Returns
Bool

Example

```
  t:=table("MS", readonly=F)
  t.removekeyword("VERSION")
# T
  t.flush()
# T
```

---

table.removecolkeyword - Function

Package	**utility**
Module	**table**
Tool	**table**

remove a specific keyword for a column

Synopsis
removecolkeyword(columnname, keyword)

Arguments

columnname	in	Name of column Allowed:String
keyword	in	Name or seqnr of keyword Allowed:String—Int

Returns
Bool

Example

```
  t:=table("MS", readonly=F)
  t.removecolkeyword("UVW", "MEASURE_UNIT")
# T
  t.flush()
# T
```

---

table.getdminfo - Function

Package	**utility**
Module	**table**
Tool	**table**

get the info about data managers

Synopsis
getdminfo()

Description
This function returns the types and names of the data managers used. For each data manager it also returns the names of the columns served by it. The information is returned as a record containing a subrecord for each data manager. Each subrecord contains the fields TYPE, NAME and COLUMNS.

Returns
record

Example

```
  t1:=table('/aips++/code/aips/implement/Tables/test/tTable_2.data_v0')
# successful readonly open of default-locked table
#   /aips++/code/aips/implement/Tables/test/tTable_2.data_v0:
#   9 columns, 10 rows
  t.getdminfo()
# [*1=[TYPE=StandardStMan, NAME=SSM, COLUMNS=ac ad ag arr1 arr2 arr3],
#  *2=[TYPE=StManAipsIO, NAME=, COLUMNS=ab],
#  *3=[TYPE=IncrementalStMan, NAME=ISM, COLUMNS=ae af]]
```

This shows that the table uses 3 storage managers.

---

table.keywordnames - Function

Package	**utility**
Module	**table**
Tool	**table**

get the names of all table keywords

Synopsis
keywordnames()

Description
This function returns a vector of strings containing the names of all table keywords.

Returns
Vector of Strings

---

table.fieldnames - Function

Package	**utility**
Module	**table**
Tool	**table**

get the names of fields in a table keyword

Synopsis
fieldnames(keyword)

Description
This function returns a vector of strings containing the names of all fields in the given table keyword. It is only valid if the keyword value is a record. If no keyword name is given, the names of all table keywords are returned.

Arguments

keyword	keyword name Allowed:String Default: ”

Returns
Vector of Strings

---

table.colkeywordnames - Function

Package	**utility**
Module	**table**
Tool	**table**

get the names of all keywords in a column

Synopsis
colkeywordnames(columnname)

Description
This function returns a vector of strings containing the names of all keywords in the column with the given name..

Arguments

columnname	column name Allowed:String

Returns
Vector of Strings

table.colfieldnames - Function

Package	**utility**
Module	**table**
Tool	**table**

get the names of fields in a keyword in a column

Synopsis
colfieldnames(columnname, keyword)

Description
This function returns a vector of strings containing the names of all fields in the given keyword in the given column. It is only valid if the keyword value is a record.
If no keyword name is given, the names of all keywords in the column are returned.

Arguments

columnname	column name Allowed:String
keyword	keyword name Allowed:String Default: ”

Returns
Vector of Strings

table.getdesc - Function

Package	**utility**
Module	**table**
Tool	**table**

get the table description

Synopsis

getdesc(actual)

Description

The table description is a Glish record that contains a complete description of the layout of the table (except for the number of rows). It can be used as a template in the creation of another table using the table constructor.
By default the actual table description is returned (thus telling the actual shapes and data managers used). It is also possible to get the table description used when creating the table.

Arguments

actual	in	actual table description? Allowed:Bool Default: T

Returns

Table description record

Example

```
  t:=table('tcal')
# successful readonly open of table  tcal :  9 columns, 33 rows
  t.getdesc()
# [StartTime=[valueType=double, dataManagerType=StandardStMan,
#     dataManagerGroup=StandardStman, option=5, maxlen=0, comment=],
#  StopTime=[valueType=double, dataManagerType=StandardStman,
#      dataManagerGroup=StandardStMan, option=5, maxlen=0, comment=],
#  Gain=[valueType=complex, dataManagerType=StandardStMan,
#      dataManagerGroup=StandardStMan, option=5, maxlen=0, comment=,
#      ndim=4, shape=[2 2 28 1] ],
#  SolutionOK=[valueType=boolean, dataManagerType=StandardStman,
#      dataManagerGroup=StandardStMan, option=5, maxlen=0, comment=],
#  Fit=[valueType=float, dataManagerType=StandardStMan,
#      dataManagerGroup=StandardStMan, option=5, maxlen=0, comment=],
#  FitWeight=[valueType=float, dataManagerType=StandardStMan,
#      dataManagerGroup=StandardStman, option=5, maxlen=0, comment=],
#  iSolutionOK=[valueType=boolean, dataManagerType=StandardStman,
#      dataManagerGroup=StandardStMan, option=5, maxlen=0, comment=,
#      ndim=2, shape=[28 1] ],
#  iFit=[valueType=float, dataManagerType=StandardStMan,
#      dataManagerGroup=StandardStman, option=5, maxlen=0, comment=,
```

```
#      ndim=2, shape=[28 1] ],
#  iFitWeight=[valueType=float, dataManagerType=StandardStMan,
#      dataManagerGroup=StandardStMan, option=5, maxlen=0, comment=,
#      ndim=2, shape=[28 1] ]]
```

---

table.getcoldesc - Function

Package	**utility**
Module	**table**
Tool	**table**

get the description of a specific column

Synopsis
getcoldesc(columnname)

Description
The column description is a Glish record that contains a complete description of the layout of a specified column (except for the number of rows). It can be used to construct a table description.

Arguments

columnname	in	Name of column Allowed:String

Returns
Column description record

Example

```
  t:=table('tcal')
# successful readonly open of table  tcal :  9 columns, 33 rows
  t.getcoldesc('Gain')
# [valueType=complex, dataManagerType=StandardStMan, dataManagerGroup=StandardStMan, option=5, comment=, ndim=4, shape=[2 2 28 1] ]
```

---

table.ok - Function

Package	**utility**
Module	**table**
Tool	**table**

Is the table tool ok?

Synopsis
ok()

Description
Perform a number of sanity checks and return T if ok. Failure (returning F) is a sign of a bug.

Returns
Bool

1.7.2 tablecolumn - Tool

Package	**utility**
Module	**table**

Allows access to a column in a table

Description
This is a convenience tool for access to a table column. It offers the same functionality as the column access functions in the table tool (e.g. getcell). The advantage is that the column name does not need to be specified each time.

Example

```
include "table.g"
vis:=table("3C273XC1.MS", readonly=T);
vistimc:=tablecolumn(vis,"TIME");
vistimc.getcol();
vistimc.done();
```

Constructors

tablecolumn	Construct table column tool

Functions

arraytype	return the column array type
close	close the tablecolumn tool
datatype	return the column data type
done	end the tablecolumn tool
fieldnames	get the names of fields in a keyword in a column
getcell	get a specific cell
getcellslice	get a slice from a specific cell
getcol	get a specific column
getcolslice	get a slice from a specific column
getdesc	get the description of a specific column
getkeyword	get value of specific column keyword
getkeywords	get values of all keywords for a column
getshapestring	get shape of arrays in a specific column
getvarcol	get a specific column (for variable arrays)
iscelldefined	test if a specific cell contains a value
isscalar	is the specified column scalar?
isvar	tell if column contains variable shaped arrays
keywordnames	get the names of all keywords in a column
makeindex	make a table index tool
makeiter	make a table iterator tool
name	return the column name
nrows	return number of rows
putcell	put a specific cell
putcellslice	put a slice into a specific cell
putcol	put a specific column
putcolslice	put a slice into a specific column
putkeyword	put a specific keyword for a column
putkeywords	put multiple keywords for a column
putvarcol	put a specific column (for variable arrays)
removekeyword	remove a specific keyword for a column
table	return the table tool

tablecolumn - Constructor

Package	**utility**
Module	**table**
Tool	**tablecolumn**

Construct table column tool

Synopsis
tablecolumn(tab, column)

Arguments

tab	in	table to be accessed Allowed:table tool
column	in	column name Allowed:String

---

tablecolumn.table - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

return the table tool

Synopsis
table()

Description
Return the table tool used with this tablecolumn tool.

Arguments

Returns
table tool

---

tablecolumn.name - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

return the column name

Synopsis
name()

Description
Return the name of the column used with this tablecolumn tool.

Arguments

Returns
String

---

tablecolumn.isscalar - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

is the specified column scalar?

Synopsis
isscalar()

Arguments

Returns
Bool

---

tablecolumn.isvar - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

tell if column contains variable shaped arrays

Synopsis
isvar()

Arguments

Returns
Bool

---

tablecolumn.datatype - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

return the column data type

Synopsis
datatype()

Arguments

Returns
String

---

tablecolumn.arraytype - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

return the column array type

Synopsis
arraytype()

Arguments

Returns
String

---

tablecolumn.nrows - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

return number of rows

Synopsis
nrows()

Returns
Int

---

tablecolumn.getdesc - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

get the description of a specific column

Synopsis
getdesc()

Arguments

Returns
Column description record

---

tablecolumn.iscelldefined - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

test if a specific cell contains a value

Synopsis
iscelldefined(rownr)

Arguments

rownr	in	Row number, starting at 1 Allowed:Int

Returns
Bool

---

tablecolumn.getcell - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

get a specific cell

Synopsis
getcell(rownr)

Arguments

rownr	in	Row number, starting at 1 Allowed:Int

Returns
Value

---

tablecolumn.getcellslice - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

get a slice from a specific cell

Synopsis
getcellslice(rownr, blc, trc, incr)

Arguments

rownr	in	Row number, starting at 1 Allowed:Int
blc	in	Bottom left corner (e.g. [1,1,1] is start of 3D array) Allowed:Vector of Ints
trc	in	Top right corner Allowed:Vector of Ints
incr	in	Stride (defaults to 1 for all axes) Allowed:Vector of Ints

Returns
Array

---

tablecolumn.getcol - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

get a specific column

Synopsis
getcol(startrow, nrow, rowincr)

Arguments

startrow	in	First row to read (default 1) Allowed:Int
nrow	in	Number of rows to read (default -1 means till the end) Allowed:Int
rowincr	in	Increment in rows to read (default 1) Allowed:Int

Returns
Array

---

tablecolumn.getvarcol - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

get a specific column (for variable arrays)

Synopsis
getvarcol(startrow, nrow, rowincr)

Arguments

startrow	in	First row to read (default 1) Allowed:Int
nrow	in	Number of rows to read (default -1 means till the end) Allowed:Int
rowincr	in	Increment in rows to read (default 1) Allowed:Int

Returns
Record

---

tablecolumn.getcolslice - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

get a slice from a specific column

Synopsis
getcolslice(blc, trc, incr, startrow, nrow, rowincr)

Arguments

blc	in	Bottom left corner (e.g. [1,1,1] is start of 3D array) Allowed:Vector of Ints
trc	in	Top right corner Allowed:Vector of Ints
incr	in	Stride (defaults to 1 for all axes) Allowed:Vector of Ints
startrow	in	First row to read (default 1) Allowed:Int
nrow	in	Number of rows to read (default -1 means till the end) Allowed:Int
rowincr	in	Increment in rows to read (default 1) Allowed:Int

Returns
Array

---

tablecolumn.putcell - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

put a specific cell

Synopsis
putcell(rownr, value)

Arguments

rownr	in	Row number(s) (1-relative) Allowed:Vector of Ints
value	in	Value

Returns
Bool

---

tablecolumn.putcellslice - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

put a slice into a specific cell

Synopsis
putcellslice(rownr, value, blc, trc, incr)

Arguments

rownr	in	Row number, starting at 1 Allowed:Int
value	in	Value
blc	in	Bottom left corner (e.g. [1,1,1] is start of 3D array) Allowed:Vector of Ints
trc	in	Top right corner Allowed:Vector of Ints
incr	in	Stride (defaults to 1 for all axes) Allowed:Vector of Ints

Returns
Bool

---

tablecolumn.putcol - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

put a specific column

Synopsis
putcol(value, startrow, nrow, rowincr)

Arguments

value	in	Array
startrow	in	First row to put (default 1) Allowed:Int
nrow	in	Number of rows to put (default -1 means till the end) Allowed:Int
rowincr	in	Increment in rows to put (default 1) Allowed:Int

Returns
Bool

---

tablecolumn.putvarcol - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

put a specific column (for variable arrays)

Synopsis
putvarcol(value, startrow, nrow, rowincr)

Arguments

value	in	Record with values
startrow	in	First row to put (default 1) Allowed:Int
nrow	in	Number of rows to put (default -1 means till the end) Allowed:Int
rowincr	in	Increment in rows to put (default 1) Allowed:Int

Returns

Bool

---

tablecolumn.putcolslice - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

put a slice into a specific column

Synopsis
putcolslice(value, blc, trc, incr, startrow, nrow, rowincr)

Arguments

value	in	Array
blc	in	Bottom left corner (e.g. [1,1,1] is start of 3D array) Allowed:Vector of Ints
trc	in	Top right corner Allowed:Vector of Ints
incr	in	Stride (defaults to 1 for all axes) Allowed:Vector of Ints
startrow	in	First row to put (default 1) Allowed:Int
nrow	in	Number of rows to put (default -1 means till the end) Allowed:Int
rowincr	in	Increment in rows to put (default 1) Allowed:Int

Returns
Bool

---

tablecolumn.getshapestring - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

get shape of arrays in a specific column

Synopsis
getshapestring(startrow, nrow, rowincr)

Arguments

startrow	in	First row to read (default 1) Allowed:Int
nrow	in	Number of rows to read (default -1 means till the end) Allowed:Int
rowincr	in	Increment in rows to read (default 1) Allowed:Int

Returns
Array of Strings

---

tablecolumn.getkeyword - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

get value of specific column keyword

Synopsis
getkeyword(keyword)

Arguments

keyword	in	Name or seqnr of keyword Allowed:String—Int

Returns
Value

---

tablecolumn.getkeywords - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

get values of all keywords for a column

Synopsis
getkeywords()

Arguments

Returns
Record

---

tablecolumn.putkeyword - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

put a specific keyword for a column

Synopsis
putkeyword(keyword, value)

Arguments

keyword	in	Name or seqnr of keyword Allowed:String—Int
value	in	Value of keyword

Returns
Bool

---

tablecolumn.putkeywords - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

put multiple keywords for a column

Synopsis
putkeywords(value)

Arguments

value	in	Record of keyword=value pairs

Returns
Bool

---

tablecolumn.removekeyword - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

remove a specific keyword for a column

Synopsis
removekeyword(keyword)

Arguments

keyword	in	Name or seqnr of keyword Allowed:String—Int

Returns
Bool

---

tablecolumn.keywordnames - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

get the names of all keywords in a column

Synopsis
keywordnames()

Arguments

Returns
Vector of Strings

---

tablecolumn.fieldnames - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

get the names of fields in a keyword in a column

Synopsis
fieldnames(keyword)

Arguments

keyword	keyword name Allowed:String Default: ”

Returns
Vector of Strings

---

tablecolumn.makeiter - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

make a table iterator tool

Synopsis
makeiter(order, sort)

Description
Create a tableiterator tool based on this column.

Arguments

order	in	Order when sorting the table Allowed:String Default: ”
sort	in	Sort the table first? Allowed:Bool,String Default: T

Returns
tableiterator tool

---

tablecolumn.makeindex - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

make a table index tool

Synopsis
makeindex(sort)

Description
Create a tableindex tool based on this column.

Arguments

sort	in	Sort the table first? Allowed:Bool,String Default: T

Returns
tableindex tool

---

tablecolumn.close - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

close the tablecolumn tool

Synopsis
close()

Returns
Bool

---

tablecolumn.done - Function

Package	**utility**
Module	**table**
Tool	**tablecolumn**

end the tablecolumn tool

Synopsis
done()

Description
End the tablecolumn tool and free up all memory. It cannot be reused anymore.

Returns
Bool

1.7.3 tablerow - Tool

Package	**utility**
Module	**table**

Allows access to rows of a table tool

Description
A table row is a record containing the values of a single row for all or optionally just specified columns from a table. In constructing the tablerow tool, one can specify which columns are to be included or excluded. Tool functions then allow getting and putting of table rows.

Example

```
include "table.g"
vis:=table("3C273XC1.MS", readonly=T);
vis.summary();
columns:=["TIME", "UVW", "DATA"];
visrow:=tablerow(vis, columns);
nrows:=vis.nrows();
for (irow in 1:nrows) print irow, visrow.get(irow);
visrow.close();
```

Constructors

tablerow	Construct table row tool

Functions

close	Close the tablerow
done	End the tablerow tool
get	Get a table row
put	Put a table row
set	Set table and columns to be accessed

tablerow - Constructor

Package	**utility**
Module	**table**
Tool	**tablerow**

Construct table row tool

Synopsis
tablerow(tab, columns, exclude)

Arguments

tab	in	table to be accessed Allowed:table tool
columns	in	Array of column names Allowed:Vector of Strings Default: All
exclude	in	Exclude instead of include the named columns? Allowed:Bool Default: F

---

tablerow.set - Function

Package	**utility**
Module	**table**
Tool	**tablerow**

Set table and columns to be accessed

Synopsis
set(tab, columns, exclude)

Description
Reset the `tablerow` tool to use the given table and columns.

Arguments

tab	in	table to be accessed Allowed:table tool
columns	in	Array of column names Allowed:Vector of Strings Default: All
exclude	in	Exclude instead of include the named columns? Allowed:Bool Default: F

Returns
Bool

---

tablerow.get - Function

Package	**utility**
Module	**table**
Tool	**tablerow**

Get a table row

Synopsis
get(rownr)

Description
Get the values in the given row. The values are returned as a record where the field names are the column names. Note that rows are numbered starting at 1.

Arguments

rownr	in	Row number Allowed:Int

Returns
Record

---

tablerow.put - Function

Package	**utility**
Module	**table**
Tool	**tablerow**

Put a table row

Synopsis
put(rownr, value, matchingfields)

Description

Put the values in the given record into the given table row. By default only the fields will be put with a name matching a column name in the `tablerow` tool. However, if `matchingfields=F` the given record must exactly match the tablerow layout (i.e. the column names in it). This is faster and can be used to put a record acquired using the /texttget function.
Note that rows are numbered starting at 1.

Arguments

rownr	in	Row number Allowed:Int
value	in	Values to put Allowed:Record
matchingfields	in	F=value record must match tablerow layout Allowed:Bool Default: T

Returns
Bool

---

tablerow.close - Function

Package	**utility**
Module	**table**
Tool	**tablerow**

Close the tablerow

Synopsis
close()

Description
Close the tablerow. It can thereafter be reused using the set function.

Returns
Bool

---

tablerow.done - Function

Package **utility**

Module **table**
Tool **tablerow**

End the tablerow tool

Synopsis
done()

Description
End the `tablerow` tool and free up all memory. It cannot be reused anymore.

1.7.4 tableiterator - Tool

Package **utility**
Module **table**

Allows iteration through a table tool

Description
A tableiterator allows iteration through a table based on the contents of one or more columns. Each step in the iteration process forms a subset of the table for which the specified columns have the same value.

Example

```
vis:=table("3C273XC1.MS");
vis.summary();
visbyants:=tableiterator(vis, "ANTENNA1 ANTENNA2");
visbyants.reset();
while(visbyants.next()) {
  nrows:=visbyants.table().nrows();
  print "Antenna1 ", visbyants.table().getcell("ANTENNA1", 1),
        " Antenna2 ", visbyants.table().getcell("ANTENNA2", 1),
        ": ", nrows, "samples";
}
```

This example iterates through the table on ANTENNA1 and ANTENNA2. Thus each call to `next` forms a (reference) table containing the rows of a single

baseline in the main table. The `table` function can be used to get access to the current subset.
Note that a `next` has to be done to get the first subset.

Constructors

tableiterator	Construct a tableiterator

Functions

close	close the tableiterator tool
done	end the tableiterator tool
next	Advance to the next table subset
reset	Reset iteration to the beginning
table	Return the current table subset
terminate	Terminate the iteration and clean up memory

tableiterator - Constructor

Package	**utility**
Module	**table**
Tool	**tableiterator**

Construct a tableiterator

Synopsis
tableiterator(tab, columns, order, sort)

Description
Construct an iterator for the given table. Each step in the iteration returns a subtable for which the values of the given columns are equal. By default the table will be sorted before the iteration is started to achieve that all equal values are grouped together.
The optional order argument can be given to define how the table has to be sorted. 'a' is ascending, 'd' is descending, other is don't care (which comes to ascending). Instead of a single letter any word can be used starting with a or d (e.g. asc).
The optional sort argument can be given to define if the table has to be sorted. F means no sorting. The default T means sorting. The default sort algorithm used is heapsort. By giving the sort argument a string value, it can also be used to specify another sort algorithm. 'n' is no sort, 'q' is quicksort, 'i' is insertion sort, other is heapsort. A full word can also be used.

Arguments

tab	in	table to be iterated Allowed:table tool
columns	in	Array of column names Allowed:Vector of Strings
order	in	Order when sorting the table Allowed:String Default: ”
sort	in	Sort the table first? Allowed:Bool,String Default: T

---

tableiterator.table - Function

Package	**utility**
Module	**table**
Tool	**tableiterator**

Return the current table subset

Synopsis
table()

Description
Return a `table` tool for the current subset.
Note that `next` has to be called to form the first subset.

Returns
table tool—Bool

---

tableiterator.reset - Function

Package	**utility**
Module	**table**
Tool	**tableiterator**

Reset iteration to the beginning

Synopsis
reset()

Description
It resets the iteration to the beginning of the table. Note that `next` has to be called to get the first subset.

Returns
Bool

---

tableiterator.next - Function

Package	**utility**
Module	**table**
Tool	**tableiterator**

Advance to the next table subset

Synopsis
next()

Description
Form the next subset of the table. It returns a false status if no more subsets are available The subset can be accessed using the `table` function.
Note that `next` has to be called to form the first subset.

Returns
Bool

---

tableiterator.terminate - Function

Package	**utility**
Module	**table**
Tool	**tableiterator**

Terminate the iteration and clean up memory

Synopsis
terminate()

Description
This function has to be called if the iteration is ended prematurely. Otherwise tables are left open. If the iteration is done until the end, `terminate` does not need to be called, but it does not harm to call it.
Hereafter it is still possible to restart the iteration using the `reset` function.

Returns
Bool

---

tableiterator.close - Function

Package	**utility**
Module	**table**
Tool	**tableiterator**

close the tableiterator tool

Synopsis
close()

Description
A synonym for terminate.

Returns
Bool

---

tableiterator.done - Function

Package	**utility**
Module	**table**
Tool	**tableiterator**

end the tableiterator tool

Synopsis
done()

Description
Terminate the iteration and free up all memory. It cannot be reused anymore.

1.7.5 tableindex - Tool

Package utility
Module table

Allows value based table access

Description
A tableindex makes it possible to find rows in a table based on their contents. When constructing the tableindex it has to be specified for which column or columns an index has to be built. Those columns will be loaded in memory and thereafter row numbers can be found in a fast way using a binary search.

Example

```
vis:=table("3C273XC1.MS");
vis.summary();
visinx:=tableindex(vis, "ANTENNA1");
rows:=visinx.rownrs(4);
rows:=visinx.rownrs(4,7);
```

This example builds an index for column ANTENNA1. The one but last line returns the row numbers (1-relative) of all rows with value 4 for ANTENNA1. There after these row numbers can, for example be used in getcell to read data. The last line is similar, but in this case the search interval 4-7 is given. So all rows with value 4, 5, 6, or 7 in ANTENNA1 are found. An optional 3rd and 4th argument can be given to tell if the interval is open or closed (i.e. if the boundaries are part of the interval). Note that for indices on floating point numbers it is best to use a search interval, because an exact match on such numbers usually fails.

Example

```
vis:=table("3C273XC1.MS");
vis.summary();
visinx:=tableindex(vis, "ANTENNA1 ANTENNA2");
rows:=visinx.rownrs([ANTENNA1=4, ANTENNA2=5);
rows:=visinx.rownrs([ANTENNA1=4, ANTENNA2=5), [ANTENNA1=4, ANTENNA2=8];
```

This example builds an index for column ANTENNA1 and ANTENNA2. Because multiple columns are used, the arguments for function rownrs have to be records now. So the one but last line returns the row numbers (1-relative) of all rows with value 4 for ANTENNA1 and 5 for ANTENNA2. The last line defines a search interval again. In this case the ordering of the columns in the index is important. It finds the rows of ANTENNA1=4 where ANTENNA2 is in the interval 5-8.
Normally a column used in a tableindex has to be a scalar column. However, if a single column is used that column can contain arrays. In such a case the `rownrs` function finds the rows where a value in its array matches the given search value.
If all key values in the index are unique, the function rownr can be used as well. It returns a single row number instead of a vector.

Constructors

tableindex	Construct a tableindex

Functions

close	Close the tableindex
done	End the tableindex tool
isunique	Are all key values in the index unique?
rownr	Find the row number matching the search value
rownrs	Find the row numbers matching search values
set	Set table and columns to be indexed
setchanged	Tell the index that columns have changed

tableindex - Constructor

Package	**utility**
Module	**table**
Tool	**tableindex**

Construct a tableindex

Synopsis
tableindex(tab, columns, sort)

Description
Construct an index for the given table. The names of one or more scalar columns forming the index have to be given. If a single name is given, the column can be an array column too. Normally a sort will be done to get the key values in the correct order. If the table is already in the order of the data in the given columns, the argument `sort` can be set to F to avoid needless sorting.

Arguments

tab	in	table to be indexed Allowed:table tool
columns	in	column names Allowed:Vector of Strings
sort	in	is a sort of the key values needed? Allowed:Bool Default: T

---

tableindex.set - Function

Package	**utility**
Module	**table**
Tool	**tableindex**

Set table and columns to be indexed

Synopsis
set(tab, columns, sort)

Description
Reset the `tableindex` tool to index the given table and columns. It works similar to the tableindex constructor.

Arguments

tab	in	table to be indexed Allowed:table tool
columns	in	column names Allowed:Vector of Strings
sort	in	is a sort of the key values needed? Allowed:Bool Default: T

Returns
Bool

---

tableindex.isunique - Function

Package	**utility**
Module	**table**
Tool	**tableindex**

Are all key values in the index unique?

Synopsis
isunique()

Description
T is returned if all key values in the index are unique. In that case the function rownr can also be used.

Returns
Bool

---

tableindex.setchanged - Function

Package	**utility**
Module	**table**
Tool	**tableindex**

Tell the index that columns have changed

Synopsis
setchanged(columns)

Description
When data are changed in an indexed column, the index should be updated. The tableindex tool recognizes addition and deletion of rows and updates the index accordingly. However, it does not recognize if data in a column are changed. This function can be used to tell the tableindex that data in one or more columns are changed and force it to update the index. If no column names are given it assumes all columns have changed.

Arguments

columns	in	names of changed columns Allowed:Vector of Strings Default: All

Returns
Bool

---

tableindex.rownr - Function

Package	**utility**
Module	**table**
Tool	**tableindex**

Find the row number matching the search value

Synopsis
rownr(key)

Description
Find the row number (1-relative) of the row containing the given search value. If no row is found, 0 is returned. This function can only be used if the index contains unique key values (thus if isunique returns T), otherwise `rownrs` should be used.
Normally the search value should be given as a record in which the field names match the column names in the index. However, if the index is formed from a single column the search value can be given directly as shown in the examples in the introduction. The data type should be correct though.

Arguments

key	in	search value Allowed:Record

Returns
Int

---

tableindex.rownrs - Function

Package	**utility**
Module	**table**
Tool	**tableindex**

Find the row numbers matching search values

Synopsis
rownrs(key, upperkey, lowerincl, upperincl)

Description
Find the row numbers of the rows matching the given search value or interval. An interval is given if the `upperkey` argument is also given. In that case the 3rd and 4th argument determine if the boundaries of the interval are inclusive (i.e. part of the interval).
As in function `rownr` the search values must be given as a record if the index consists of multiple columns.

Arguments

key	in	search value Allowed:Record
upperkey	in	end of search interval Allowed:Record Default: unset
lowerincl	in	include lower boundary Allowed:Bool Default: T
upperincl	in	include upper boundary? Allowed:Bool Default: T

Returns
Vector of Ints

---

tableindex.close - Function

Package	**utility**
Module	**table**
Tool	**tableindex**

Close the tableindex

Synopsis
close()

Description
Close the tableindex. It can thereafter be reused using the set function.

Returns
Bool

---

tableindex.done - Function

Package	**utility**
Module	**table**
Tool	**tableindex**

End the tableindex tool

Synopsis
done()

Description
End the tableindex and free up all memory. It cannot be reused anymore.

1.7.6 is_table - Function

Package	**utility**
Module	**table**

Is this a valid table tool?

Synopsis
is_table(tab)

Description
Determine if this is a table by the following criteria:

- tab must be a record AND
- tab must have a field ok AND
- tab.ok must be a function AND
- tab.ok() must be TRUE

Arguments

tab	in	Putative table tool Allowed:Any value

Returns
Bool

1.7.7 tableopentables - Function

Package utility
Module table

return names of open tables

Synopsis
tableopentables(tableserver)

Description
Return the names of all tables open by this tableserver.

Arguments

tableserver	in	Table server to be used Allowed:Any tableserver Default: defaulttableserver

Returns
Vector of Strings

Example

```
print tableopentables();
```

1.7.8 tableexists - Function

Package utility
Module table

does this table exist on disk?

Synopsis
tableexists(tablename, tableserver)

Arguments

tablename	in	Name of table on disk Allowed:String
tableserver	in	Table server to be used Allowed:Any tableserver Default: defaulttableserver

Returns
Bool

Example

```
  print tableexists("3C273XC1.MS");
# T
```

1.7.9 tableiswritable - Function

Package utility
Module table

is this table writable?

Synopsis
tableiswritable(tablename, tableserver)

Arguments

tablename	in	Name of table on disk Allowed:String
tableserver	in	Table server to be used Allowed:Any tableserver Default: defaulttableserver

Returns
Bool

Example

```
  print tableiswritable("3C273XC1.MS");
# T
```

1.7.10 tableinfo - Function

Package utility
Module table

return the info record

Synopsis
tableinfo(tablename, tableserver)

Arguments

tablename	in	Name of table on disk Allowed:String
tableserver	in	Table server to be used Allowed:Any tableserver Default: defaulttableserver

Returns

Record

Example

```
  print tableinfo("3C273XC1.MS");
# [type=Measurement Set, subType=observed, readme=This is a MeasurementSet Table holding measurements from a Telescope]
```

In case the table does not exist, it returns a similar record where the values are empty strings.

1.7.11 tabledemo - Function

Package utility
Module table

Perform a demonstration: Not yet implemented

Synopsis
tabledemo()

1.7.12 tabletest - Function

Package utility
Module table

Perform a test

Synopsis
tabletest()

Description
Most of the functions allowed are exercised on small example tables that are created. If any operation fails, then the fail is returned. If the entire test succeeds, a T is returned.
It calls the test function in the script `ttable.g`, which might also be useful as an example how to use the `table` tool.

Returns
Bool—Fail

1.7.13 tablecommand - Function

Package utility
Module table

Execute a table command

Synopsis
tablecommand(comm, handle, tableserver)

Description
This function makes it possible to do selection and sorting on tables using an SQL-like grammar. It returns a table tool which can be used as any other table tool.
It is also possible to do a calculation on a table which returns the result as a record, vector, or scalar (see function calc).
The grammar for the command string is SQL-like and is described in Note 199.
Between SELECT and FROM, you can give some column names (separated by commas). Then the output table will contain those columns only. If no column names are given, all columns will be selected. E.g.:
`SELECT column1, column2,column3 FROM table.in GIVING table.out`
The WHERE part (like `column1*column2>10`) contains an arbitrary expression. Functions (like sin, max, ceil) are supported. Only scalar columns (or keywords) can be used in the expression. Complex numbers must be given as, for example, `3.4+4i` (similar to Glish). With some extra syntax (not explained here) it is even possible to use keywords from other tables in the expression. Rows obeying the WHERE expression will be selected.
Sorting can be done with the ORDERBY clause. You can give there any number of (scalar) columns separated by commas. You can use a mix of ascending (is default) and descending. E.g.:
`ORDERBY column1 DESC, column2, column3 ASC, column4 DESC`
The GIVING clause defines the output table containing the requested rows and columns. It can be used as any other table.
Each part (column list, WHERE, ORDERBY) is optional, but at least one of them should be given. Otherwise no operation would be done on the input table. The GIVING part is also optional.

Arguments

comm	in	Command string Allowed:String containing any valid table command
handle	in	Table handle Allowed:Is meant for internal use only Default: F
tableserver	in	Table server to be used Allowed:Any tableserver Default: defaulttableserver

Returns
table tool, record, vector, or scalar

Example

```
t := tablecommand('SELECT FROM table.in
                   WHERE column1*column2 > 10
                   ORDERBY column1 GIVING table.out');
t.summary();
tablecommand('CALC FROM table.in CALC column1+2')
tablecommand('CALC 1+2')
```

The last example shows that it can be used as a desk calculator.

1.7.14 tablecopy - Function

Package utility
Module table

copy a table

Synopsis
tablecopy(tablename, newtablename, deep, tableserver)

Description
Copy a table on disk. All subtables are also copied. References to another table are preserved.
The argument `deep` determines how a reference table (i.e. the result of a query) is copied. By default a file copy is made, thus the resulting table still contains references and no actual data. If, however, texttttdeep=T is given, a

deep copy is made which means that the actual data are copied. Also all subtables are copied.

Arguments

tablename	in	Name of existing table on disk Allowed:String
newtablename	in	Name of newtable on disk Allowed:String
deep	in	Make a deep copy of a reference table? Allowed:Bool Default: F
tableserver	in	Table server to be used Allowed:Any tableserver Default: defaulttableserver

Returns
Bool

1.7.15 tablerename - Function

Package utility
Module table

rename a table

Synopsis
tablerename(tablename, newtablename, tableserver)

Description
Rename a table on disk. All subtables are also renamed. References to another table are preserved.

Arguments

tablename	in	Name of existing table on disk Allowed:String
newtablename	in	Name of newtable on disk Allowed:String
tableserver	in	Table server to be used Allowed:Any tableserver Default: defaulttableserver

Returns
Bool

1.7.16 tabledelete - Function

Package utility
Module table

delete a table

Synopsis
tabledelete(tablename, checksubtables, ack, tableserver)

Description
Delete a table on disk. All subtables are also deleted. The delete fails when the table is still in use in another process. When it is in use in this process only, it is closed and deleted. Optionally it also checks if subtables are in use in another process.

Arguments

tablename	in	Name of table on disk Allowed:String
checksubtables	in	Check if subtables are multiused?) Allowed:Bool Default: T
ack	in	Acknowledge deletion of table) Allowed:Bool Default: T
tableserver	in	Table server to be used Allowed:Any tableserver Default: defaulttableserver

Returns
Bool

1.7.17 tablecloseall - Function

Package utility
Module table

close all open table tool objects

Synopsis
tablecloseall()

Description
It can easily happen that a table tool object is overwritten which may leave open the underlying table object. For instance:

```
t := table('a.b')
t := table('c.d')
```

overwrites `t` and leaves open table `a.b`.
This function closes all open table objects, thus also all open tables for which a valid tool object still exists.

Returns
Bool

1.7.18 tablecreatedesc - Function

Package utility
Module table

create a table descriptor

Synopsis
tablecreatedesc(columndesc)

Description

Creates a table descriptor from a set of column descriptions. The resulting table descriptor can be used in the constructor.
Any number of column descriptors may be given as arguments. Just add more arguments as needed.

Arguments

columndesc	in	Column descriptor Allowed:Any valid column descriptor

Returns

Table descriptor record

Example

```
scd1:=tablecreatescalarcoldesc("col2","aa")
scd2:=tablecreatescalarcoldesc("col1",1,"IncrementalStMan")
scd3:=tablecreatescalarcoldesc("colrec1",[=])
acd1:=tablecreatearraycoldesc("arr1",1,0,[2,3,4])
acd2:=tablecreatearraycoldesc("arr2",as_complex(0))
td:=tablecreatedesc(scd1, scd2, scd3, acd1, acd2)
tab:=table("mytable", tabledesc=td, nrow=100)
```

This creates a table descriptor td from five column descriptors and then creates a 100-row table called mytable from the table descriptor.
The columns contain respectivily strings, integer scalars, records, 3D integer arrays with fixed shape [2,3,4], and complex arrays with variable shape.

See Also

table tablecreatescalardesc tablecreatearraydesc

1.7.19 tablecreatescalarcoldesc - Function

Package utility
Module table

create descriptor of a scalar column

Synopsis

tablecreatescalarcoldesc(columnname, value, datamanagertype, datamanagergroup, options, maxlen, comment, tableserver)

Description

A descriptor for a scalar column can be created from a name for the column, and a data value, which is used only to determine the type of the column. Note that a record value is also possible.
It is possible to create the column descriptor in more detail by giving the data manager name, group, option and comment as well.
The data manager type tells which data manager (storage manager) is used to store the columns. The data manager type and group are explained in more detail in the Tables module description. The option allowed is:

Undefined If not given, all possible values of the scalar have a meaning. If given, a value equal to the undefined values in ValType.h is a truely undefined value.

The column descriptor is used in creating a table descriptor.

Arguments

columnname	in	Name of column Allowed:String
value	in	Example data value Allowed:Any type
datamanagertype	in	Type of data manager Allowed:String: StandardStMan—IncrementalStMan—TiledColumnStMan—TiledCellStMan—StManAipsIO Default: ”
datamanagergroup	in	Group of data manager Allowed:String Default: ”
options	in	Options Allowed:Int Default: 0
maxlen	in	Maximum string length Allowed:Int Default: 0
comment	in	Comment: informational for user Allowed:String Default: ”
tableserver	in	Table server to be used Allowed:Any tableserver Default: defaulttableserver

Returns
Column descriptor record

Example

```
scd1:=tablecreatescalarcoldesc("col2","aa"));
scd2:=tablecreatescalarcoldesc("col1",1,"IncrementalStMan");
td:=tablecreatedesc(scd1, scd1);
```

This creates a table description consisting of an integer column col1, and a string column col2. col1 uses the IncrementalStMan storage manager, while col2 uses the default storage manager StandardStMan.

See Also
tablecreatedesc tablecreatearraycoldesc

1.7.20 tablecreatearraycoldesc - Function

Package utility
Module table

create descriptor of an array column

Synopsis
tablecreatearraycoldesc(columname, value, ndim, shape, datamanagertype, datamanagergroup, options, maxlen, comment, tableserver)

Description
A descriptor for an array column is created from the following arguments:

The name of the column.

A data value, which is only used to determine the data type of the column.

Optionally the number of dimensions. A value > 0 means that all arrays in the column must have that dimensionality. Note that the arrays can still differ in shape unless the shape vector is also given.

Optionally a vector of integers giving the shape of the array in each cell. If given, it enables option FixedShape (see below) and sets the number of

dimensions (if not given). All arrays in the column get the given shape and the array is created as soon as a row is added.
Note that the shape vector gives the shape in each table cell; the number of rows in the table should NOT be part of it (otherwise you may run out of disk space).

Optionally the data manager type and group telling which data manager (storage manager) is used to store the columns. By default StandardStMan is used. The data manager type and group are explained in more detail in the Tables module description.

Optionally array options which can be added to combine them.

1 means Direct. It tells that the data are directly stored in the table. Direct forces option FixedShape. If not given, the array is indirect, which means that the data will be stored in a separate file.

4 means FixedShape. This option does not need to be given, because it is enforced when the shape is given. FixedShape means that the shape of the array must be the same in each cell of the column. Otherwise the array shapes may vary along the column and is it possible that a cell does not contain an array at all. Note that when given (or implicitly by option Direct), the shape argument must be given as well.

Optionally maxlen telling the maximum length of strings. The default value 0 means no maximum length.

Optionally a comment string.

The column descriptor is used in creating a table descriptor.

Arguments

columnname	in	Name of column Allowed:String
value	in	Example data value Allowed:Any type
ndim	in	Number of dimensions for a cell Allowed:Int Default: Length of shape
shape	in	Length of each axis for a cell Allowed:Vector of Int Default: []
datamanagertype	in	Type of data manager Allowed:String: StandardStMan—IncrementalStMan—TiledColumnStMan—TiledCellStMan—TiledShapeStMan—StManAipsIO Default: ”
datamanagergroup	in	Group of data manager Allowed:String Default: ”
options	in	Options Allowed:Int Default: 0
maxlen	in	Maximum string length Allowed:Int Default: 0
comment	in	Comment: informational for user Allowed:String Default: ”
tableserver	in	Table server to be used Allowed:Any tableserver Default: defaulttableserver

Returns
Column descriptor record

Example

```
acd1:=tablecreatearraycoldesc("arr1",as_float(1),0,[2,3,4]);
td:=tablecreatedesc(acd1);
```

This creates a table description consisting of an array column arr1 containing 3D arrays of floats with shape [2,3,4]. Note how the `value` parameter is used to set the type of the values that can be placed in the column. Other useful glish functions for this purpose are `as_double(), as_complex(), as_boolean()`, etc. See the “Type Conversions” subsection of the Glish User's Manual for a list.

See Also
tablecreatedesc tablecreatescalarcoldesc

1.7.21 tabledefinehypercolumn - Function

Package utility
Module table

define a hypercolumn in a table descriptor

Synopsis
tabledefinehypercolumn(tabledesc, name, ndim, datacolumns, coordcolumns, idcolumns)

Description
Defines a hypercolumn in the given table descriptor. A hypercolumn is an entity used by the Tiled Storage Managers (TSM). It defines which columns have to be stored together with a TSM.

Arguments

tabledesc	inout	Table descriptor Allowed:Any valid table descriptor
name	in	Name of hypercolumn Allowed:String
ndim	in	Dimensionality of hypercolumn Allowed:Int
datacolumns	in	Data columns to be stored with TSM Allowed:Vector of strings
coordcolumns	in	Coordinate columns to be stored with TSM Allowed:Vector of strings Default: none
idcolumns	in	Id columns to be stored with TSM Allowed:Vector of strings Default: none

Returns
Bool

Example

```
scd1:=tablecreatescalarcoldesc("col2","aa")
scd2:=tablecreatescalarcoldesc("col1",1,"IncrementalStMan")
scd3:=tablecreatescalarcoldesc("colrec1",[=])
acd1:=tablecreatearraycoldesc("arr1",1,0,[2,3,4])
acd2:=tablecreatearraycoldesc("arr2",as_complex(0))
td:=tablecreatedesc(scd1, scd2, scd3, acd1, acd2)
tabledefinehypercolumn(td, "TiledArray", 4, "arr1")
tab:=table("mytable", tabledesc=td, nrow=100)
```

This creates a table descriptor td from five column descriptors and then creates a 100-row table called mytable from the table descriptor.
The columns contain respectivily strings, integer scalars, records, 3D integer arrays with fixed shape [2,3,4], and complex arrays with variable shape. The first array is stored with the Tiled Storage Manager (in this case the TiledColumnStMan).

See Also
tablecreatedesc tablecreatescalardesc tablecreatearraydesc

1.7.22 tablebrowser - Function
Package utility
Module table

"Table browser"

Synopsis
tablebrowser(table, readonly, rows2read, show, hide)

Arguments

table	Table name or handle Allowed:Any valid table name or handle
readonly	Flag to open the table readonly or not Allowed:Boolean T or F Default: T
rows2read	Then number of rows to cache in memory (set to -1 to read all rows) Allowed:integer Default: 1000
show	Array of columns to display (case insensitive) Allowed:Array of Strings Default: T
hide	Array of columns to hide from display (case insensitive) Allowed:Array of Strings Default: T

Returns
Glish agent

Example
include "tablebrowser.g"

```
nt := tablebrowser('MS')
```

This will load and display the MS table readonly in the table browser. To enable editing of the table use the following:

```
nt := tablebrowser('MS', readonly=F)
```

To read all rows of a table into memory use:

```
nt := tablebrowser('MS', rows2read=-1)
```

Warning.. large tables can take a long time to load (several minutes). Another application may close the table browser using the following:

```
nt->close(T)
```

Note: If the show and hide arguments are used together, show takes precedence. A hide keyword for a column is overruled when using show.

Description
Browse a table using a graphical interface. The current implementation supports subset of the table tool's functions. The graphical interfaces has three parts:

- **Menu Bar** has five options:

File -

Open starts a Table chooser,

Save will save changes to the table (not visibile with readonly set to T)

Save As starts a Table chooser which lets you make a copy of the table with a new name,

Report Bug , Displays the AIPS++bug reporting form,

Close , Dismisses the GUI.

Edit -

Cut Copy Paste New Row are currently not available,

Properties displays a properites dialog which allow you to edit some display parameters, size of the vector to show, how many rows to cache in memory, and how big a canvas to cache,

Help Browser choose the html browser to use for help.

View -

Table Summary displays a dialog containing table summary information,

Table Keywords displays a dialog containing the table keywords,

Hide/Show Columns displays a dialog which allows one to specify which columns to show or hide.

Table -

Query display a table query dialog, which will display the resulting table in a new window (uses TaQL),

Plot display a dialog for selecting what to plot, columns maybe specified and used in Glish expressions.

Select display a table query dialog, which will highlight the rows matching a query (uses TaQL).

Help - drives the browser to the new table browser html.

- **Goto** Keying a number into the "Goto" entry and pressing return will put the row number at the top of the canvas.

- **Display Canvas** has the following features:

Vertical Scrollbar if a table does not fit into memory, the vertical scrolling bar can not be grabbed and dragged. Clicking in the trough and on the arrows will do reposition the scrollbar. If the table fits into memory the scroll bar works as expected.

Column Labels Each column label has a menu attached to it,

Select selects an entire column,

Keywords displays a dialog containing the keywords for that column The following column keywords will be used by the table browser

- **BROWSER_LABEL** , what label to use for the column,
- **BROWSER_FORMAT** , how to format the members of that column (uses C syntax, currently not implemented), and
- **BROWSER_HIDE** , whether to display that column or not

Hide removes the column from the display canvas,

Format selects how to format the column, uses C syntax.

Rows and Individual Cells may be selected by pressing a mouse button on the text. Vectors may be displayed as "[1 2 3]" or "[3]integer" depending on how many elements of the vector are chosen to display. Selecting on the "[3]integer" form will cause an arraybrowser to be displayed with the values of the vector or array displayed in it.

See Also
table tool

1.7.23 tablecomparer - Tool

Package	**utility**
Module	**table**

Compare various aspects of two tables to see if they are equivalent

Description
A `tablecomparer` tool compares various aspects of two tables to see if they are equivalent. Currently, attributes that can be compared include column names, column keyword names, column values, and table keyword names. When checking the values of the column data, a tolerance can be specified on a per column basis and/or a per column type (i.e., float, double, complex, or dcomplex) basis. Currently only absolute tolerances can be specified. The user can also specify whether tests should be run on subtables as well as the main tables.

Constructors

tablecomparer	Construct a tablecomparer tool

Functions

checkcoldata	Compare column data
checkcolkeywordnames	Compare column keyword names
checkcolnames	Check column names

checkkeywordnames	Compare table keyword names
compare	Compare tables by running one or more tests
details	Get details regarding failed tests
select	Specify which columns to check (or not to check)
settolerance	Set tolerances when comparing column data
summary	Summarize the comparisons
tolerance	Get the tolerance of the specified column

tablecomparer - Constructor

Package	**utility**
Module	**table**
Tool	**tablecomparer**

Construct a tablecomparer tool

Synopsis
tablecomparer(test, model, label, verbose)

Description
Create a `tablecomparer` tool.

Arguments

test	Name of the test table Allowed:Any valid table name Default: None
model	Name of the model (fiducial) table to compare the test table against Allowed:Any valid table name Default: None
label	Label for the (top level) table Allowed:String Default: 'TABLE'
verbose	Controls the volume of output, 0 no log messages, higher numbers mean more Allowed:Int Default: 1

Returns
tablecomparer tool

Example

```
# create a tablecomparer tool, and specify you want a lot of output
- tc := tablecomparer('test.table','fiducial.table',verbose=2);
```

---

tablecomparer.checkcolnames - Function

Package	**utility**
Module	**table**
Tool	**tablecomparer**

Check column names

Synopsis
checkcolnames(checksubtables)

Description
Check column names to see if they are identical between the two tables. Note that all column names will be checked regardless of any column selection made previously with `select()`.

Arguments

checksubtables	Check column names of subtables as well? Allowed:Boolean Default: T

Returns
T if the column names between the two tables are the same, F otherwise

Example

```
# create a tablecomparer tool, and specify you want a lot of output
- tc := tablecomparer('test.table','fiducial.table',verbose=2);
- tc.checkcolnames()
NORMAL: Column name test passed on table TABLE
Opening  CAL_DESC
NORMAL: successful readonly open of default-locked table
```

```
test.table/CAL_DESC: 12 columns, 1 rows
NORMAL: successful readonly open of default-locked table
fiducial.table/CAL_DESC: 12 columns, 1 rows
NORMAL: Column name test passed on table CAL_DESC
Opening  CAL_HISTORY
NORMAL: successful readonly open of default-locked table
test.table/CAL_HISTORY: 4 columns, 0 rows
NORMAL: successful readonly open of default-locked table
fiducial.table/CAL_HISTORY: 4 columns, 0 rows
NORMAL: Column name test passed on table CAL_HISTORY
T
```

Or, in the case where the test fails:

```
- t := tablecomparer('badtest.table','fiducial.table',verbose=2)
NORMAL: successful readonly open of default-locked table
badtest.table: 36 columns, 210 rows
NORMAL: successful readonly open of default-locked table
fiducial.table: 37 columns, 210 rows
- t.checkcolnames()
SEVERE: Column name test failed on table TABLE
SEVERE: TEST TABLE COLS: TIME TIME_EXTRA_PREC INTERVAL ANTENNA1 FEED1
FIELD_ID ARRAY_ID OBSERVATION_ID SCAN_NUMBER PROCESSOR_ID STATE_ID
PHASE_ID PULSAR_BIN PULSAR_GATE_ID FREQ_GROUP FREQ_GROUP_NAME
FIELD_NAME FIELD_CODE SOURCE_NAME SOURCE_CODE CALIBRATION_GROUP
REF_ANT REF_FEED REF_RECEPTOR REF_FREQUENCY MEAS_FREQ_REF
REF_DIRECTION MEAS_DIR_REF CAL_DESC_ID CAL_HISTORY_ID
TOTAL_SOLUTION_OK TOTAL_FIT TOTAL_FIT_WEIGHT SOLUTION_OK FIT
FIT_WEIGHT
SEVERE: MODEL TABLE COLS: TIME TIME_EXTRA_PREC INTERVAL ANTENNA1 FEED1
FIELD_ID ARRAY_ID OBSERVATION_ID SCAN_NUMBER PROCESSOR_ID STATE_ID
PHASE_ID PULSAR_BIN PULSAR_GATE_ID FREQ_GROUP FREQ_GROUP_NAME
FIELD_NAME FIELD_CODE SOURCE_NAME SOURCE_CODE CALIBRATION_GROUP GAIN
REF_ANT REF_FEED REF_RECEPTOR REF_FREQUENCY MEAS_FREQ_REF
REF_DIRECTION MEAS_DIR_REF CAL_DESC_ID CAL_HISTORY_ID
TOTAL_SOLUTION_OK TOTAL_FIT TOTAL_FIT_WEIGHT SOLUTION_OK FIT
FIT_WEIGHT
Opening  CAL_DESC
NORMAL: successful readonly open of default-locked table
badtest.table/CAL_DESC: 12 columns, 1 rows
NORMAL: successful readonly open of default-locked table
fiducial.table/CAL_DESC: 12 columns, 1 rows
NORMAL: Column name test passed on table CAL_DESC
Opening  CAL_HISTORY
NORMAL: successful readonly open of default-locked table
badtest.table/CAL_HISTORY: 4 columns, 0 rows
NORMAL: successful readonly open of default-locked table
```

```
fiducial.table/CAL_HISTORY: 4 columns, 0 rows
NORMAL: Column name test passed on table CAL_HISTORY
F

\vspace{.18in}
\hrule


   \subsubsection{tablecomparer.checkcolkeywordnames - Function\label{table:tablecomparer.checkcolkeywordnames.function}\index{table.tablecomparer.checkcolkeywordnames}\index{checkcolkeywordnames}}
   \subsectionM{tablecomparer.checkcolkeywordnames}{utility}{table}{tablecomparer}
   Compare column keyword names\\
   \flushleft\vspace{.20in}\textbf {Synopsis}\newline
   checkcolkeywordnames(checksubtables)

\flushleft\vspace{.20in}\textbf {Description}\newline
This function compares the keyword names of all columns not excluded
by the {\tt select()} function.  If all column keyword names match
between the two tables, it returns {\tt T}, if not, it returns {\tt F}.

\flushleft\vspace{.20in}\textbf {Arguments}\\
   \hfill \\
   \starthelptab
   \hline

   checksubtables & & \multicolumn{2}{p{3.36in}|}{Check column names of subtables as well?}\\
   & & Allowed: & Boolean\\
   & & Default: & T  \\ \hline
\end{tabular}
\flushleft\vspace{.20in}\textbf {Returns}\newline
   T if the tests pass, F otherwise\\
\flushleft\vspace{.20in}\textbf {Example}\newline
\begin{verbatim}
- tc := tablecomparer('test.table','fiducial.table',verbose=2)
NORMAL: successful readonly open of default-locked table test.table: 36 columns, 210 rows
NORMAL: successful readonly open of default-locked table fiducial.table: 37 columns, 210 rows
- tc.checkcolkeywordnames()
NORMAL: Column keyword name test passed on table TABLE column TIME
NORMAL: Column keyword name test passed on table TABLE column TIME_EXTRA_PREC
NORMAL: Column keyword name test passed on table TABLE column INTERVAL
NORMAL: Column keyword name test passed on table TABLE column ANTENNA1
NORMAL: Column keyword name test passed on table TABLE column FEED1
NORMAL: Column keyword name test passed on table TABLE column FIELD_ID
NORMAL: Column keyword name test passed on table TABLE column ARRAY_ID
NORMAL: Column keyword name test passed on table TABLE column OBSERVATION_ID
NORMAL: Column keyword name test passed on table TABLE column SCAN_NUMBER
NORMAL: Column keyword name test passed on table TABLE column PROCESSOR_ID
```

```
NORMAL: Column keyword name test passed on table TABLE column STATE_ID
NORMAL: Column keyword name test passed on table TABLE column PHASE_ID
NORMAL: Column keyword name test passed on table TABLE column PULSAR_BIN
NORMAL: Column keyword name test passed on table TABLE column PULSAR_GATE_ID
NORMAL: Column keyword name test passed on table TABLE column FREQ_GROUP
NORMAL: Column keyword name test passed on table TABLE column FREQ_GROUP_NAME
NORMAL: Column keyword name test passed on table TABLE column FIELD_NAME
NORMAL: Column keyword name test passed on table TABLE column FIELD_CODE
NORMAL: Column keyword name test passed on table TABLE column SOURCE_NAME
NORMAL: Column keyword name test passed on table TABLE column SOURCE_CODE
NORMAL: Column keyword name test passed on table TABLE column CALIBRATION_GROUP
NORMAL: Column keyword name test passed on table TABLE column REF_ANT
NORMAL: Column keyword name test passed on table TABLE column REF_FEED
NORMAL: Column keyword name test passed on table TABLE column REF_RECEPTOR
NORMAL: Column keyword name test passed on table TABLE column REF_FREQUENCY
NORMAL: Column keyword name test passed on table TABLE column MEAS_FREQ_REF
NORMAL: Column keyword name test passed on table TABLE column REF_DIRECTION
NORMAL: Column keyword name test passed on table TABLE column MEAS_DIR_REF
NORMAL: Column keyword name test passed on table TABLE column CAL_DESC_ID
NORMAL: Column keyword name test passed on table TABLE column CAL_HISTORY_ID
NORMAL: Column keyword name test passed on table TABLE column TOTAL_SOLUTION_OK
NORMAL: Column keyword name test passed on table TABLE column TOTAL_FIT
NORMAL: Column keyword name test passed on table TABLE column TOTAL_FIT_WEIGHT
NORMAL: Column keyword name test passed on table TABLE column SOLUTION_OK
NORMAL: Column keyword name test passed on table TABLE column FIT
NORMAL: Column keyword name test passed on table TABLE column FIT_WEIGHT
NORMAL: successful readonly open of default-locked table test.table/CAL_DESC: 12 columns, 1 rows
NORMAL: successful readonly open of default-locked table fiducial.table/CAL_DESC: 12 columns, 1 rows
NORMAL: Column keyword name test passed on table CAL_DESC column NUM_SPW
NORMAL: Column keyword name test passed on table CAL_DESC column NUM_CHAN
NORMAL: Column keyword name test passed on table CAL_DESC column NUM_RECEPTORS
NORMAL: Column keyword name test passed on table CAL_DESC column N_JONES
NORMAL: Column keyword name test passed on table CAL_DESC column SPECTRAL_WINDOW_ID
NORMAL: Column keyword name test passed on table CAL_DESC column CHAN_FREQ
NORMAL: Column keyword name test passed on table CAL_DESC column MEAS_FREQ_REF
NORMAL: Column keyword name test passed on table CAL_DESC column CHAN_WIDTH
NORMAL: Column keyword name test passed on table CAL_DESC column CHAN_RANGE
NORMAL: Column keyword name test passed on table CAL_DESC column POLARIZATION_TYPE
NORMAL: Column keyword name test passed on table CAL_DESC column JONES_TYPE
NORMAL: Column keyword name test passed on table CAL_DESC column MS_NAME
NORMAL: successful readonly open of default-locked table test.table/CAL_HISTORY: 4 columns, 0 rows
NORMAL: successful readonly open of default-locked table fiducial.table/CAL_HISTORY: 4 columns, 0 rows
NORMAL: Column keyword name test passed on table CAL_HISTORY column CAL_PARMS
NORMAL: Column keyword name test passed on table CAL_HISTORY column CAL_TABLES
NORMAL: Column keyword name test passed on table CAL_HISTORY column CAL_SELECT
NORMAL: Column keyword name test passed on table CAL_HISTORY column CAL_NOTES
```

T

---

tablecomparer.checkcoldata - Function

Package	**utility**
Module	**table**
Tool	**tablecomparer**

Compare column data

Synopsis
checkcoldata(checksubtables)

Description
This function compares the data of all columns not excluded by the `select()` function. Tolerances on a per column and/or per column type basis basis can be specified by running `setprecision()` before running this function. This function will check both regular and variable array column data, in the latter case doing a record by record test for equality. It assumes that unset values are equal. If all column data match (to within the specified precisions, if any) between the two tables, the function returns `T`, if not, it returns `F`.

Arguments

checksubtables	Check column names of subtables as well? Allowed:Boolean Default: T

Returns
T if the tests pass, F otherwise

Example

```
- tc := tablecomparer('test.table','fiducial.table',verbose=2)
NORMAL: successful readonly open of default-locked table test.table: 37 columns, 210 rows
NORMAL: successful readonly open of default-locked table fiducial.table: 37 columns, 210 rows
- tc.checkcoldata()
NORMAL: Column data test passed on table TABLE column TIME
```

```
NORMAL: Column data test passed on table TABLE column TIME_EXTRA_PREC
NORMAL: Column data test passed on table TABLE column INTERVAL
NORMAL: Column data test passed on table TABLE column ANTENNA1
NORMAL: Column data test passed on table TABLE column FEED1
NORMAL: Column data test passed on table TABLE column FIELD_ID
NORMAL: Column data test passed on table TABLE column ARRAY_ID
NORMAL: Column data test passed on table TABLE column OBSERVATION_ID
NORMAL: Column data test passed on table TABLE column SCAN_NUMBER
NORMAL: Column data test passed on table TABLE column PROCESSOR_ID
NORMAL: Column data test passed on table TABLE column STATE_ID
NORMAL: Column data test passed on table TABLE column PHASE_ID
NORMAL: Column data test passed on table TABLE column PULSAR_BIN
NORMAL: Column data test passed on table TABLE column PULSAR_GATE_ID
NORMAL: Column data test passed on table TABLE column FREQ_GROUP
NORMAL: Column data test passed on table TABLE column FREQ_GROUP_NAME
NORMAL: Column data test passed on table TABLE column FIELD_NAME
NORMAL: Column data test passed on table TABLE column FIELD_CODE
NORMAL: Column data test passed on table TABLE column SOURCE_NAME
NORMAL: Column data test passed on table TABLE column SOURCE_CODE
NORMAL: Column data test passed on table TABLE column CALIBRATION_GROUP
NORMAL: Column data test passed on table TABLE column GAIN
NORMAL: Column data test passed on table TABLE column REF_ANT
NORMAL: Column data test passed on table TABLE column REF_FEED
NORMAL: Column data test passed on table TABLE column REF_RECEPTOR
NORMAL: Column data test passed on table TABLE column REF_FREQUENCY
NORMAL: Column data test passed on table TABLE column MEAS_FREQ_REF
NORMAL: Column data test passed on table TABLE column REF_DIRECTION
NORMAL: Column data test passed on table TABLE column MEAS_DIR_REF
NORMAL: Column data test passed on table TABLE column CAL_DESC_ID
NORMAL: Column data test passed on table TABLE column CAL_HISTORY_ID
NORMAL: Column data test passed on table TABLE column TOTAL_SOLUTION_OK
NORMAL: Column data test passed on table TABLE column TOTAL_FIT
NORMAL: Column data test passed on table TABLE column TOTAL_FIT_WEIGHT
NORMAL: Column data test passed on table TABLE column SOLUTION_OK
NORMAL: Column data test passed on table TABLE column FIT
NORMAL: Column data test passed on table TABLE column FIT_WEIGHT
NORMAL: successful readonly open of default-locked table test.table/CAL_DESC: 12 columns, 1 rows
NORMAL: successful readonly open of default-locked table fiducial.table/CAL_DESC: 12 columns, 1 rows
NORMAL: Column data test passed on table CAL_DESC column NUM_SPW
NORMAL: Column data test passed on table CAL_DESC column NUM_CHAN
NORMAL: Column data test passed on table CAL_DESC column NUM_RECEPTORS
NORMAL: Column data test passed on table CAL_DESC column N_JONES
NORMAL: Column data test passed on table CAL_DESC column SPECTRAL_WINDOW_ID
NORMAL: Column data test passed on table CAL_DESC column CHAN_FREQ
NORMAL: Column data test passed on table CAL_DESC column MEAS_FREQ_REF
NORMAL: Column data test passed on table CAL_DESC column CHAN_WIDTH
```

```
NORMAL: Column data test passed on table CAL_DESC column CHAN_RANGE
NORMAL: Column data test passed on table CAL_DESC column POLARIZATION_TYPE
NORMAL: Column data test passed on table CAL_DESC column JONES_TYPE
NORMAL: Column data test passed on table CAL_DESC column MS_NAME
NORMAL: successful readonly open of default-locked table test.table/CAL_HISTORY: 4 columns, 0 rows
NORMAL: successful readonly open of default-locked table fiducial.table/CAL_HISTORY: 4 columns, 0 rows
NORMAL: Column data test passed on table CAL_HISTORY column CAL_PARMS
NORMAL: Column data test passed on table CAL_HISTORY column CAL__TABLES
NORMAL: Column data test passed on table CAL_HISTORY column CAL_SELECT
NORMAL: Column data test passed on table CAL_HISTORY column CAL_NOTES
T
```

---

tablecomparer.checkkeywordnames - Function

Package	**utility**
Module	**table**
Tool	**tablecomparer**

Compare table keyword names

Synopsis

checkkeywordnames(checksubtables)

Description

This function compares table keyword names. If the keyword names in one table match those in the other, it returns `T`.

Arguments

checksubtables	Check column names of subtables as well? Allowed:Boolean Default: T

Returns

T if the tests pass, F otherwise

Example

```
- tc := tablecomparer('test.table','fiducial.table',verbose=2)
NORMAL: successful readonly open of default-locked table test.table:
37 columns, 210 rows
NORMAL: successful readonly open of default-locked table fiducial.table: 37 columns, 210 rows
- tc.checkkeywordnames()
NORMAL: Keyword field names test passed for table TABLE
NORMAL: successful readonly open of default-locked table test.table/CAL_DESC: 12 columns, 1 rows
NORMAL: successful readonly open of default-locked table fiducial.table/CAL_DESC: 12 columns, 1 rows
NORMAL: Keyword field names test passed for table CAL_DESC
NORMAL: successful readonly open of default-locked table test.table/CAL_HISTORY: 4 columns, 0 rows
NORMAL: successful readonly open of default-locked table fiducial.table/CAL_HISTORY: 4 columns, 0 rows
NORMAL: Keyword field names test passed for table CAL_HISTORY
T
```

---

tablecomparer.settolerance - Function

Package	**utility**
Module	**table**
Tool	**tablecomparer**

Set tolerances when comparing column data

Synopsis
settolerance(float, double, complex, dcomplex, cols)

Description
This function sets tolerances that are used by `checkcoldata` when comparing (numerical) column data. Currently, only absolute (not fractional) tolerances are supported. Tolerances can be set according to the type of a column (`float, double, complex, dcomplex`) or they can be specified on a per column basis using the `cols` parameter. In the case where both the column and its type have been given a tolerance, the column specific tolerance is used. When in doubt, you can check what the tolerance of a specific column is by running the `tolerance()` function. In the case of complex numbers, the tolerance is a real number representing the radius about the value in question in the complex plane.
Here are a couple of examples. Suppose you have a table with a column of type double called `DCOL`. If you run `settolerance(double=1e-8)` then the acceptable tolerance when checking the values in this column will be 1e-8.
The way to specify tolerances on a per column basis is to pass in a record with fields of the form `TABLELABEL.COLUMNNAME`, bearing in mind that the default value of `label` in the `tablecomparer()` constructor is `TABLE` and that

subtables of the main table are labeled by their names. So in the previous example, if you used the default value of `label` when constructing a tablecomparer tool, you could run

```
tc := tablecomparer('test.table','fiducial.table')
tol := [=];
tol.TABLE.DCOL := 1e-8;
tc.setprecision(cols := tol);
```

If you wanted to set the tolerance of `DCOL` in the main table to 1e-8 and the tolerance of `GCOL` of subtable `GTABLE` to be 1e-5, you could run

```
tc := tablecomparer('test.table','fiducial.table')
tol := [=];
tol.TABLE.DCOL := 1e-8;
tol.GTABLE.GCOL := 1e-5;
tc.setprecision(cols := tol);
```

Arguments

float	Default tolerance for all columns of type float Allowed:Float Default: 0
double	Default tolerance for all columns of type double Allowed:Double Default: 0
complex	Default tolerance for all columns of type complex Allowed:Float Default: 0
dcomplex	Default tolerance for all columns of type complex Allowed:Double Default: 0
cols	Column specific tolerances (see above for examples Allowed:Record containing real values Default: [=]

Returns
T

Example

```
- tc := tablecomparer('test.table','fiducial.table',verbose=3)
- # set the default tolerance of all columns of type double to 1e-8
- # all float columns to 1e-4, and set the tolerance of the GAIN
```

```
- # column in the main table to 1e-5 and the CHA_WIDTH column in the
- # CAL_DESC table to 1e-4
- tol := [=];
- tol.TABLE.GAIN := 1e-5;
- tol.CAL_DESC.CHAN_WIDTH := 1e-4;
- tc.settolerance(float=1e-5, double=1e-8, cols=tol);
T
# compare column data using these tolerances
- tc.checkcoldata();
NORMAL: Checking column TIME
NORMAL: Using tolerance 1e-08 on table TABLE column TIME
NORMAL: Column data test passed on table TABLE column TIME
NORMAL: Checking column TIME_EXTRA_PREC
NORMAL: Using tolerance 1e-08 on table TABLE column TIME_EXTRA_PREC
NORMAL: Column data test passed on table TABLE column TIME_EXTRA_PREC
NORMAL: Checking column INTERVAL
NORMAL: Using tolerance 1e-08 on table TABLE column INTERVAL
NORMAL: Column data test passed on table TABLE column INTERVAL
NORMAL: Checking column ANTENNA1
NORMAL: Column data test passed on table TABLE column ANTENNA1
NORMAL: Checking column FEED1
NORMAL: Column data test passed on table TABLE column FEED1
NORMAL: Checking column FIELD_ID
NORMAL: Column data test passed on table TABLE column FIELD_ID
NORMAL: Checking column ARRAY_ID
NORMAL: Column data test passed on table TABLE column ARRAY_ID
NORMAL: Checking column OBSERVATION_ID
NORMAL: Column data test passed on table TABLE column OBSERVATION_ID
NORMAL: Checking column SCAN_NUMBER
NORMAL: Column data test passed on table TABLE column SCAN_NUMBER
NORMAL: Checking column PROCESSOR_ID
NORMAL: Column data test passed on table TABLE column PROCESSOR_ID
NORMAL: Checking column STATE_ID
NORMAL: Column data test passed on table TABLE column STATE_ID
NORMAL: Checking column PHASE_ID
NORMAL: Column data test passed on table TABLE column PHASE_ID
NORMAL: Checking column PULSAR_BIN
NORMAL: Column data test passed on table TABLE column PULSAR_BIN
NORMAL: Checking column PULSAR_GATE_ID
NORMAL: Column data test passed on table TABLE column PULSAR_GATE_ID
NORMAL: Checking column FREQ_GROUP
NORMAL: Column data test passed on table TABLE column FREQ_GROUP
NORMAL: Checking column FREQ_GROUP_NAME
NORMAL: Column data test passed on table TABLE column FREQ_GROUP_NAME
NORMAL: Checking column FIELD_NAME
NORMAL: Column data test passed on table TABLE column FIELD_NAME
```

```
NORMAL: Checking column FIELD_CODE
NORMAL: Column data test passed on table TABLE column FIELD_CODE
NORMAL: Checking column SOURCE_NAME
NORMAL: Column data test passed on table TABLE column SOURCE_NAME
NORMAL: Checking column SOURCE_CODE
NORMAL: Column data test passed on table TABLE column SOURCE_CODE
NORMAL: Checking column CALIBRATION_GROUP
NORMAL: Column data test passed on table TABLE column CALIBRATION_GROUP
NORMAL: Checking column GAIN
NORMAL: Using tolerance 1e-05 on table TABLE column GAIN
NORMAL: Column data test passed on table TABLE column GAIN
NORMAL: Checking column REF_ANT
NORMAL: Column data test passed on table TABLE column REF_ANT
NORMAL: Checking column REF_FEED
NORMAL: Column data test passed on table TABLE column REF_FEED
NORMAL: Checking column REF_RECEPTOR
NORMAL: Column data test passed on table TABLE column REF_RECEPTOR
NORMAL: Checking column REF_FREQUENCY
NORMAL: Using tolerance 1e-08 on table TABLE column REF_FREQUENCY
NORMAL: Column data test passed on table TABLE column REF_FREQUENCY
NORMAL: Checking column MEAS_FREQ_REF
NORMAL: Column data test passed on table TABLE column MEAS_FREQ_REF
NORMAL: Checking column REF_DIRECTION
NORMAL: Using tolerance 1e-08 on table TABLE column REF_DIRECTION
NORMAL: Column data test passed on table TABLE column REF_DIRECTION
NORMAL: Checking column MEAS_DIR_REF
NORMAL: Column data test passed on table TABLE column MEAS_DIR_REF
NORMAL: Checking column CAL_DESC_ID
NORMAL: Column data test passed on table TABLE column CAL_DESC_ID
NORMAL: Checking column CAL_HISTORY_ID
NORMAL: Column data test passed on table TABLE column CAL_HISTORY_ID
NORMAL: Checking column TOTAL_SOLUTION_OK
NORMAL: Column data test passed on table TABLE column TOTAL_SOLUTION_OK
NORMAL: Checking column TOTAL_FIT
NORMAL: Using tolerance 1e-05 on table TABLE column TOTAL_FIT
NORMAL: Column data test passed on table TABLE column TOTAL_FIT
NORMAL: Checking column TOTAL_FIT_WEIGHT
NORMAL: Using tolerance 1e-05 on table TABLE column TOTAL_FIT_WEIGHT
NORMAL: Column data test passed on table TABLE column TOTAL_FIT_WEIGHT
NORMAL: Checking column SOLUTION_OK
NORMAL: Column data test passed on table TABLE column SOLUTION_OK
NORMAL: Checking column FIT
NORMAL: Using tolerance 1e-05 on table TABLE column FIT
NORMAL: Column data test passed on table TABLE column FIT
NORMAL: Checking column FIT_WEIGHT
NORMAL: Using tolerance 1e-05 on table TABLE column FIT_WEIGHT
```

```
NORMAL: Column data test passed on table TABLE column FIT_WEIGHT
NORMAL: successful readonly open of default-locked table test.table/CAL_DESC: 12 columns, 1 rows
NORMAL: successful readonly open of default-locked table fiducial.table/CAL_DESC: 12 columns, 1 rows
NORMAL: Checking column NUM_SPW
NORMAL: Column data test passed on table CAL_DESC column NUM_SPW
NORMAL: Checking column NUM_CHAN
NORMAL: Column data test passed on table CAL_DESC column NUM_CHAN
NORMAL: Checking column NUM_RECEPTORS
NORMAL: Column data test passed on table CAL_DESC column NUM_RECEPTORS
NORMAL: Checking column N_JONES
NORMAL: Column data test passed on table CAL_DESC column N_JONES
NORMAL: Checking column SPECTRAL_WINDOW_ID
NORMAL: Column data test passed on table CAL_DESC column SPECTRAL_WINDOW_ID
NORMAL: Checking column CHAN_FREQ
NORMAL: Using tolerance 1e-08 on table CAL_DESC column CHAN_FREQ
NORMAL: Column data test passed on table CAL_DESC column CHAN_FREQ
NORMAL: Checking column MEAS_FREQ_REF
NORMAL: Column data test passed on table CAL_DESC column MEAS_FREQ_REF
NORMAL: Checking column CHAN_WIDTH
NORMAL: Using tolerance 0.0001 on table CAL_DESC column CHAN_WIDTH
NORMAL: Column data test passed on table CAL_DESC column CHAN_WIDTH
NORMAL: Checking column CHAN_RANGE
NORMAL: Column data test passed on table CAL_DESC column CHAN_RANGE
NORMAL: Checking column POLARIZATION_TYPE
NORMAL: Column data test passed on table CAL_DESC column
POLARIZATION_TYPE
NORMAL: Checking column JONES_TYPE
NORMAL: Column data test passed on table CAL_DESC column JONES_TYPE
NORMAL: Checking column MS_NAME
NORMAL: Column data test passed on table CAL_DESC column MS_NAME
NORMAL: successful readonly open of default-locked table test.table/CAL_HISTORY: 4 columns, 0 rows
NORMAL: successful readonly open of default-locked table fiducial.table/CAL_HISTORY: 4 columns, 0 rows
NORMAL: Checking column CAL_PARMS
NORMAL: Column data test passed on table CAL_HISTORY column CAL_PARMS
NORMAL: Checking column CAL_TABLES
NORMAL: Column data test passed on table CAL_HISTORY column CAL_TABLES
NORMAL: Checking column CAL_SELECT
NORMAL: Column data test passed on table CAL_HISTORY column CAL_SELECT
NORMAL: Checking column CAL_NOTES
NORMAL: Column data test passed on table CAL_HISTORY column CAL_NOTES
T
```

---

tablecomparer.select - Function

Package utility

Module	**table**
Tool	**tablecomparer**

Specify which columns to check (or not to check)

Synopsis
select(checkonlycols, nocheckcols)

Description
This function allows the user to specify which columns to check or not to check when running `checkcolkeywordnames()` and `checkcoldata()`. The `checkonlycols` parameter indicates that these functions should check only the columns specified, while the `nocheckcols` parameter indicates that all columns except those specified should be checked. If both are specified, it is the same as specifying only the `checkonlycols` parameter. Each parameter can be either a vector of column names or a record of column names. In the case of a vector, a column of that name in either the main table or any of its subtables matches (assuming the `check*` function is run with `checksubtables=T`. In the case of a record being used for these parameters, the field names of the record are the table labels, and each field can be assigned a vector of column names. So for examaple to check only the TIME and GAIN column of the main table and the SPECTRAL_WINDOW_ID column of the CAL_DESC subtable you would do (bearing in mind that the defalt label for the top-level table is `TABLE`):

```
tc := tablecomparer('test.table','fiducial.table')
occ := [=];
occ.MAIN := ['TIME','GAIN'];
occ.CAL_DESC := 'SPECTRAL_WINDOW_ID'
tc.select(checkonlycols=occ)
tc.checkcoldata();
```

Arguments

checkonlycols	Check only these columns Allowed:Array or record Default: []
nocheckcols	Do not check these columns (not used if checkonlycols is specified) Allowed:Array or record Default: []

Returns
T

Example

```
- tc := tablecomparer('test.table','fiducial.table', verbose=2)
# check only the GAIN column of the main table
- occ := [=]
- occ.TABLE := 'GAIN';
- tc.select(occ);
- tc.checkcoldata();
NORMAL: Skipping column TIME of table TABLE because of selection.
NORMAL: Skipping column TIME_EXTRA_PREC of table TABLE because of selection.
NORMAL: Skipping column INTERVAL of table TABLE because of selection.
NORMAL: Skipping column ANTENNA1 of table TABLE because of selection.
NORMAL: Skipping column FEED1 of table TABLE because of selection.
NORMAL: Skipping column FIELD_ID of table TABLE because of selection.
NORMAL: Skipping column ARRAY_ID of table TABLE because of selection.
NORMAL: Skipping column OBSERVATION_ID of table TABLE because of selection.
NORMAL: Skipping column SCAN_NUMBER of table TABLE because of selection.
NORMAL: Skipping column PROCESSOR_ID of table TABLE because of selection.
NORMAL: Skipping column STATE_ID of table TABLE because of selection.
NORMAL: Skipping column PHASE_ID of table TABLE because of selection.
NORMAL: Skipping column PULSAR_BIN of table TABLE because of selection.
NORMAL: Skipping column PULSAR_GATE_ID of table TABLE because of selection.
NORMAL: Skipping column FREQ_GROUP of table TABLE because of selection.
NORMAL: Skipping column FREQ_GROUP_NAME of table TABLE because of selection.
NORMAL: Skipping column FIELD_NAME of table TABLE because of selection.
NORMAL: Skipping column FIELD_CODE of table TABLE because of selection.
NORMAL: Skipping column SOURCE_NAME of table TABLE because of selection.
NORMAL: Skipping column SOURCE_CODE of table TABLE because of selection.
NORMAL: Skipping column CALIBRATION_GROUP of table TABLE because of selection.
NORMAL: Column data test passed on table TABLE column GAIN
NORMAL: Skipping column REF_ANT of table TABLE because of selection.
NORMAL: Skipping column REF_FEED of table TABLE because of selection.
NORMAL: Skipping column REF_RECEPTOR of table TABLE because of selection.
NORMAL: Skipping column REF_FREQUENCY of table TABLE because of selection.
NORMAL: Skipping column MEAS_FREQ_REF of table TABLE because of selection.
NORMAL: Skipping column REF_DIRECTION of table TABLE because of selection.
NORMAL: Skipping column MEAS_DIR_REF of table TABLE because of selection.
NORMAL: Skipping column CAL_DESC_ID of table TABLE because of selection.
NORMAL: Skipping column CAL_HISTORY_ID of table TABLE because of selection.
NORMAL: Skipping column TOTAL_SOLUTION_OK of table TABLE because of selection.
NORMAL: Skipping column TOTAL_FIT of table TABLE because of selection.
NORMAL: Skipping column TOTAL_FIT_WEIGHT of table TABLE because of selection.
NORMAL: Skipping column SOLUTION_OK of table TABLE because of selection.
NORMAL: Skipping column FIT of table TABLE because of selection.
NORMAL: Skipping column FIT_WEIGHT of table TABLE because of selection.
```

```
NORMAL: successful readonly open of default-locked table test.table/CAL_DESC: 12 columns, 1 rows
NORMAL: successful readonly open of default-locked table fiducial.table/CAL_DESC: 12 columns, 1 rows
NORMAL: Skipping column NUM_SPW of table CAL_DESC because of selection.
NORMAL: Skipping column NUM_CHAN of table CAL_DESC because of selection.
NORMAL: Skipping column NUM_RECEPTORS of table CAL_DESC because of selection.
NORMAL: Skipping column N_JONES of table CAL_DESC because of selection.
NORMAL: Skipping column SPECTRAL_WINDOW_ID of table CAL_DESC because of selection.
NORMAL: Skipping column CHAN_FREQ of table CAL_DESC because of selection.
NORMAL: Skipping column MEAS_FREQ_REF of table CAL_DESC because of selection.
NORMAL: Skipping column CHAN_WIDTH of table CAL_DESC because of selection.
NORMAL: Skipping column CHAN_RANGE of table CAL_DESC because of selection.
NORMAL: Skipping column POLARIZATION_TYPE of table CAL_DESC because of selection.
NORMAL: Skipping column JONES_TYPE of table CAL_DESC because of selection.
NORMAL: Skipping column MS_NAME of table CAL_DESC because of selection.
NORMAL: successful readonly open of default-locked table test.table/CAL_HISTORY: 4 columns, 0 rows
NORMAL: successful readonly open of default-locked table fiducial.table/CAL_HISTORY: 4 columns, 0 rows
NORMAL: Skipping column CAL_PARMS of table CAL_HISTORY because of selection.
NORMAL: Skipping column CAL_TABLES of table CAL_HISTORY because of selection.
NORMAL: Skipping column CAL_SELECT of table CAL_HISTORY because of selection.
NORMAL: Skipping column CAL_NOTES of table CAL_HISTORY because of selection.
T
```

---

tablecomparer.tolerance - Function

Package	**utility**
Module	**table**
Tool	**tablecomparer**

Get the tolerance of the specified column

Synopsis
tolerance(col, tlabel)

Description
This function returns the tolerance of the specified column in the specified table as it was set by `settolerance()` (0 if no set tolerance applies to the specified column).

Arguments

col	The name of the column to get the tolerance of Allowed:A valid column name Default: None
tlabel	The label of the table or subtable in which this column is Allowed:A valid table label Default: ” (the top level table)

Returns
T

Example

```
- tc := tablecomparer('test.table','fiducial.table')
- tol := [=];
- tol.TABLE.DCOL := 1e-8;
- tol.GTABLE.GCOL := 1e-5;
- tc.setprecision(double=1e-9, cols=tol);
- tc.tolerance('DCOL')
1e-8
- tc.tolerance('GCOL','GTABLE')
1e-05
- # CCOL is a column of type double
- tc.tolerance('CCOL')
1e-9
- # HCOL is a column of type float
- tc.tolerance('HCOL')
0
```

---

tablecomparer.compare - Function

Package	**utility**
Module	**table**
Tool	**tablecomparer**

Compare tables by running one or more tests

Synopsis
compare(checksubtables, checkcolnames, checkcolkeywordnames, checkcoldata, checktablekeywordnames)

Description
This function runs one or more `check*` tests on the table. If all tests pass, it returns `T`, it returns `F` otherwise.

Arguments

checksubtables	Check subtables as well? Allowed:Boolean Default: T
checkcolnames	Run the column names test? Allowed:Boolean Default: T
checkcolkeywordnames	Run the column keyword names test? Allowed:Boolean Default: T
checkcoldata	Run the column data test? Allowed:Boolean Default: T
checktablekeywordnames	Run the table keyword names test? Allowed:Boolean Default: T

Returns
T if all specified tests pass, F otherwise

Example

```
- tc := tablecomparer('test.table','fiducial.table', verbose=2)
- # run all available tests
- tc.compare();
<lots of messages>
T
```

---

tablecomparer.details - Function

Package	**utility**
Module	**table**
Tool	**tablecomparer**

Get details regarding failed tests

Synopsis
details()

Description
This function returns a record containing the details of failed tests. Currently, this record contains information only column data tests which fail. In the case of homogenous array data, this record contains `test`, `model`, and `diff` fields. These fields themselves are records containing fields of the form `LABEL.COLUMN` which hold the actual column data of of the test and model and the difference of these columns, respectively. In the case of heterogeneous (variable) array columns, the diff record is absent. Thus, the `test.TABLE.DATA` field contains the DATA column of the main table of the test dataset on which the DATA column test failed.

Arguments

Returns
Record

Example

```
- tc := tablecomparer('test.table','fiducial.table', verbose=2)
- tc.compare();
- # the DATA column test of the IMAGE subtable failed. Stuff that
- # column into a glish array for inspection
- d := tc.details();
- tdata := d.test.IMAGE.DATA
```

---

tablecomparer.summary - Function

Package	**utility**
Module	**table**
Tool	**tablecomparer**

Summarize the comparisons

Synopsis
summary(dolog)

Description
This function returns a summary of the results of comparisons that have been run and optionally outputs them to the logger. A value of `T` means that a test passed, a value of `F` means that it failed.

Arguments

dolog	Write summary to logger? Allowed:Boolean Default: F

Returns
Record

Example

```
- tc := tablecomparer('test.table','fiducial.table', verbose=2)
- tc.compare();
- results := tc.summary(T);
NORMAL: TABLE.COLNAMES: T
NORMAL: TABLE.COLKEYWORDNAMES.TIME: T
NORMAL: TABLE.COLKEYWORDNAMES.TIME_EXTRA_PREC: T
NORMAL: TABLE.COLKEYWORDNAMES.INTERVAL: T
NORMAL: TABLE.COLKEYWORDNAMES.ANTENNA1: T
NORMAL: TABLE.COLKEYWORDNAMES.FEED1: T
NORMAL: TABLE.COLKEYWORDNAMES.FIELD_ID: T
NORMAL: TABLE.COLKEYWORDNAMES.ARRAY_ID: T
NORMAL: TABLE.COLKEYWORDNAMES.OBSERVATION_ID: T
NORMAL: TABLE.COLKEYWORDNAMES.SCAN_NUMBER: T
NORMAL: TABLE.COLKEYWORDNAMES.PROCESSOR_ID: T
NORMAL: TABLE.COLKEYWORDNAMES.STATE_ID: T
NORMAL: TABLE.COLKEYWORDNAMES.PHASE_ID: T
NORMAL: TABLE.COLKEYWORDNAMES.PULSAR_BIN: T
NORMAL: TABLE.COLKEYWORDNAMES.PULSAR_GATE_ID: T
NORMAL: TABLE.COLKEYWORDNAMES.FREQ_GROUP: T
NORMAL: TABLE.COLKEYWORDNAMES.FREQ_GROUP_NAME: T
NORMAL: TABLE.COLKEYWORDNAMES.FIELD_NAME: T
NORMAL: TABLE.COLKEYWORDNAMES.FIELD_CODE: T
NORMAL: TABLE.COLKEYWORDNAMES.SOURCE_NAME: T
```

```
NORMAL: TABLE.COLKEYWORDNAMES.SOURCE_CODE: T
NORMAL: TABLE.COLKEYWORDNAMES.CALIBRATION_GROUP: T
NORMAL: TABLE.COLKEYWORDNAMES.GAIN: T
NORMAL: TABLE.COLKEYWORDNAMES.REF_ANT: T
NORMAL: TABLE.COLKEYWORDNAMES.REF_FEED: T
NORMAL: TABLE.COLKEYWORDNAMES.REF_RECEPTOR: T
NORMAL: TABLE.COLKEYWORDNAMES.REF_FREQUENCY: T
NORMAL: TABLE.COLKEYWORDNAMES.MEAS_FREQ_REF: T
NORMAL: TABLE.COLKEYWORDNAMES.REF_DIRECTION: T
NORMAL: TABLE.COLKEYWORDNAMES.MEAS_DIR_REF: T
NORMAL: TABLE.COLKEYWORDNAMES.CAL_DESC_ID: T
NORMAL: TABLE.COLKEYWORDNAMES.CAL_HISTORY_ID: T
NORMAL: TABLE.COLKEYWORDNAMES.TOTAL_SOLUTION_OK: T
NORMAL: TABLE.COLKEYWORDNAMES.TOTAL_FIT: T
NORMAL: TABLE.COLKEYWORDNAMES.TOTAL_FIT_WEIGHT: T
NORMAL: TABLE.COLKEYWORDNAMES.SOLUTION_OK: T
NORMAL: TABLE.COLKEYWORDNAMES.FIT: T
NORMAL: TABLE.COLKEYWORDNAMES.FIT_WEIGHT: T
NORMAL: TABLE.COLVALUES.TIME: T
NORMAL: TABLE.COLVALUES.TIME_EXTRA_PREC: T
NORMAL: TABLE.COLVALUES.INTERVAL: T
NORMAL: TABLE.COLVALUES.ANTENNA1: T
NORMAL: TABLE.COLVALUES.FEED1: T
NORMAL: TABLE.COLVALUES.FIELD_ID: T
NORMAL: TABLE.COLVALUES.ARRAY_ID: T
NORMAL: TABLE.COLVALUES.OBSERVATION_ID: T
NORMAL: TABLE.COLVALUES.SCAN_NUMBER: T
NORMAL: TABLE.COLVALUES.PROCESSOR_ID: T
NORMAL: TABLE.COLVALUES.STATE_ID: T
NORMAL: TABLE.COLVALUES.PHASE_ID: T
NORMAL: TABLE.COLVALUES.PULSAR_BIN: T
NORMAL: TABLE.COLVALUES.PULSAR_GATE_ID: T
NORMAL: TABLE.COLVALUES.FREQ_GROUP: T
NORMAL: TABLE.COLVALUES.FREQ_GROUP_NAME: T
NORMAL: TABLE.COLVALUES.FIELD_NAME: T
NORMAL: TABLE.COLVALUES.FIELD_CODE: T
NORMAL: TABLE.COLVALUES.SOURCE_NAME: T
NORMAL: TABLE.COLVALUES.SOURCE_CODE: T
NORMAL: TABLE.COLVALUES.CALIBRATION_GROUP: T
NORMAL: TABLE.COLVALUES.GAIN: T
NORMAL: TABLE.COLVALUES.REF_ANT: T
NORMAL: TABLE.COLVALUES.REF_FEED: T
NORMAL: TABLE.COLVALUES.REF_RECEPTOR: T
NORMAL: TABLE.COLVALUES.REF_FREQUENCY: T
NORMAL: TABLE.COLVALUES.MEAS_FREQ_REF: T
NORMAL: TABLE.COLVALUES.REF_DIRECTION: T
```

```
NORMAL: TABLE.COLVALUES.MEAS_DIR_REF: T
NORMAL: TABLE.COLVALUES.CAL_DESC_ID: T
NORMAL: TABLE.COLVALUES.CAL_HISTORY_ID: T
NORMAL: TABLE.COLVALUES.TOTAL_SOLUTION_OK: T
NORMAL: TABLE.COLVALUES.TOTAL_FIT: T
NORMAL: TABLE.COLVALUES.TOTAL_FIT_WEIGHT: T
NORMAL: TABLE.COLVALUES.SOLUTION_OK: T
NORMAL: TABLE.COLVALUES.FIT: T
NORMAL: TABLE.COLVALUES.FIT_WEIGHT: T
NORMAL: TABLE.KEYWORDFIELDNAMES: T
NORMAL: CAL_DESC.COLNAMES: T
NORMAL: CAL_DESC.COLKEYWORDNAMES.NUM_SPW: T
NORMAL: CAL_DESC.COLKEYWORDNAMES.NUM_CHAN: T
NORMAL: CAL_DESC.COLKEYWORDNAMES.NUM_RECEPTORS: T
NORMAL: CAL_DESC.COLKEYWORDNAMES.N_JONES: T
NORMAL: CAL_DESC.COLKEYWORDNAMES.SPECTRAL_WINDOW_ID: T
NORMAL: CAL_DESC.COLKEYWORDNAMES.CHAN_FREQ: T
NORMAL: CAL_DESC.COLKEYWORDNAMES.MEAS_FREQ_REF: T
NORMAL: CAL_DESC.COLKEYWORDNAMES.CHAN_WIDTH: T
NORMAL: CAL_DESC.COLKEYWORDNAMES.CHAN_RANGE: T
NORMAL: CAL_DESC.COLKEYWORDNAMES.POLARIZATION_TYPE: T
NORMAL: CAL_DESC.COLKEYWORDNAMES.JONES_TYPE: T
NORMAL: CAL_DESC.COLKEYWORDNAMES.MS_NAME: T
NORMAL: CAL_DESC.COLVALUES.NUM_SPW: T
NORMAL: CAL_DESC.COLVALUES.NUM_CHAN: T
NORMAL: CAL_DESC.COLVALUES.NUM_RECEPTORS: T
NORMAL: CAL_DESC.COLVALUES.N_JONES: T
NORMAL: CAL_DESC.COLVALUES.SPECTRAL_WINDOW_ID: T
NORMAL: CAL_DESC.COLVALUES.CHAN_FREQ: T
NORMAL: CAL_DESC.COLVALUES.MEAS_FREQ_REF: T
NORMAL: CAL_DESC.COLVALUES.CHAN_WIDTH: T
NORMAL: CAL_DESC.COLVALUES.CHAN_RANGE: T
NORMAL: CAL_DESC.COLVALUES.POLARIZATION_TYPE: T
NORMAL: CAL_DESC.COLVALUES.JONES_TYPE: T
NORMAL: CAL_DESC.COLVALUES.MS_NAME: T
NORMAL: CAL_DESC.KEYWORDFIELDNAMES: T
NORMAL: CAL_HISTORY.COLNAMES: T
NORMAL: CAL_HISTORY.COLKEYWORDNAMES.CAL_PARMS: T
NORMAL: CAL_HISTORY.COLKEYWORDNAMES.CAL_TABLES: T
NORMAL: CAL_HISTORY.COLKEYWORDNAMES.CAL_SELECT: T
NORMAL: CAL_HISTORY.COLKEYWORDNAMES.CAL_NOTES: T
NORMAL: CAL_HISTORY.COLVALUES.CAL_PARMS: T
NORMAL: CAL_HISTORY.COLVALUES.CAL_TABLES: T
NORMAL: CAL_HISTORY.COLVALUES.CAL_SELECT: T
NORMAL: CAL_HISTORY.COLVALUES.CAL_NOTES: T
NORMAL: CAL_HISTORY.KEYWORDFIELDNAMES: T
```

```
- print results;
[TABLE=[COLNAMES=T, COLKEYWORDNAMES=[TIME=T, TIME_EXTRA_PREC=T,
INTERVAL=T, ANTENNA1=T, FEED1=T, FIELD_ID=T, ARRAY_ID=T,
OBSERVATION_ID=T, SCAN_NUMBER=T, PROCESSOR_ID=T, STATE_ID=T,
PHASE_ID=T, PULSAR_BIN=T, PULSAR_GATE_ID=T, FREQ_GROUP=T,
FREQ_GROUP_NAME=T, FIELD_NAME=T, FIELD_CODE=T, SOURCE_NAME=T,
SOURCE_CODE=T, CALIBRATION_GROUP=T, GAIN=T, REF_ANT=T, REF_FEED=T,
REF_RECEPTOR=T, REF_FREQUENCY=T, MEAS_FREQ_REF=T, REF_DIRECTION=T,
MEAS_DIR_REF=T, CAL_DESC_ID=T, CAL_HISTORY_ID=T, TOTAL_SOLUTION_OK=T,
TOTAL_FIT=T, TOTAL_FIT_WEIGHT=T, SOLUTION_OK=T, FIT=T, FIT_WEIGHT=T],
COLVALUES=[TIME=T, TIME_EXTRA_PREC=T, INTERVAL=T, ANTENNA1=T, FEED1=T,
FIELD_ID=T, ARRAY_ID=T, OBSERVATION_ID=T, SCAN_NUMBER=T,
PROCESSOR_ID=T, STATE_ID=T, PHASE_ID=T, PULSAR_BIN=T,
PULSAR_GATE_ID=T, FREQ_GROUP=T, FREQ_GROUP_NAME=T, FIELD_NAME=T,
FIELD_CODE=T, SOURCE_NAME=T, SOURCE_CODE=T, CALIBRATION_GROUP=T,
GAIN=T, REF_ANT=T, REF_FEED=T, REF_RECEPTOR=T, REF_FREQUENCY=T,
MEAS_FREQ_REF=T, REF_DIRECTION=T, MEAS_DIR_REF=T, CAL_DESC_ID=T,
CAL_HISTORY_ID=T, TOTAL_SOLUTION_OK=T, TOTAL_FIT=T,
TOTAL_FIT_WEIGHT=T, SOLUTION_OK=T, FIT=T, FIT_WEIGHT=T],
KEYWORDFIELDNAMES=T], CAL_DESC=[COLNAMES=T,
COLKEYWORDNAMES=[NUM_SPW=T, NUM_CHAN=T, NUM_RECEPTORS=T, N_JONES=T,
SPECTRAL_WINDOW_ID=T, CHAN_FREQ=T, MEAS_FREQ_REF=T, CHAN_WIDTH=T,
CHAN_RANGE=T, POLARIZATION_TYPE=T, JONES_TYPE=T, MS_NAME=T],
COLVALUES=[NUM_SPW=T, NUM_CHAN=T, NUM_RECEPTORS=T, N_JONES=T,
SPECTRAL_WINDOW_ID=T, CHAN_FREQ=T, MEAS_FREQ_REF=T, CHAN_WIDTH=T,
CHAN_RANGE=T, POLARIZATION_TYPE=T, JONES_TYPE=T, MS_NAME=T],
KEYWORDFIELDNAMES=T], CAL_HISTORY=[COLNAMES=T,
COLKEYWORDNAMES=[CAL_PARMS=T, CAL_TABLES=T, CAL_SELECT=T,
CAL_NOTES=T], COLVALUES=[CAL_PARMS=T, CAL_TABLES=T, CAL_SELECT=T,
CAL_NOTES=T], KEYWORDFIELDNAMES=T]]
```

1.7.24 tableplot - Tool

Package utility
Module table

Plot data from tables via Glish

Description

`tableplot` is a plotting tool for a general AIPS++ table. Table columns can be plotted against each other, and can be combined using the TaQL to create expressions for derived quantities, the result of which can then be plotted. Data from more than one table can be accessed and plotted at the same time. Expressions producing arrays result in overlay plots. The default plot style is single-panel, but if an iteration axis is specified, multi-panel plots are supported. Zooming and region-based flagging is possible on single and multi panel plots. A GUI, adapted to a particular kind of table (measurement set) can call the tableplot functions and customize the generated plots.

Constructors

tableplot	Construct a tableplot tool

Functions

clearflags	Clear all flags in the table
done	end the tableplot tool
flagdata	Flag Data for selected flag regions
iterplotnext	Start/Continue plotting
iterplotstart	Initialize plotting with an iteration axis
iterplotstop	Stop plot iterations.
markflags	Mark a rectangular region to flag
plotdata	Plot the result of a general TaQL expression
settables	Specify list of tables to operate on
zoomplot	Zoom/Unzoom on marked regions

tableplot - Constructor

Package	**utility**
Module	**table**
Tool	**tableplot**

Construct a tableplot tool

Synopsis
tableplot()

Description
Create a `tableplot` tool.

Returns
tableplot tool

Example

```
# create a tableplot tool
  tp := tableplot();
```

---

tableplot.settables - Function

Package	**utility**
Module	**table**
Tool	**tableplot**

Specify list of tables to operate on

Synopsis
settables(tabnames)

Description
Specify a list of table names. All plots will operate on these tables.

Arguments

tabnames	List of strings : Table names Allowed:Vector of Strings Default: no default

Returns
0 if tables are valid, -1 otherwise

Example

```
# create a tableplot tool and set the list of tables to plot from.
  tp := tableplot();
  tp.settables(tabnames=['3c273.ms','3c48.ms']);
```

---

tableplot.plotdata - Function

Package	**utility**
Module	**table**
Tool	**tableplot**

Plot the result of a general TaQL expression

Synopsis
plotdata(poption, labels, datastr)

Description
This function evaluates the specified TaQL expressions for the X and Y axes of a two-dimensional scatter plot, extracts the resulting columns, and plots them. TaQL expressions resulting in arrays result in overlay plots. If multiple tables are specified in `settables`, then the TaQL expressions are applied to all tables and overlay plots are generated. Currently only single panel plots are supported for this function.
Valid TaQL strings must satisfy the following conditions.

1. Each TaQL string must result in a Double scalar or array.
   `'AMPLITUDE(DATA[1,1])'` results in a Double scalar (valid).
   `'AMPLITUDE(DATA[1:2,1])'` results in a Double array (valid).
   `'MEAN(AMPLITUDE(DATA[1:2,1]))'` results in a Double scalar (valid).
   `'DATA[1,1]'` results in a Complex scalar (NOT valid).
   `'AMPLITUDE(DATA[1,1])<10'` results in a Bool scalar (NOT valid).

2. All TaQL functions resulting in Double Scalars/Arrays are allowed, except for those involving an explicit collapse axis (means,sums,etc..). Note that these functions are different from mean,sum,etc.. which are supported.

3. TaQL strings must be provided as pairs of strings, with the X-TaQL first, followed by the Y-TaQL. There are 3 cases.
   X-TaQL − Scalar, Y-TaQL − Scalar (one-to-one single plot)
   X-TaQL − Scalar, Y-TaQL − Array (one-to-many overlay plot)
   X-TaQL − Array, Y-TaQL − Array (if the shapes are the same, then a one-to-one mapping is done, otherwise only the first X-TaQL result is used for a one-to-many mapping with the Y-TaQL Array.)

4. For cross plots (for example amplitude vs channel plots in an MS), the X-TaQL must be a string containing 'CROSS'. The Y-TaQL is used to read out the data from the table, and the x-values are the column indices (channel numbers) chosen by the Y-TaQL.

Various plotting options can be set as follows.

```
plotopts.nxpanels := 1;      # number of panels along the horizontal axis.
plotopts.nypanels := 1;      # number of panels along the vertical axis.
plotopts.windowsize := 5;    # horizontal size of the pgplot window (inches)
plotopts.aspectratio := 0.8; # aspect-ratio of the pgplot window (dx/dy)
plotopts.fontsize := 1.0;    # font size of labels
```

```
plotopts.plotcolour := 21;   # first digit : colour, second digit : colour increment (overlays)
plotopts.plotsymbol := 5;    # plot each point with a cross
plotopts.linewidth := 2;     # line width (all lines)
plotopts.plotrange := [xmin,xmax,ymin,ymax] # specified plot range over-rides the data range if
                                            # it lies within the data range.
plotopts.useflags := 0;      # 0:plot unflagged data, 1:plot flagged data,
                             # 2:plot both flagged and unflagged data (different colours)
```

Plot labelling is controlled from glish.

Arguments

poption	Record of plot options Allowed:Record Default: nxpanels=1,nypanels=1,windowsize=6, aspectratio=0.8,fontsize=1.0, plotcolour=10, plotsymbol=1, linewidth=2, plotrange=[-1e+30,1e+30,-1e+30,1e+30],useflags=0
labels	List of strings : Title,Xlabel,Ylabel Allowed:Vector of Strings Default: no default
datastr	List of TaQL strings : X,Y Allowed:Vector of Strings Default: no default

Returns
1 if plotting is successful, -1 otherwise

Example

```
# create a tableplot tool, set tables and plot amplitude vs uvdist
# for two Measurement Set tables.
# Operate on channel 1 for stokes 1 and 2 using the DATA column and
# produce an overlay plot. Use two different colours for each overlay
# and plot both flagged and unflagged data in separate colours.

  tp := tableplot();
  tp.settables(tabnames=['3c273.ms','3c48.ms']);

  plotopts.nxpanels := 1;
  plotopts.nypanels := 1;
  plotopts.windowsize := 5;
  plotopts.aspectratio := 0.8;
```

```
plotopts.fontsize := 1.0;
plotopts.plotcolour := 21;
plotopts.plotstyle := 2;
plotopts.linewidth := 2;
plotopts.timeplot := 1;
plotopts.plotsymbol := 5;
plotopts.plotrange := [1000.0,2000.0,100.0,5000.0];
plotopts.useflags:=2;

labels := ['Amplitude vs UVdist','uvdist','amplitude'];
xystr := ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1:2,1])'];
tp.plotdata(poption=plotopts,labels=labels,datastr=xystr);
```

TaQL strings for the above example can also be written as follows.

```
xystr := ['SQRT(UVW[1]*UVW[1]+UVW[2]*UVW[2])','AMPLITUDE(DATA[1:2,1])'];
```

Multiple X-TaQL, Y-TaQL pairs can be given and result in an overlay plot. A plot of UV coverage can be obtained by an overlay plot as follows.

```
plotopts.nxpanels := 1;
plotopts.nypanels := 1;
plotopts.aspectratio := 1.0;
labels := ['UV coverage','u','v'];
xystr := ['UVW[1]','UVW[2]','-1*UVW[1]','-1*UVW[2]'];
tp.plotdata(poption=plotopts,labels=labels,datastr=xystr);
```

TaQL strings for cross plots (amplitude vs channel number) can be written as follows.

```
xystr := ['CROSS','AMPLITUDE(DATA[1,1:32])'];
```

---

tableplot.iterplotstart - Function

Package	**utility**
Module	**table**
Tool	**tableplot**

Initialize plotting with an iteration axis

Synopsis
iterplotstart(poption, labels, datastr, iteraxes)

Description
Begin a series of plots using subtables constructed via an iteration axes. In addition to plotdata parameters, set a vector string of iteration axes (Table column names) and use iterplotnext() to step through. Multi-panel plots are supported with this function. Only forward step through is allowed.

Arguments

poption	Record of plot options Allowed:Record Default: nxpanels=1,nypanels=1,windowsize=6, aspectratio=0.8,fontsize=1.0
labels	List of strings : Title,Xlabel,Ylabel Allowed:Vector of Strings Default: no default
datastr	List of TaQL strings : X,Y Allowed:Vector of Strings Default: no default
iteraxes	List of strings : Iteration axes Allowed:Vector of Strings Default: no default

Returns
0 if plot initialization is successful, -1 otherwise

Example

```
# create a tableplot tool, set the list of MS tables to plot from,
# and initialize a plot of Amplitude vs UV distance for
# channel 1 and stokes 1, iterating over Antenna1, and creating
# two plot panels per iteration page.
  tp := tableplot();
  tp.settables(tabnames=['3c273.ms','3c48.ms']);
  plotopts.nxpanels := 1;
  plotopts.nypanels := 2;
  plotopts.windowsize := 6;
  plotopts.aspectratio := 1.2;
  plotopts.fontsize := 2.0;
  labels := ['Amplitude vs UVdist (iterating over Antenna1)','uvdist','amplitude'];
  xystr := ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1,1])'];
  iteraxes := ['ANTENNA1'];
  tp.iterplotstart(poption=plotopts,labels=labels,datastr=xystr,iteraxes=iteraxes);
```

To iterate over baseline and plot Amplitude vs time, for stokes 1, channel 1.

```
plotopts.nxpanels := 1;
plotopts.nypanels := 4;
labels := ['Amplitude vs UVdist (iterating over Baseline)','uvdist','amplitude'];
xystr := ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1,1])'];
iteraxes := ['ANTENNA1','ANTENNA2'];
tp.iterplotstart(poption=plotopts,labels=labels,datastr=xystr,iteraxes=iteraxes);
```

---

tableplot.iterplotnext - Function

Package	**utility**
Module	**table**
Tool	**tableplot**

Start/Continue plotting

Synopsis
iterplotnext()

Description
Start/Continue plotting by stepping through the iteration axes.

Returns
1 if additional iteration steps remain , 1 if last iteration has completed

Example

```
# iterate through the data
  tp := tableplot();
  tp.settables(tabnames=['3c273.ms','3c48.ms']);
  plotopts.nxpanels := 1;
  plotopts.nypanels := 2;
  plotopts.windowsize := 6;
  plotopts.aspectratio := 1.2;
  plotopts.fontsize := 2.0;
  labels := ['Amplitude vs UVdist (iterating over Antenna1)','uvdist','amplitude'];
  xystr := ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1,1])'];
  iteraxes := ['ANTENNA1'];
```

```
  tp.iterplotstart(poption=plotopts,labels=labels,datastr=xystr,iteraxes=iteraxes);
  ret := tp.iterplotnext();
  ret := tp.iterplotnext();
  ret := tp.iterplotnext();
...
```

---

tableplot.iterplotstop - Function

Package	**utility**
Module	**table**
Tool	**tableplot**

Stop plot iterations.

Synopsis
iterplotstop()

Description
To be called at the end of the plot iterations, or in between if desired.

Arguments

Returns
0 if all is valid, -1 otherwise

Example

```
# iterate through and stop after 5 iterations of 2 plots per page
  tp := tableplot();
  tp.settables(tabnames=['3c273.ms','3c48.ms']);
  plotopts.nxpanels := 1;
  plotopts.nypanels := 2;
  labels := ['Amplitude vs UVdist (iterating over Antenna1)','uvdist','amplitude'];
  xystr := ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1,1])'];
  iteraxes := ['ANTENNA1'];
  tp.iterplotstart(poption=plotopts,labels=labels,datastr=xystr,iteraxes=iteraxes);
```

```
ret := tp.iterplotnext();
ret := tp.iterplotnext();
ret := tp.iterplotnext();
ret := tp.iterplotnext();
ret := tp.iterplotnext();
tp.iterplotstop();
```

---

tableplot.markflags - Function

Package	**utility**
Module	**table**
Tool	**tableplot**

Mark a rectangular region to flag

Synopsis
markflags(panel)

Description
Mark a rectangular region to be flagged. Each call to `markflags` allows one region to be drawn. Any number of successive calls can be made. In the case of multi-panel plots, a panel number must be specified with each call. This function marks and stores a list of marked regions. To apply flags, the `flagdata` function must be used. The panel number corresponds to a row-major ordering of panels.
Marking the region requires two consecutive mouse clicks at the two diagonally opposite corners. A hatched rectangle will appear over the selected region. (Note mouse dragging to mark a region with an expanding box, is not supported yet).

Arguments

panel	Panel number Allowed:Integer Default: 1

Returns
0 if successful, -1 otherwise

Example

```
# mark 2 flag regions on a multi-panel plot, one in panel 1 and one
# in panel 2.
  tp.markflags(panel=1);
  tp.markflags(panel=2);
```

---

tableplot.flagdata - Function

Package	**utility**
Module	**table**
Tool	**tableplot**

Flag Data for selected flag regions

Synopsis
flagdata(diskwrite, rowflag)

Description
Set flags for all regions marked using `markflags()`. The plot is automatically redrawn after applying flags.
If reduction TaQL functions such as `sum,mean` are used, flags corresponding to all accessed values will be modified. For example, with a measurement set table, flagging on the mean amplitude of stokes 1 and channels 1 to 5, given by 'MEAN(AMPLITUDE(DATA[1,1:5]))' results in flags being set for all 5 accessed channels.
For a measurement set, by default, flags are set only for accessed channels and stokes when the DATA column is used. However all channels/stokes can be flagged for the marked flag regions by setting the corresponding row flag.

Arguments

diskwrite	1 to write flags to the table on disk, 0 to set dummy flags Allowed:Integer Default: 0
rowflag	1 to set a row-flag in addition to individual data flags, 0 to ignore row flags if individual flags exist Allowed:Integer Default: 0

Returns
0 if flagging is successful, -1 otherwise

Example

```
# mark 2 flag regions on a multi-panel plot, one in panel 1 and one
# in panel 2. Then apply the flags and write to disk.
  tp.markflags(panel=1);
  tp.markflags(panel=2);
  tp.flagdata(diskwrite=1,rowflag=0);
```

---

tableplot.zoomplot - Function

Package	**utility**
Module	**table**
Tool	**tableplot**

Zoom/Unzoom on marked regions

Synopsis
zoomplot(panel, direction)

Description
Allows a chosen region to expand to fill in the plot view panel. Multiple zoom calls are allowed. One call to unzoom returns to the original plot. In multi-panel plots, zooming can be done separately on each panel.

Arguments

panel	Panel number Allowed:Integer Default: 1
direction	1 to zoom, 0 to unzoom Allowed:Integer Default: 1

Returns
0 if region is valid and zoom successful, -1 otherwise

Example

```
# zoom and then unzoom on panel 2
  tp.zoomplot(panel=2,direction=1);
  tp.zoomplot(panel=2,direction=0);
```

Flagging and zooming can be interleaved, i.e. flag regions can be marked on a zoomed plot using `markflags()`, and applied using `flagdata()` before or after unzooming.

```
# Zoom on panel 1 and perform flagging.
  tp.zoomplot(panel=1,direction=1);
  tp.markflags(panel=1);
  tp.markflags(panel=1);
  tp.flagdata(diskwrite=1,rowflag=0);
  tp.zoomplot(panel=1,direction=0);
```

---

tableplot.clearflags - Function

Package	**utility**
Module	**table**
Tool	**tableplot**

Clear all flags in the table

Synopsis
clearflags()

Description
Currently, this function clears all flags from the table. This will be modified to allow for selective un-flagging of previously flagged regions (specified by indexing into a stored history of marked flag-regions).

Returns
0 if successful, -1 otherwise

Example

```
# clear all flags from two measurement set tables
  tp := tableplot();
  tp.settables(tabnames=['3c273.ms','3c48.ms']);
```

```
tp.clearflags();
tp.done();
```

---

tableplot.done - Function

Package	**utility**
Module	**table**
Tool	**tableplot**

end the tableplot tool

Synopsis
done()

Description
End the tableplot tool

Returns
0 if successful, -1 otherwise