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(columname, 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

columname	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