Arguments

target	Target to be constructed Allowed:String	
args	Arguments Allowed:String	
makefile	Makefile to be used Allowed:File name Default: 'makefile'	
Execute	Execute commands Allowed:Bool Default: T	
script	Name of script file to be written Allowed:String	
debug	Write debugging information? Allowed:Bool Default: F	

Returns
Bool

Example

```
include 'make.g';
make('clean');
make('data');
make('3C273XC1.image');
```

1.6.32 executive - Tool

Package utility
Module misc

Tool for running scripts in a queue
include "executive.g"

Description

`executive` implements a queue for executing commands or scripts:

- Command sequences may be specified - these are written to a script for execution.
- Scripts are queued and run in order of submission.
- Running scripts may be killed, and queued scripts may be deleted from the queue.
- The scripts may be specified for execution on another host.
- A log file may be optionally specified.
- The number of concurrently running jobs allowed may be specified.

Important *Note that the scripts run in a separate instance of glish. Thus variables such as tools defined in the current glish session will not be available inside the script.*

Constructors

executive	Construct a executive tool

Functions

concurrent	Return number of concurrent jobs allowed
done	Terminate the executive
gui	Open a GUI for this tool
list	Return the queue status
make	Run a makefile
run	Run a sequence of commands
running	Return number of running jobs
runscript	Run a script
setconcurrent	Set number of concurrent jobs allowed
show	Show the queue status
stop	Stop a specified job
test	Run a test for this tool

executive - Constructor

Package	**utility**
Module	**misc**
Tool	**executive**

Construct a executive tool

Synopsis
executive(concurrent)

Description
Construct a executive tool for subsequent use. A tool called `defaultexecutive` is constructed automatically. A useful shorthand for `defaultexecutive` is `dex`.

Arguments

concurrent		Number of concurrent jobs Allowed:Int Default: 1

Returns
executive tool

Example

```
include 'executive.g';
dex.runscript('doit.g');
```

---

executive.runscript - Function

Package	**utility**
Module	**misc**
Tool	**executive**

Run a script

Synopsis
runscript(script, host, log)

Description
Run a script, optionally on another host. Specification of a log file is optional - if not specified, output is logged in the current glish session.
Important The script must exit explicitly if `executive` is to detect the end of processing. Thus, you **must** put as the last executed line `exit(1)`.
Important *Note that the scripts run in a separate instance of glish. Thus variables such as tools defined in the current glish session will not be available inside the script.*

Arguments

script	Script to be executed Allowed:String Default: None
host	Host where script is to be run: default is current host Allowed:String Default: ”
log	Log file Allowed:String Default: ”

Returns
job number

Example

```
include 'executive.g';
myexecutive:=executive();
myexecutive.runscript('doit.g', host='zeus');
myexecutive.run('doitagain()', inc='ireallydoit.g', host='zeus');
```

---

executive.run - Function

Package	**utility**
Module	**misc**
Tool	**executive**

Run a sequence of commands

Synopsis
run(commands, inc, host, log)

Description
Run a sequence of commands, optionally on another host. Specification of a log file is optional - if not specified, output is logged in the current glish session. An include file may be specified.
The number returned is the job number, which is a monotonically increasing number starting at one. You can use this number to kill or remove a given job.

Important *Note that the scripts run in a separate instance of glish. Thus variables such as tools defined in the current glish session will not be available inside the script.*

Arguments

commands	Commands to be executed Allowed:String Default: None
inc	Include file to be invoked on start Allowed:String Default: ”
host	Host where script is to be run: default is current host Allowed:String Default: ”
log	Log file Allowed:String Default: ”

Returns
job number

Example

```
include 'executive.g';
myexecutive:=executive();
myexecutive.run('doit()', inc='ireallydoit.g', host='zeus');
```

---

executive.make - Function

Package	**utility**
Module	**misc**
Tool	**executive**

Run a makefile

Synopsis
make(target, args, makefile, host, log)

Description
Run make on a makefile and execute the resulting command, optionally on another host. Specification of a log file is optional - if not specified, output is logged in the current glish session.
The number returned is the job number, which is a monotonically increasing number starting at one. You can use this number to kill or remove a given job.
Important *Note that the scripts run in a separate instance of glish. Thus variables such as tools defined in the current glish session will not be available inside the script.*

Arguments

target	Target to be constructed Allowed:String
args	Arguments Allowed:String
makefile	Makefile to be used Allowed:File name Default: 'makefile'
host	Host where script is to be run: default is current host Allowed:String Default: ''
log	Log file Allowed:String Default: ''

Returns
job number

Example

```
include 'executive.g';
myexecutive:=executive();
myexecutive.run('doit()', inc='ireallydoit.g', host='zeus');
```

---

executive.done - Function

Package	**utility**
Module	**misc**
Tool	**executive**

Terminate the executive

Synopsis
done()

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

Returns
bool

Example

```
include 'executive.g';
myexecutive:=executive();
myexecutive.run('doit()', inc='ireallydoit.g', host='zeus');
#
# Wait for stuff to finish....
#
myexecutive.done();
```

---

executive.concurrent - Function

Package	**utility**
Module	**misc**
Tool	**executive**

Return number of concurrent jobs allowed

Synopsis
concurrent()

Description
Return the number of concurrent jobs allowed.

Returns
Int

Example

```
include 'executive.g';
myexecutive:=executive();
myexecutive.run('doit()', inc='ireallydoit.g', host='zeus');
print myexecutive.concurrent();
```

---

executive.running - Function

Package	**utility**
Module	**misc**
Tool	**executive**

Return number of running jobs

Synopsis
running()

Description
Return the number of running jobs.

Returns
Int

Example

```
include 'executive.g';
myexecutive:=executive();
myexecutive.run('doit()', inc='ireallydoit.g', host='zeus');
print myexecutive.running();
```

---

executive.setconcurrent - Function

Package	**utility**
Module	**misc**
Tool	**executive**

Set number of concurrent jobs allowed

Synopsis
setconcurrent(concurrent)

Description
Set the number of concurrent jobs allowed.

Arguments

concurrent	Number of concurrent jobs Allowed:Int Default: 1

Returns
Int

Example

```
include 'executive.g';
myexecutive:=executive();
myexecutive.setconcurrent(2);
myexecutive.run('doit()', inc='ireallydoit.g', host='zeus');
myexecutive.run('doitaswell()', inc='ireallydoit.g', host='zeus');
myexecutive.run('doiteventually()', inc='ireallydoit.g', host='zeus');
```

---

executive.show - Function

Package	**utility**
Module	**misc**
Tool	**executive**

Show the queue status

Synopsis
show()

Description
Show the current queue status

Returns
Bool

Example

```
include 'executive.g';
myexecutive:=executive();
myexecutive.setconcurrent(2);
myexecutive.run('doit()', inc='ireallydoit.g', host='zeus');
myexecutive.run('doitaswell()', inc='ireallydoit.g', host='zeus');
myexecutive.run('doiteventually()', inc='ireallydoit.g', host='zeus');
myexecutive.show();
```

---

executive.list - Function

Package	**utility**
Module	**misc**
Tool	**executive**

Return the queue status

Synopsis
list()

Description
Return the current queue status in a record

Returns
Record

Example

```
include 'executive.g';
```

```
myexecutive:=executive()
myexecutive.setconcurrent(2);
myexecutive.run('doit()', inc='ireallydoit.g', host='zeus');
myexecutive.run('doitaswell()', inc='ireallydoit.g', host='zeus');
myexecutive.run('doiteventually()', inc='ireallydoit.g', host='zeus');
print myexecutive.list();
```

---

executive.stop - Function

Package	**utility**
Module	**misc**
Tool	**executive**

Stop a specified job

Synopsis
stop(id)

Description
Stop a specified job, either by killing the glish client executing it (if it is running) or by removing it from the waiting queue. The job is identified by the job id number returned by `run` or `runscript`.

Arguments

id	Job id from run or runscript Allowed:Int Default: None

Returns
Bool

Example

```
include 'executive.g';
myexecutive:=executive();
myexecutive.setconcurrent(2);
myexecutive.run('doit()', inc='ireallydoit.g', host='zeus');
```

```
myexecutive.run('doitaswell()', inc='ireallydoit.g', host='zeus');
myexecutive.run('doiteventually()', inc='ireallydoit.g', host='zeus');
myexecutive.stop(2);
print myexecutive.list();
```

---

executive.gui - Function

Package	**utility**
Module	**misc**
Tool	**executive**

Open a GUI for this tool

Synopsis
gui()

Description
Opens a GUI for this tool.

Returns
Bool

Example

```
include 'executive.g';
myexecutive:=executive();
myexecutive.gui();
```

---

executive.test - Function

Package	**utility**
Module	**misc**
Tool	**executive**

Run a test for this tool

Synopsis
test()

Description
Run a test (actually executivetest) for this tool.

Returns
Bool

Example

```
include 'executive.g';
myexecutive:=executive()
myexecutive.test();
```

1.6.33 executivetest - Function

Package utility
Module misc

Perform a test of executive

Synopsis
executivetest()

Description
Perform a test. The commands executed are:

```
t:=executive(2);
t.run('print time();print system;', host, log=spaste(testdir, '/etest1.log'));
t.run('print time();print system;', log=spaste(testdir, '/etest2.log'));
t.run('print time();print system;', host);
t.run('print time();print system;', host, log=spaste(testdir, '/etest4.log'));
t.run('print time();print system;', host, log=spaste(testdir, '/etest5.log'));
t.stop(1);
t.show();
t.stop(4);
t.status();
```

Returns
Bool

Arguments

Example

```
include 'executive.g'
executivetest();
```

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