1.6 misc - Module

Miscellaneous utilities
include "misc.g"

Description
The misc module contains a number of generally useful tools:

`misc` contains a grab-bag of useful functions.

`os` contains system-specific functions like file and directory handling.

`aipsrc` is available for querying the AIPS++ resources (".aipsrc" files). `aipsrcedit` allows interactive editing of AIPS++ resources.

`getrc` is available for querying the AIPS++ resources (".aisprc" files) during system startup to avoid recursive dependencies involving aipsrc.

`sysinfo` allows interrogation of other miscellaneous AIPS++ information.

`printer` is available for printing files and Glish values.

`assay` is available for testing the installation.

`stopwatch` implements a set of timing functions in the style of a traditional stopwatch.

`sh` creates a persistent shell client, amortizing startup over many different function calls.

`substitute` is available for substituting Glish variables and expressions in a string.

`itemcontainer` is a generic closure tool container.

`aips2loc` plots the lines of code in AIPS++ as a function of time.

`ftp` allows ftp operation from inside AIPS++.

`make` provides a facility equivalent to Unix make inside Glish.

`executive` provides a facility for running scripts in a queue.

Tools

aipsrc	tool to find aips resources from .aipsrc files
aipsrcedit	Tool to edit aipsrc file
assay	Assay test and demo functions
executive	Tool for running scripts in a queue
ftp	Tool to ftp files
getrc	find aips recources from .aipsrc files using the getrc utility

itemcontainer	A generic closure toolcontainer
misc	Misc tool
os	Functions to perform system-specific operations
plugins	attach code to tools at runtime
printer	Print files
sh	A persistent shell client
stopwatch	stopwatch style timer functions
substitute	Functions to substitute Glish variables and expressions
sysinfo	Miscellaneous information
timer	Execute functions periodically, wait for a specified time

Functions

about	Information about AIPS++
aips2loc	Plot lines of code in AIPS++
ask	Displays the AIPS++ question submission form
bug	Provides a GUI to for user's to report problems.
edit	Edit and optionally execute a file
executivetest	Perform a test of executive
faq	Drives the user's web browser to the AIPS++FAQ
ftptest	Test the ftp tool
help	Provides user help.
is_itemcontainer	Is this an itemcontainer tool?
make	Function for a production system
note	Send a message to the logger or terminal
register	Provides a GUI to for user's to register AIPS++.
serverexists	Does the specified server exist
throw	Send a SEVERE message, then fail
web	Drives a web browser to the last requested help

1.6.1 misc - Tool
Package utility
Module misc

Misc tool

Description
The misc tool contains a number of string, file, and miscellaneous functions.
NOTE Some (limited) functions emulating C-based input and output routines are available. These have been replaced by functions native to Glish. We strongly recommend that you use the Glish functions instead.

Some of these functions are implemented completely in Glish without using the gmisc client. They remain in this tool for now. These functions are indicated in the individual descriptions.
This uses a server process misc which is started when first needed and then automatically restarted if it fails (hopefully never!).

Constructors

misc	Construct a misc tool

Functions

dir	Return the directory for a given directory name
escapespecial	escape characters in a string special to the shell
fclose	close fileid
fgets	interface to fgets
fields	list fields in a record or tool
fileexists	Does this file exist?
filetype	returns the UNIX type of file
fopen	interface to c fopen
fprintf	C-style formatted output
initspinner	start an ascii activity spinner
killspinner	kill an ascii activity spinner
listfields	Hierarchically list fields in a record
parentdir	Return the parent directory
patternmatch	Find matching strings using shell expressions
readfile	reads a file into a buffer
shellcmd	Execute a shell command
stripleadingblanks	Strip leading blanks from a string
striptrailingblanks	Strip trailing blanks from a string
thisdir	Return the name of this directory
tovector	Convert variable to a vector
type	Returns tool type; "misc"

misc - Constructor

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

Construct a misc tool

Synopsis
misc()

A misc tool is created by the global constructor function misc. A default misc tool called defaultmisc is created automatically on inclusion of misc.g. A convenient shorthand name for defaultmisc is dms. Normally the defaultmisc tool is all that you'll need.

---

misc.type - Function

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

Returns tool type; "misc"

Synopsis
type()

Description
Returns the type of tool, in this case, "misc".

Returns
String

Example

```
- dms.type()
misc
```

---

misc.shellcmd - Function

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

Execute a shell command

Synopsis
shellcmd(command, log)

Description
Execute a shell command, logging results optionally. The Glish function `shell()` can also be used directly but it does not split lines or log. This function does not use the gmisc client, just the Glish shell client.

Arguments

command	Command string Allowed:String
log	Log the output? Allowed:Bool Default: T

Example

```
- dms.shellcmd('ls -l', log=F)
total 54
lrwxrwxrwx   1 root     bin           31 Jul 29 16:45 RCS -> /aips++/daily/rcs/trial/
drwxr-xr-x   2 tcornwel staff        512 Aug 29 15:11 aips++.log.table
-r--r--r--   1 tcornwel staff        256 Aug  1 21:31 makefile
-rw-r--r--   1 tcornwel staff       1556 Aug 29 09:28 ms.cc
-r--r--r--   1 tcornwel staff       7275 Aug 29 09:29 ms.g
-rw-r--r--   1 tcornwel staff       8889 Aug 29 09:28 ms.help
-rw-r--r--   1 tcornwel staff      22235 Aug 29 09:28 msImpl.cc
-rw-r--r--   1 tcornwel staff       4200 Aug 29 09:28 msImpl.h
-rw-r--r--   1 tcornwel staff        269 Aug  1 21:41 templates
drwxrwsr-x   2 tcornwel staff        512 Aug  1 21:41 tmplinst
T
- shell('ls -l')
total 54 lrwxrwxrwx   1 root     bin           31 Jul 29 16:45 RCS -> /aips++/daily/r

cornwel staff        1556 Aug 29 09:28 ms.cc -r--r--r--   1 tcornwel staff        7275

l staff        4200 Aug 29 09:28 msImpl.h -rw-r--r--   1 tcornwel staff        3059 Ju
```

Returns
Bool

---

misc.escapespecial - Function

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

escape characters in a string special to the shell

Synopsis
escapespecial(string)

Description
This simply replaces the characters `*`, `#`, `(`, `)`, and `|`, by the sequences `\*` – `\|`, respectively. This is mainly used on strings that are to be passed to the shell. In a compromise between speed and completeness, this is only a subset of all possible significant characters, but it covers the most commonly problematic cases. There is no escape mechanism to avoid the filtering. This uses the builtin regex substitution capabilities of Glish.

Arguments

string	String Allowed:String	

Returns
String

Example

```
- dms.escapespecial('abc def ghi jkl mno')
abc def ghi jkl mno
- dms.escapespecial('a#c d*f g(i j)l m|o')
a\#c d\*f g\(i j\)l m\|o
-
+ files := shell('ls -1 A*foo')
- files
A*foo Anotherfoo
-
+ for (f in files) { print shell(spaste('ls -l ',f)) }
-rw-rw-r--   1 dbriggs  visitor        5 Dec 19 18:02 A*foo
-rw-rw-r--   1 dbriggs  visitor        4 Dec 19 18:16 Anotherfoo
-rw-rw-r--   1 dbriggs  visitor        4 Dec 19 18:16 Anotherfoo
-
+ for (f in files) { print shell(spaste('ls -l ',dms.escapespecial(f))) }
-rw-rw-r--   1 dbriggs  visitor        5 Dec 19 18:02 A*foo
-rw-rw-r--   1 dbriggs  visitor        4 Dec 19 18:16 Anotherfoo
```

The * in the filename, which we don't know about a priori, has confused the shell. This is cured simply by wrapping all unknown strings in dms.escapespecial() before passing them to the shell.

---

misc.striptrailingblanks - Function

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

Strip trailing blanks from a string

Synopsis
striptrailingblanks(string)

Description
This uses the builtin regex substitution and not the gmisc client.

Arguments

string	String to be stripped Allowed:String

Returns
String

Example

```
- dms.striptrailingblanks("foo         ")
foo
```

---

misc.stripleadingblanks - Function

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

Strip leading blanks from a string

Synopsis
stripleadingblanks(string)

Description
This uses the builtin regex substitution and not the gmisc client.

Arguments

string	String to be stripped Allowed:String

Returns
String

Example

```
- dms.stripleadingblanks("      foo")
foo
```

---

misc.patternmatch - Function

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

Find matching strings using shell expressions

Synopsis
patternmatch(pattern, strings)

Description
This pattern matching function uses the shell expression syntax familiar to anyone who has used a UNIX shell. In brief:

***** Matches any string, including the null string.

? Matches any single character.

Note that `[...]` matches any one of the enclosed characters. A pair of characters separated by - matches any character lexically between the pair, inclusive. If the first character following the opening `[` is a !, any character not enclosed is matched.
The following characters have a special meaning to the shell and cause termination of a word unless quoted:

`;  &  (  )  |  ^  <  >  newline  space  tab`

A character may be quoted (that is, made to stand for itself) by preceding it with a backslash (\) or inserting it between a pair of quote marks (" or ""). For further definition of the syntax, see the UNIX man page for sh. This is a simple syntax that suffices for many applications. If a more capable matching syntax is needed, use the builtin regex capabilities of Glish.
This is implemented entirely in Glish and not the gmisc client.

Arguments

pattern	Pattern Allowed:Any shell expression
strings	Strings to be tested Allowed:String

Example

```
- dms.patternmatch ('12**', "12345 89712 123 12 11")
12345 123 12
```

---

misc.fields - Function

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

list fields in a record or tool

Synopsis
fields(rec)

Description

This function is similar to the glish function field_names, except that it lists the names of fields in a human-readable format.

Arguments

rec	record name Allowed:record

Example

```
- pg := pgplotter()
- dms.fields(pg)
activatecallback    hist                qcol                settings            addredrawfunctio
```

---

misc.fileexists - Function

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

Does this file exist?

Synopsis

fileexists(file, opt)

Description

This provides a simple interface to the UNIX test command for testing for the existence of files. See the man page for test for a definition of the options that can be used to test for particular file types. The default test is to see if a file exists and has a size greater than zero.

Arguments

file	file name Allowed:String
opt	option for UNIX test command Allowed:String Default: -s

Example

```
- dms.fileexists('foo')
F
- dms.fileexists('.')
T
- dms.fileexists('.', '-f')
F
```

---

misc.dir - Function

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

Return the directory for a given directory name

Synopsis

dir(directoryname)

Description

This returns a set of strings containing the files in the specified directory. Note that the directory name '.' is always returned.

Arguments

directoryname	Directory name Allowed:String

Example

```
- dms.dir('.')
. tmplinst ms.help makefile RCS templates ms.cc ms.g msImpl.cc msImpl.h aips++.log.tab
- dms.dir('RCS')
RCS UVFitsWriter.h,v makefile,v ms.cc,v ms.g,v ms.help,v msImpl.cc,v templates,v msImpl
```

---

misc.parentdir - Function

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

Return the parent directory

Synopsis
parentdir(directoryname)

Description
This returns the name of the parent directory.

Arguments

directoryname	Directory name Allowed:String

Example

```
- dms.parentdir('/home/tarzan/tcornwel')
/home/tarzan
- dms.parentdir('/home/tarzan/tcornwel/aips++')
/home/tarzan/tcornwel
- dms.parentdir('.')         # If current directory is /home/tarzan/tcornwel
/home/tarzan
```

---

misc.thisdir - Function

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

Return the name of this directory

Synopsis
thisdir(directoryname)

Description
This returns the full name of this directory.

Arguments

directoryname	Directory name Allowed:String

Example

```
- dms.thisdir('.')
/home/tarzan/tcornwel
```

---

misc.filetype - Function

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

returns the UNIX type of file

Synopsis
filetype(filename)

Arguments

filename	File name Allowed:String

Example

```
- dms.filetype('.')
Directory
- dms.filetype('ms.cc')
Regular File
```

---

misc.fopen - Function

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

interface to c fopen

Synopsis
fopen(filename, mode)

Description
This function and those following emulate the low-level input/output functions available in C. We expect to replace these in the near future.
This function opens a file, returning a file id (an integer) that must be used to close the file.

Arguments

filename	File name Allowed:String
mode	UNIX mode of access Allowed:Usually r, w, or a

Returns
fileid

Example

```
- bugReport := spaste('/tmp/bugreport.',emailID)
- fp := dms.fopen(bugReport, 'w')
- dms.fclose(fp)
T
```

In the example, a filename found in the string variable bugReport is opened for writing.

---

misc.fclose - Function

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

close fileid

Synopsis
fclose(fileid)

Arguments

fileid	File id Allowed:file id as returned by fopen

Returns
fileid

Example

```
- fp := dms.fopen('myfile', 'w')
# Do some stuff here
- dms.fclose(fp)
T
```

---

misc.fprintf - Function

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

C-style formatted output

Synopsis
fprintf(fileid, format, ...)

Description
This and the following routines provide input and output formatting capabilities similar to those provided in C libraries. See the UNIX man page on *e.g.* fprintf for more information. Differences from the C routines are described.
The ellipsis ... denotes that any number of arguments may be provided.

Arguments

fileid	File id Allowed:file id as returned by fopen
format	c-style format description Allowed:String
...	Additional arguments Allowed:Strings, floats, ints, etc...

Example

```
- fp := dms.fopen('myfile', 'w')
- sourcename := 'mySource'
- hour := 12
- minute := 4
- seconds := 42.3
- dms.fprintf(fp, '%s %02d:%02d:%05.2f\n', sourcename, hour, minute, seconds)
- dms.fclose(fp)
T
```

In this example you'd get the following in myfile
mySource 12:04:42.30

---

misc.fgets - Function

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

interface to fgets

Synopsis
fgets(fileid, filler)

Description
Get a string buffer from a file. fgets is similar to the C function fgets but different in a couple of key ways. First it reads up to 500 characters or until a new line is reached in the file. Unfortunately 500 is hard-coded (the current implementation requires this, you don't want to know). Second, if you supply a fill character it will replace blank characters with your fill character.

Arguments

fileid	File id Allowed:file id as returned by fopen
filler	string buffer to replace blanks Allowed:String Default: ”

Returns
Contents of current line in the file

Example

```
- fp := dms.fopen('myfile', 'r')
- a := dms.fgets(fp)
- print a
mySource 12:04:42.30
```

dms.fgets(fp) will return 0 when it reaches the end of file.

---

misc.readfile - Function

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

reads a file into a buffer

Synopsis
readfile(file)

Description
The file is read into a buffer. This is useful for reading binary files, for example.

Arguments

file	File name Allowed:String

Example

```
- buff:=dms.readfile("/export/home/estella/AIPS/DATA/CAD000000.0B4;")
```

---

misc.fopen - Function

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

interface to c fopen

Synopsis
fopen(filename, mode)

Description
This function and those following emulate the low-level input/output functions available in C. We expect to replace these in the near future.
This function opens a file, returning a file id (an integer) that must be used to close the file.

Arguments

filename	File name Allowed:String
mode	UNIX mode of access Allowed:Usually r, w, or a

Returns
fileid

Example

```
- bugReport := spaste('/tmp/bugreport.',emailID)
- fp := dms.fopen(bugReport, 'w')
- dms.fclose(fp)
T
```

In the example, a filename found in the string variable bugReport is opened for writing.

---

misc.initspinner - Function

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

start an ascii activity spinner

Synopsis
initspinner(interval)

Description
This initializes a traditional ascii activity spinner. The characters `|`, `/`, `-`, and `\` are printed periodically, followed by a backspace. The printing is done outside the standard Glish output mechanism, resulting in a spinning bar. If any Glish output is done during this time, orphaned characters will be left over as the cursor moves. In particular, this will probably confuse Glish's notion of where the cursor is during editing. It is probably best not to leave this active while the user is expected to type anything. Use killspinner to shut this down. As there should only be one of these things active at once, there can only be one spinner per misc tool, and the spinner argument is implicit.

Arguments

interval	Interval in seconds between characters Allowed:Real

---

misc.killspinner - Function

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

kill an ascii activity spinner

Synopsis

killspinner()

Description
This shuts down the spinner as started by initspinner.

---

misc.listfields - Function

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

Hierarchically list fields in a record

Synopsis
listfields(rec, listdata, depth)

Arguments

rec	Record to list Allowed:Glish record
listdata	List data at bottom of record Allowed:Boolean Default: T
depth	Depth in record Allowed:Integer Default: 0

Returns
T

Description
This function hiearchically lists the contents of a record by drawing a tree and optionally printing the data at the end of each branch.
If you choose not to list the data at the end of the branch, then its type will be printed for you instead.
The `depth` argument is just used to offset the printed output as it works its way down the record. If you put some other value, other than 0, then the text will be offset to the right differently. You shouldn't need to use it.

Example

```
r := [=]
r.x := [=]
r.y := [=];
r.x.a := [1,2,3]
r.x.b := 'trout'
r.y.a := [10,20,30]
r.y.b := 'salmon'
r.y.c := [=]
r.y.c.i := 1
r.y.c.k := [=]
r.y.c.k.x := [1,2,3,4]
r.y.c.k.y := -400
r.y.c.j := 1
#
dms.listfields(r)
  x
    a = [1 2 3]
    b = trout
  y
    a = [10 20 30]
    b = salmon
    c
      i = 1
      k
        x = [1 2 3 4]
        y = -400
      j = 1
#
dms.listfields(r, listdata=F)
  x
    a = integer [3]
    b = string
  y
    a = integer [3]
    b = string
    c
      i = integer
      k
        x = integer [4]
        y = integer
      j = integer
```

---

misc.tovector - Function

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

Convert variable to a vector

Synopsis
tovector(thing, type, unsetvalue)

Arguments

thing	Variable to convert Allowed:Any Glish variable
type	Specify the output type of the vector Allowed:String chosen from 'string', 'float', 'double', 'integer' Default: 'string'
unsetvalue	Unset value for unset items in records Allowed:Numeric value Default: unset

Returns
Vector of given type or fail

Description
This function converts a Glish variable to a vector of the given type. It is a bit specialized, and ultimately this functionality should be moved to a central parsing location like entryparser.g
Its purpose is so that a vector variable can be specified as a vector, a string, or a vector of strings. Thus no matter how the vector is entered to Glish, the right thing comes back. It is really meant for numeric and string vectors. If you put in what you want out (e.g. input a float vector and ask for the output to be a float vector) that is ok. If the input is a string, you can use space or comma or both delimiters.
After attempting to convert the input (argument `thing`) to a vector, it is type converted to the specified type.
The `unsetvalue` argument is only relevant if the input `thing` is a record. Then, any fields in the record that are unset, are replaced by `unsetvalue`. If `unsetvalue=unset` and a record field is unset, then an exception will be thrown. The record fields are then converted to vector of the appropriate type.

Example

```
- dms.tovector("1 2.2 3", 'double')     # Vector of strings
[1 2.2 3]
- dms.tovector('1 2.2 3', 'double')     # String (space delimiters)
[1 2.2 3]
- dms.tovector('1,  2.2,  3', 'double') # String (space and comma delimiters)
[1 2.2 3]
- dms.tovector([1,2.2,3], 'float')      # Numeric vector
[1 2.2 3]
- dms.tovector([1,2.2,3], 'integer')    # Float -> integer
- r := [=]
- r.a := '1'
- r.b := unset                          # Unset field
- r.c := 2
- dms.tovector(r,'float', 20)           # Record -> float
[1 20 2]
```

1.6.2 os - Tool
Package utility
Module misc

Functions to perform system-specific operations
include "os.g"

Constructors

Functions

basename	get file part in path names
copy	copy a file, directory, or symlink
dir	return a vector of strings with directory listing
dirname	get directory part in path names
done	end the os tool
edit	edit a file
fileexists	test if files exists
filetime	return the file time
filetimestring	return the file time string
filetype	determine file type
freespace	determine freespace on file systems

fullname	expand path names
isvalidpathname	test if path names are valid
lockinfo	get info about table use and locking
ls	directory listing
mail	mail a message to a user
mkdir	create directories
move	move a file, directory, or symlink
remove	remove files, directories, or symlinks
showtableuse	tell if a table is used or locked
size	determine file or directory sizes
type	Return the type of this tool

Description

The os tool is used to perform system-specific operations, in particular operations on files — for example:

- testing if a file exists
- getting a directory listing
- getting the size of file(s)
- getting the free space on a device
- dealing with path names
- copying, moving, or removing a file or directory

Most functions have a switch telling if a symbolic link has to be followed. When a path name is given, it can contain environment variables (e.g. `$HOME`) and/or tildes (e.g. ~ or ~`user`).
Note that glish has some built-in functions to do file IO. It also has the `stat` function to obtain file information. However, those functions do not handle environment variables and tilde expansion.

Example

```
include 'os.g'
dos.fileexists ('~/file.name');
dos.freespace ('.');
```

os.isvalidpathname - Function

Package	**utility**
Module	**misc**
Tool	**os**

test if path names are valid

Synopsis
isvalidpathname(pathname)

Description
This function tests if the given path names are valid. It tests if files with the given names exists or can be created.

Arguments

pathname	in	path names Allowed:vector of strings

Returns
vector of bools

---

os.fileexists - Function

Package	**utility**
Module	**misc**
Tool	**os**

test if files exists

Synopsis
fileexists(file, follow)

Description
This function tests if the given files exists. If `follow==T` symbolic links will be followed.

Arguments

file	in	file names Allowed:vector of strings
follow	in	follow symlink? Allowed:bool Default: T

Returns
vector of bools

---

os.dir - Function

Package	**utility**
Module	**misc**
Tool	**os**

return a vector of strings with directory listing

Synopsis
dir(directoryname, pattern, types, all, follow)

Description
Get a vector of strings containing the names of files in the given directory. Other arguments can be given to select specific files. To simply list the files in a human-readable format, use dos.ls() instead.

`pattern` can be given to select specific file names using a file name pattern as given in a shell. E.g. `pattern='*.g'` to get all .g files.
By default no name selection is done.

`types` can be given to select on a file (access) types. It is a string containing zero or more letters indicating the required type(s). There are two groups of types:

- `r d s` indicate regular file, directory, resp. symbolic link.
- `R W X` indicate readable, writable, resp. executable.

If one or more letters from a group are given, only such files will be selected.
By default no type selection is done.

`all` can be given to indicate that file names starting with a . should be selected.
By default they are not selected.
Note that even if a pattern `'.*'` is given, `all=T` should be given to get those files.

If `follow==T` symbolic links will be followed. In that case using 's' in the `types` argument makes no sense.

Arguments

directoryname	in	directory name Allowed:string Default: '.'
pattern	in	file name pattern Allowed:string Default: ''
types	in	which types? Allowed:string Default: ''
all	in	also dot files? Allowed:bool Default: F
follow	in	follow symlink? Allowed:bool Default: T

Example

```
dos.dir (pattern='*.pg')   #get all files ending in .pg
dos.dir (types='rX')       #get all regular executable files
v:=dos.dir ('~/aips++/test', types='d')    # get directories
for (nm in v) {                            # (potential tables)
  if (dos.filetype(spaste('~/aips++/test/',nm)) == 'Table') {
    print nm;                                         #show tables
  }
}
```

Returns

vector of strings or fail

---

os.ls - Function

Package	**utility**
Module	**misc**
Tool	**os**

directory listing

Synopsis

ls(dir)

Description
This function lists directories in a human-readable format, like the unix command ls. To retrieve a glish record containing files in a directory, use dos.dir() instead.

Arguments

dir	in	directory name Allowed:string Default: current directory

Example

```
- dos.ls()
aips++.inputsv2.table mymsgs                tkeep3.g
aips++.records.table  tkeep.g
junk_SP               tkeep2.g
T
- dos.ls('/')
aips++     dev        home       lost+found opt        sbin       users
bin        etc        initrd     misc       proc       tftpboot   usr
boot       export     lib        mnt        root       tmp        var
T
-
```

Returns
T or F

---

os.mkdir - Function

Package	**utility**
Module	**misc**
Tool	**os**

create directories

Synopsis

mkdir(directoryname, makeparent)

Description
Create directories. An error is given if a directory already exists. By default parent directories are not created if they don't exist yet.

Arguments

directoryname	in	directory names Allowed:vector of strings
makeparent	in	make parent directories if needed Allowed:bool Default: F

Returns
T or fail

---

os.fullname - Function

Package	**utility**
Module	**misc**
Tool	**os**

expand path names

Synopsis
fullname(pathname)

Description
Expand path names to their full absolute names by substituting environment variables (e.g. `$HOME`), applying tilde expansion, and prefixing a relative name with the working directory. It also takes care that parts . and .. are removed and that multiple slashes are replaced by a single slash. A possible trailing slash is also removed.
Note that it does not substitute symbolic links.

Example

```
  dos.fullname ('~mhaller')
/home/rzmws0/mhaller
  dos.fullname ('~mhaller//./../')
/home/rzmws0
```

Arguments

pathname	in	path name Allowed:vector of strings Default: '.'

Returns
vector of strings

---

os.dirname - Function

Package	**utility**
Module	**misc**
Tool	**os**

get directory part in path names

Synopsis
dirname(pathname)

Description
After the path names are expanded to their full names (see above), it returns the directory part of each full name.

Example

```
  dos.dirname ('~mhaller')
/home/rzmws0
```

Arguments

pathname	in	path name Allowed:vector of strings Default: '.'

Returns
vector of strings

---

os.basename - Function

Package	**utility**
Module	**misc**
Tool	**os**

get file part in path names

Synopsis
basename(pathname)

Description
After the path names are expanded to their full names (see above), it returns the file part of each full name.

Example

```
  dos.basename ('~mhaller')
mhaller
```

Arguments

pathname	in	path name Allowed:vector of strings Default: '.'

Returns
vector of strings

---

os.filetype - Function

Package	**utility**
Module	**misc**
Tool	**os**

determine file type

Synopsis
filetype(filename, follow)

Description
Determine the file type of each file. It also recognizes tables.
By default it follows a symbolic link.
The string returned can have the following values:

Table

Directory

Regular File

SymLink

Invalid (meaning that the file does not exist)

Unknown

Note that in principle the type 'SymLink' can only be returned when `follow=F`.

Example

```
  dos.filetype ('~/aips++/test/hol.ms')
Table
```

Arguments

filename	in	file name Allowed:vector of strings
follow	in	follow symlink? Allowed:bool Default: T

Returns
vector of strings

---

os.filetime - Function

Package	**utility**
Module	**misc**
Tool	**os**

return the file time

Synopsis
filetime(pathname, whichtime, follow)

Description
Return for each file the time the file was last accessed (1), modified (2), or its status changed (3). Default is 2 (time modified).
The time returned is the Modified Julian Date (as UTC).
By default it follows a symbolic link.

Example

```
  dos.filetime ('~/aips++/test/hol.ms')
# 51638.4
```

Arguments

pathname	in	path name Allowed:vector of strings Default: '.'
whichtime	in	which time to return Allowed:integer Default: 2
follow	in	follow symlink? Allowed:bool Default: T

Returns
vector of doubles or fail

---

os.filetimestring - Function

Package	**utility**
Module	**misc**
Tool	**os**

return the file time string

Synopsis
filetimestring(pathname, whichtime, follow)

Description
Return for each file the time the file was last accessed (1), modified (2), or its status changed (3). Default is 2 (time modified).
The time returned is the local time.
By default it follows a symbolic link.

Example

```
  dos.filetimestring ('~/aips++/test/hol.ms')
# 04-Apr-2000/12:43:27
```

Arguments

pathname	in	path name Allowed:vector of strings Default: '.'
whichtime	in	which time to return Allowed:integer Default: 2
follow	in	follow symlink? Allowed:bool Default: T

Returns
vector of doubles or fail

---

os.size - Function

Package	**utility**
Module	**misc**
Tool	**os**

determine file or directory sizes

Synopsis
size(pathname, follow)

Description
Determine the file sizes in bytes. When a file is a directory, the size is the sum of the sizes of all files in the directory, including all files in subdirectories. It is similar to the UNIX command `du -s`.
By default it follows a symbolic link.

Example

```
  dos.size ('~/aips++/test/hol.ms')
# 2.59398e+07
```

Arguments

pathname	in	path name Allowed:vector of strings Default: '.'
follow	in	follow symlink? Allowed:bool Default: T

Returns
vector of doubles or fail

---

os.freespace - Function

Package	**utility**
Module	**misc**
Tool	**os**

determine freespace on file systems

Synopsis
freespace(pathname, follow)

Description
Determine the free space (in bytes) on the file systems containing the given files or directories.
By default it follows a symbolic link.

Example

```
  dos.freespace ('~/aips++/test/hol.ms')
# 1.09639e+09
```

Arguments

pathname	in	path name Allowed:vector of strings Default: '.'
follow	in	follow symlink? Allowed:bool Default: T

Returns
vector of doubles or fail

---

os.copy - Function

Package	**utility**
Module	**misc**
Tool	**os**

copy a file, directory, or symlink

Synopsis
copy(source, target, overwrite, follow)

Description
Copy a file, directory (recursively), or symbolic link.
If the source is a directory, the entire directory (including all subdirectories) is copied (similar to UNIX command `cp -r`). However, unlike that UNIX command the source directory itself is not copied. Only its contents are copied.
If the source is a file or symlink and the target is a directory, the source is copied to that directory (similar to UNIX command `cp`).
By default a source symbolic link is followed. In that case the symbolic link is not copied, but the file or directory it links to.
An error is given if the target already exists, unless `overwrite=T`. In that case the target will be removed first.

Example
This command copies the entire MeasurementSet to /aips++2/gvandiep. Note that the name of the MeasurementSet has to be given in the target.

```
dos.copy ('~/aips++/test/hol.ms', '/aips++2/gvandiep/hol.ms')
```

Arguments

source	in	path name of source Allowed:string
target	in	path name of target Allowed:string
overwrite	in	overwrite if target exists? Allowed:bool Default: F
follow	in	follow symlink? Allowed:bool Default: T

Returns
T or fail

---

os.move - Function

Package	**utility**
Module	**misc**
Tool	**os**

move a file, directory, or symlink

Synopsis
move(source, target, overwrite, follow)

Description
Move a file, directory (recursively), or symbolic link. It can also move across file systems.
If the source is a directory, the entire directory (including all subdirectories) is moved (similar to UNIX command `mv`). However, unlike that UNIX command the source directory itself is not moved. Only its contents are moved.
If the source is a file or symlink and the target is a directory, the source is moved to that directory (similar to UNIX command `mv`).

By default a source symbolic link is followed. In that case the symbolic link is not moved, but the file or directory it links to.
An error is given if the target already exists, unless `overwrite=T`. In that case the target will be removed first.

Example
This command moves the entire MeasurementSet to /aips++2/gvandiep. Note that the name of the MeasurementSet has to be given in the target.

```
dos.move ('/aips++2/gvandiep/hol.ms', '~/aips++/test/hol.ms')
```

Arguments

source	in	path name of source Allowed:string
target	in	path name of target Allowed:string
overwrite	in	overwrite if target exists? Allowed:bool Default: F
follow	in	follow symlink? Allowed:bool Default: T

Returns
T or fail

---

os.remove - Function

Package	**utility**
Module	**misc**
Tool	**os**

remove files, directories, or symlinks

Synopsis
remove(pathname, recursive, mustexist, follow)

Description
Remove one or more files, directories (optionally recursively), or symbolic links.

If the source is a directory, it will be by default be removed recursively. If `recursive=F` only the directory will be removed. In that case it results in an error if it is not empty.
By default a source symbolic link is followed. In that case not only the symbolic link is removed, but also the file or directory it links to.
By default an error is given if the target to be deleted does not exist. This can be prevented by giving `mustexist=F`.

Example
This command removes an entire MeasurementSet.

```
dos.remove ('~/aips++/test/hol.ms');
```

Arguments

pathname	in	path names Allowed:vector of strings
recursive	in	remove directories recursively? Allowed:bool Default: T
mustexist	in	does the file need to exist? Allowed:bool Default: T
follow	in	follow symlink? Allowed:bool Default: T

Returns
T or fail

---

os.lockinfo - Function

Package	**utility**
Module	**misc**
Tool	**os**

get info about table use and locking

Synopsis
lockinfo(tablename)

Description
Get info about use and locking of a table. If so, give the pid of the process using or locking it. The function `showtableuse` uses this function to print a message about the use and locking of a table.
It returns a vector of 3 integer elements. The first element tells if the table is used or locked.
0 = not used
1 = used, but not locked
2 = read-locked
3 = write-locked
The second element gives the pid of the process using or locking the table.
The third element is 1 if the lock is permanent (otherwise 0).
Note that multiple processes can use a table or hold a read-lock. This function gives the pid of only one of those processes.

Example
Note that this example is done better by function `showtableuse`.

```
res := dos.lockinfo ('~/aips++/test/hol.ms');
if (res[1] != 0) {
  print (paste ('The table is used (and maybe locked) by process', res[2]));
}
```

Arguments

tablename	in	table name Allowed:string

Returns
vector of 2 integers or fail

---

os.showtableuse - Function

Package	**utility**
Module	**misc**
Tool	**os**

tell if a table is used or locked

Synopsis
showtableuse(tablename)

Description
Tell if a table is used or locked in another process. A message is sent to the logger telling if the table is used or locked and by which process.
Note that multiple processes can use a table or hold a read-lock. This function gives the pid of only one of those processes.

Example

```
  dos.showtableuse ('~/aips++/test/hol.ms');
# NORMAL: Table ~/aips++/test/hol.ms is opened
# (but not locked) in process 5242
```

Arguments

tablename	in	table name Allowed:string

Returns
T or F

---

os.mail - Function

Package	**utility**
Module	**misc**
Tool	**os**

mail a message to a user

Synopsis
mail(message, recipient, sender, subject, cc, bcc)

Description
Mail a message to one or more recipients. It is also possible to give zero or more cc- and bcc-addresses. Note that `\n` can be used to create multiple lines in the message. If the sender is given as a boolean, it is assumed that the message contains lines containing the sender, subject, etc.. Otherwise the message is prepended with such lines.
The mailer to be used can be defined in the aipsrc variable `os.mailer`. It defaults to `mailx`.

The function is used by bug and ask. Currently it uses a shell command to do the mailing.

Example
The following command sends a message with noordam as the sender.

```
dos.mail ('line1\nline2', 'gvd mhaller', 'noordam', 'some subject',
          cc='tcornwel@nrao.edu wyoung@nrao.edu')
```

Arguments

message	in	message to send Allowed:string
recipient	in	whom to send the message Allowed:string
sender	in	whom the message comes from Allowed:string Default: T
subject	in	subject of the email Allowed:string Default: ”
cc	in	whom to cc the message Allowed:string Default: ”
bcc	in	whom to bcc (blind copy) the message Allowed:string Default: ”

Returns
T or fail

---

os.edit - Function

Package	**utility**
Module	**misc**
Tool	**os**

edit a file

Synopsis
edit(file, editor, async)

Description
This command starts a shell to edit a file. It can only be used in a GUI environment. An editor like vi should not be used, because it does not start up in a separate window. The shell may be synchronous or asynchronous (default). Use the synchronous option if the file is needed immediately.
The default editor is emacs.

Example

```
dos.edit ('file.cc');
```

Arguments

file	in	file to be edited Allowed:string
editor	in	editor to be used Allowed:string Default: emacs
async	in	Run asynchronously Allowed:boolean Default: T

Returns
T or fail

---

os.type - Function

Package	**utility**
Module	**misc**
Tool	**os**

Return the type of this tool

Synopsis
type()

Returns
string

Description
This tool function returns the string “os”. If you have a tool, but don’t know what kind it is, this function usually exists to tell you what kind of tool you have.

Example

```
  myos := os()
  print 'type=', myos.type()
# type=os
```

---

os.done - Function

Package	**utility**
Module	**misc**
Tool	**os**

end the os tool

Synopsis
done()

Returns
bool

1.6.3 aipsrc - Tool

Package	**utility**
Module	**misc**

tool to find aips resources from .aipsrc files
include ”aipsrc.g”

Constructors

aipsrc	Construct aipsrc tool

Functions

aipsarch	The AIPSARCH directory string
aipshome	The user's aips++ directory string
aipshost	The AIPSHOST directory string
aipsroot	The AIPSROOT directory string
aipssite	The AIPSSITE directory string
done	free resources used by tool
find	lookup a resource definition
findbool	lookup a boolean resource definition
findfloat	lookup a float value resource definition
findint	lookup a float value resource definition
findlist	lookup a value in a code list
id	identification of tool
init	re-read all the .aipsrc files
type	type of tool
tzoffset	The local time zone offset

Description

The aipsrc tool is used to interrogate the .aipsrc files (AIPS++resource files) to find values for a given keyword, and to connect with some of the more widely used keywords. More on aipsrc files can be found in the system manual, and in the aipsrcdata description.
Including the 'aipsrc.g' file creates a *defaultaipsrc* tool (also called **drc**.
Only in special cases it is necessary to create your own aipsrc tool.

Example

```
- include "aipsrc.g"
T
- drc.find(where, "help.directory")   # use default server
T
- arc := aipsrc()                     # create your own tool
- arc.find(where, "help.directory")
T
- print where
/aips++/daily/sun4sol_gnu/doc/aips2help
- arc.find(where, "does.not.exist")
F
- arc.tzoffset()
10
- arc.done()                          # delete your own tool
```

aipsrc - Constructor

Package	**utility**
Module	**misc**
Tool	**aipsrc**

Construct aipsrc tool

Synopsis
aipsrc(host, forceneweserver)

Arguments

host	in	host on which to run tool Allowed:valid host name string Default: ''
forceneweserver	in	force the use of a new server Allowed:bool Default: F

Returns
tool

Description
Create a quanta `tool`on the specified host (or by default the host you are running on).

---

aipsrc.find - Function

Package	**utility**
Module	**misc**
Tool	**aipsrc**

lookup a resource definition

Synopsis
find(value, keyword, def, usehome)

Description

find will return *T* if the specified keyword was found in an aipsrc file, *F* otherwise. If a find was made, its value will be returned. If a default value was specified it will be used if the aipsrc keyword was not present.

Arguments

value	The value of the resource (Out) Allowed:String
keyword	The name of the resource to look up (In) Allowed:String
def	The default value to use if no aipsrc resource found Allowed:String Default: F
usehome	Use the users own .aipsrc file? (In) Allowed:Bool Default: T

Returns
Bool – is the resource defined?

---

aipsrc.findbool - Function

Package	**utility**
Module	**misc**
Tool	**aipsrc**

lookup a boolean resource definition

Synopsis
findbool(value, keyword, def)

Description
findbool will return *T* if the specified keyword was found in an aipsrc file, *F* otherwise. If a find was made, its value is returned, otherwise the specified (or default) default value `def` will be used. The test for boolean is based on a first character being one of y Y t T 1 2 3 4 5 6 7 8 9 to return true, all others are interpreted as false.

Arguments

value	The value of the resource (Out) Allowed:Bool	
keyword	The name of the resource to look up (In) Allowed:String	
def	Default to use if keyword not found Allowed:Bool Default: T	

Returns
Bool – is the resource defined?

---

aipsrc.findfloat - Function

Package	**utility**
Module	**misc**
Tool	**aipsrc**

lookup a float value resource definition

Synopsis
findfloat(value, keyword, def, undef, unres)

Description
findfloat will return *T* if the specified keyword was found in an aipsrc file, *F* otherwise. If a find was made, its value is returned, otherwise the specified (or default) default value `def` will be used. The value defined in the keyword is converting according to standard rules, i.e. the field upto a non-conforming character will be returned.
The value can be defined with optional units in the aipsrc file. By specifying in the findint a default unit string that has to be used in the absence of given units, and specifying units the result will be converted to before the value is returned. E.g.

```
drc.findfloat(x, 'test.float' , 5, 'h', 'd')
```

will assume that if the keyword is not present, the value of 5 is returned in x. If the keyword is given, but has no units specified, it will be assumed that the user has given the value in 'h'. The value returned will be converted to 'd' (i.e. twenty-four times smaller). If the user has specified units, the value returned will be converted to these ('d' in this case).

Example
With the following aipsrc keywords:

```
test.float: 5
test.float2: 5:30
test.float3: 5a
```

The following will be the result:

```
- arc:=aipsrc()
- arc.findfloat(x, 'test.float')
T
- x
5
- arc.findfloat(x, 'test.float1')
F
- x
0
- arc.findfloat(x, 'test.float1', 7)
F
- x
7
- arc.findfloat(x, 'test.float', undef='h', unres='d')
T
- x
0.208333333
- arc.findfloat(x, 'test.float2', undef='h', unres='d')
T
- x
0.229166667
- arc.findfloat(x, 'test.float3', undef='h', unres='d')
T
- x
1826.25
- arc.done()            # remove tool
```

Arguments

value	The value of the resource (Out) Allowed:Double
keyword	The name of the resource to look up (In) Allowed:String
def	Default to use if keyword not found Allowed:Numeric Default: 0
undef	Default units to use if none given in aipsrc file Allowed:valid unit string Default: F
unres	The units the result will be converted to before giving the value from aipsrc file Allowed:valid unit string Default: F

Returns
Bool – is the resource defined?

---

aipsrc.findint - Function

Package	**utility**
Module	**misc**
Tool	**aipsrc**

lookup a float value resource definition

Synopsis
findint(value, keyword, def, undef, unres)

Description
findint will return *T* if the specified keyword was found in an aipsrc file, *F* otherwise. If a find was made, its value is returned, otherwise the specified (or default) default value `def` will be used. The value defined in the keyword is read as a float, and converted according to standard rules.
The value can be defined with optional units in the aipsrc file. By specifying in the findint a default unit string that has to be used in the absence of given units, and specifying units the result will be converted to before the value is returned. E.g.

```
drc.findint(x, 'test.int' , 5, 'km', 'm')
```

will assume that if the keyword is not present, the value of 5 is returned in x. If the keyword is given, but has no units specified, it will be assumed that the user has given the value in 'km'. The value returned will be converted to 'm' (i.e. thousand times bigger). If the user has specified units, the value returned will be converted to these ('m' in this case).

Example
With the following aipsrc keywords:

```
test.int: 2.9
test.int2: 2.9hm
```

The following will be the result:

```
- arc:=aipsrc()
- arc.findint(x, 'test.int')
T
- x
2
- arc.findint(x, 'test.int1')
F
- x
0
- arc.findint(x, 'test.int1', 7)
F
- x
7
- arc.findint(x, 'test.int', 7, 'm', 'km')
T
- x
0
- arc.findint(x, 'test.int', 7, 'km', 'm')
T
- x
2900
- arc.findint(x, 'test.int1', 7, 'km', 'm')
F
- x
7
- arc.findint(x, 'test.int2', 7, 'km', 'm')
T
- x
290
- arc.findint(x, 'test.int2', 7, '', '')
T
- x
290
- arc.findint(x, 'test.int2', 7)
```

```
T
- x
2
- arc.done()            # remove tool
```

Arguments

value	The value of the resource (Out) Allowed:Int
keyword	The name of the resource to look up (In) Allowed:String
def	Default value to use if keyword not found Allowed:Numeric Default: 0
undef	Default units to use if none given in aipsrc file Allowed:valid unit string Default: F
unres	The units the result will be converted to before giving the value from aipsrc file Allowed:valid unit string Default: F

Returns
Bool – is the resource defined?

---

aipsrc.findlist - Function

Package	**utility**
Module	**misc**
Tool	**aipsrc**

lookup a value in a code list

Synopsis
findlist(value, keyword, vlist, def)

Description
findlist will return *T* if the specified keyword was found in an aipsrc file and the value conformed to one in the list given, *F* otherwise. If a find was made, its value is compared with the codes given in a list (comparison is case insensitive, and only enough characters have to be in the value to give a

unique match). The position in the list will be returned. If the keyword was not found, or if no match was found, a default value (if given) will be matched and its position returned. If the default is not given, or also does not match the code list, the returned value will be beyond the given list.

Example

With the following aipsrc keywords:

```
test.list: ScR
test.list1: test
```

The following will be the result:

```
- y := "none screen gui"
- drc.findlist(x, 'test.list', y, 'gui')
T
- x
2
- drc.findlist(x, 'test.list1', y, 'gui')
F
- x
3
- drc.findlist(x, 'test.list1', y)
F
- x
4
```

Arguments

value	The value of the resource (Out) Allowed:Int
keyword	The name of the resource to look up (In) Allowed:String
vlist	List of code values that will be recognised Allowed:String array
def	Default value to use if keyword not present, or value does not match code value list Allowed:String Default: ”

Returns

Bool – is the resource defined and has correct code?

---

aipsrc.init - Function

Package	**utility**
Module	**misc**
Tool	**aipsrc**

re-read all the .aipsrc files

Synopsis
init()

---

aipsrc.aipsroot - Function

Package	**utility**
Module	**misc**
Tool	**aipsrc**

The AIPSROOT directory string

Synopsis
aipsroot()

Returns
String

---

aipsrc.aipsarch - Function

Package	**utility**
Module	**misc**
Tool	**aipsrc**

The AIPSARCH directory string

Synopsis
aipsarch()

Returns
String

---

aipsrc.aipssite - Function

Package	**utility**
Module	**misc**
Tool	**aipsrc**

The AIPSSITE directory string

Synopsis
aipssite()

Returns
String

---

aipsrc.aipshost - Function

Package	**utility**
Module	**misc**
Tool	**aipsrc**

The AIPSHOST directory string

Synopsis
aipshost()

Returns
String

---

aipsrc.aipshome - Function

Package	**utility**
Module	**misc**
Tool	**aipsrc**

The user's aips++ directory string

Synopsis
aipshome()

Description
aipshome will return the user's aips++ base directory. This will either be

```
~/aips++
```

or the value of the aipsrc keyword 'user.aipsdir' if it has been defined.

Returns
String

---

aipsrc.tzoffset - Function

Package	**utility**
Module	**misc**
Tool	**aipsrc**

The local time zone offset

Synopsis
tzoffset()

Description
tzoffset will return the local time zone offset in hours (East, i.e. add it to UTC).

Returns
Double – offset in hours

---

aipsrc.id - Function

Package	**utility**
Module	**misc**
Tool	**aipsrc**

identification of tool

Synopsis
id()

Arguments

Returns
object id

Description
id will return the object identification. Its use is meant for the toolmanager.

Example

```
drc.id()
[sequence=1, pid=5765, time=1020744553, host=cetus, agentid=2]
```

---

aipsrc.type - Function

Package	**utility**
Module	**misc**
Tool	**aipsrc**

type of tool

Synopsis
type()

Arguments

Returns
string

Description
type will return the tool name.

Example

```
- drc.type()
aipsrc
```

---

aipsrc.done - Function

Package	**utility**
Module	**misc**
Tool	**aipsrc**

free resources used by tool

Synopsis
done(kill)

Arguments

kill	in	force kill of the default tool Allowed:bool Default: F

Returns
bool or fail

Description

If you have created your own aipsrc `tool` (rather than using the default *drc* one), and you no longer need it, its resources can be destroyed by the `done` function. Note that the default `tool` can only be destriyed by setting the `kill` argument explicitly to T.

Example

```
- arc := aipsrc()
- arc.aipsarch()
/nfs/aips++/weekly/sun4sol_gnu
- arc.done()
T
- arc.aipsarch()                    # the tool was destroyed
warning, operand to .aipsarch is not a record
error, F is not a function value
F
```

1.6.4 aipsrcedit - Tool

Package **utility**
Module **misc**

Tool to edit aipsrc file

Description
The aipsrc file contains information for steering the process of an AIPS++ installation, site dependent variables (like default printer) and the actual running of programs. `aipsrcedit` allows interactive editing of the aipsrc values. Note that the file is written to a file aipsrc in the current directory. The user must then copy it to the desired location (*e.g.* $HOME/.aipsrc). AIPS++ must then be restarted before the changes take effect.
The system generates a `defaultaipsrcedit` called `dae` so the user will normally never have to call the constructor.

Example

```
include 'aipsrcedit.g'
dae.gui();
```

Constructors

aipsrcedit	Construct an aipsrcedit tool

Functions

gui	Use the GUI

aipsrcedit - Constructor

Package	**utility**
Module	**misc**
Tool	**aipsrcedit**

Construct an aipsrcedit tool

Synopsis
aipsrcedit()

Description
This is used to construct an `aipsrcedit` tool. The system generates a `defaultaipsrcedit` called `dae` so the user will normally never have to call the constructor.

---

aipsrcedit.gui - Function

Package	**utility**
Module	**misc**
Tool	**aipsrcedit**

Use the GUI

Synopsis
gui()

Description
This shows the `aipsrcedit` GUI.

Returns
Bool

Example

```
include 'aipsrcedit.g'
dae.gui();
```

1.6.5 getrc - Tool

Package	**utility**
Module	**misc**

find aips recources from .aipsrc files using the getrc utility
include "getrc.g"

Constructors

Functions

find	lookup a resource definition

Description
The getrc tool is used to interrogate the .aipsrc files (AIPS++ resource files) to find values for a given keyword, and to connect with some of the more widely used keywords. More on aipsrc files can be found in the system manual, and in the aipsrcdata description. The getrc tool is used only during startup to avoid various dependencies between a few core include files. Once "aipsrc.g" has been included, getrc should not be used. However, if getrc is used it will detect that aipsrc has been included and that will be used. There is only one getrc tool, which is immediately available upon the inclusion of getrc.g.

Example

```
- include "getrc.g"
T
- getrc.find(where, "help.directory")
T
- print where
/aips++/daily/sun4sol_gnu/doc/aips2help
- getrc.find(where, "does.not.exist")
F
```

getrc.find - Function

Package	**utility**
Module	**misc**
Tool	**getrc**

lookup a resource definition

Synopsis
find(value, keyword, def, usehome)

Description
find will return *T* if the specified keyword was found in an aipsrc file, *F* otherwise. If a find was made, its value will be returned. If a default value was specified it will be used if the aipsrc keyword was not present.

Arguments

value	The value of the resource (Out) Allowed:String
keyword	The name of the resource to look up (In) Allowed:String
def	The default value to use if no aipsrc resource found Allowed:String Default: F
usehome	Use the users own .aipsrc file? (In) Allowed:Bool Default: T

Returns
Bool – is the resource defined?

1.6.6 sysinfo - Tool

Package utility
Module misc

Miscellaneous information
include "sysinfo.g"

Constructors

Functions

arch	Architecture

host	Host we are running on
memory	Memory in MB
numcpu	Number of CPU's on the present computer
root	Root of the directory installation
site	Site of the installation
version	AIPS++ version information

Description
The sysinfo class is used to interrogate the system for miscellaneous "standard" information — for example, the current version of AIPS++, or the number of CPU's on the present machine.
The machine specific information (memory, number of CPU's) is dependent on the system manager having set .aipsrc variables appropriately.

Example

```
- info := sysinfo()
- info.version(major=major,dolog=T)
- if (major > 7)...
```

sysinfo.numcpu - Function

Package	**utility**
Module	**misc**
Tool	**sysinfo**

Number of CPU's on the present computer

Synopsis
numcpu()

Returns
integer

---

sysinfo.memory - Function

Package	**utility**
Module	**misc**
Tool	**sysinfo**

Memory in MB

Synopsis
memory()

Returns
integer

---

sysinfo.root - Function

Package	**utility**
Module	**misc**
Tool	**sysinfo**

Root of the directory installation

Synopsis
root()

Returns
string

---

sysinfo.arch - Function

Package	**utility**
Module	**misc**
Tool	**sysinfo**

Architecture

Synopsis
arch()

Returns
string

---

sysinfo.site - Function

Package	**utility**
Module	**misc**
Tool	**sysinfo**

Site of the installation

Synopsis
site()

Returns
string

---

sysinfo.host - Function

Package	**utility**
Module	**misc**
Tool	**sysinfo**

Host we are running on

Synopsis
host()

Returns
string

---

sysinfo.version - Function

Package	**utility**
Module	**misc**
Tool	**sysinfo**

AIPS++ version information

Synopsis
version(major, minor, patch, date, info, formatted, dolog)

Arguments

major	major version number (Out) Allowed:int Default: F
minor	minor version number (Out) Allowed:int Default: F
patch	patch number(Out) Allowed:int Default: F
date	date of the release (Out) Allowed:string Default: F
info	information about the release (Out) Allowed:string Default: F
formatted	the above information combined (Out) Allowed:string Default: F
dolog	Log the version information to the logger (In) Allowed:boolean Default: T

Description
The output arguments are all defaulted so you can retrieve the ones you are interested in.

1.6.7 about - Function
Package utility
Module misc

Information about AIPS++

Synopsis
about(use_gui)
include "about.g"

Description
`about()` is used to version and other information about AIPS++ to the user.

If a GUI environment is available it puts it in a window, otherwise it logs the information.
This function will normally be called from an "About..." menu item.

Arguments

use_gui	Should we use a gui? Allowed:boolean Default: have_gui	

Example

```
- about()
```

1.6.8 printer - Tool

Package utility
Module misc

Print files
include "printer.g"

Constructors

printer	Create a printer access tool

Functions

gui	Send some files to the printer using a GUI
print	Send some files to the printer
printvalues	Print some Glish values
reinit	Reset the printing parameters

Description

The printer class provides access to "printing" text and postscript files both to a printer or to ghostview. The printing can take place in a number of formats (portrait, landscape, two-up) and paper formats (letter, A3, A4). The actual work is carried out behind the scenes by the `pri` shell script, which in turn calls `lwf_a`. Both of these are documented in the system manual.
A GUI control panel is also provided by this class – this is suitable for use

both by users directly, and through a "Print..." item on a GUI.
The major change that is needed is that defaults should be pulled in from aipsrc rather than assumed (printer.default is used though if the PRINTER environment variable is not set).

Example

```
- p := printer()
- p.gui()
```

printer - Constructor

Package	**utility**
Module	**misc**
Tool	**printer**

Create a printer access tool

Synopsis

printer(printername, mode, paper, display, printernote)

Arguments

printername	in	Device name of the printer. Defaults to PRINTER environment variable or printer.default aipsrc variable. Default: F
mode	in	How to print the file Allowed:p=portrait,l=landscape,80=two-up width=80,72=two-up width=72 Default: p
paper	in	Paper format Allowed:l=US letter, A4, A3 Default: l
display	in	Print to ghostview instead of print? Allowed:boolean Default: F
printernote	in	The note function to use to log our messages. Allowed:a note function Default: note

Returns

Printer tool

printer.reinit - Function

Package	**utility**
Module	**misc**
Tool	**printer**

Reset the printing parameters

Synopsis
reinit(printername, mode, paper, display)

Arguments

printername	in	Device name of the printer. Defaults to PRINTER environment variable. Default: F
mode	in	How to print the file Allowed:p=portrait,l=landscape,80=two-up width=80,72=two-up width=72 Default: p
paper	in	Paper format Allowed:l=US letter, A4, A3 Default: l
display	in	Print to ghostview instead of print? Allowed:boolean Default: F

---

printer.print - Function

Package	**utility**
Module	**misc**
Tool	**printer**

Send some files to the printer

Synopsis
print(files)

Arguments

files	in	List of file names (text or postscript) Allowed:string array

Description
This routine prints files (in the background), optionally removing them after printing them. It uses reports information via note about what it has done, whether some of the files don't exist, and so on.

---

printer.gui - Function

Package	**utility**
Module	**misc**
Tool	**printer**

Send some files to the printer using a GUI

Synopsis
gui(files, landscape)

Arguments

files	in	List of file names (text or postscript) Allowed:string array Default: ""
landscape	in	Default to landscape printing? Allowed:boolean Default: F

Description
While this routine is intended to be attached to a "Print..." menu item, it may also be called directly. If the list of files is set (not empty), then the user may-not edit the list, it will only print the files which are passed in. Probably there should be a preview button as well, and the ability to "freeze" other items besides the file list - in particular the "delete" button.

---

printer.printvalues - Function

Package	**utility**
Module	**misc**
Tool	**printer**

Print some Glish values

Synopsis
printvalues(values, needcr, usegui)

Arguments

values	in	Glish variable to be printed Allowed:any
needcr	in	Do we need carriage returns at the end of every line? Allowed:boolean Default: T
usegui	in	Use the GUI? Allowed:boolean Default: F

Description
This function formats a Glish variable (which might be an array), writes it into a temporary file, and prints it and then removes the temporary file. `needcr` normally needs to be True, the only time it doesn't is if you are printing a Glish string which already has embedded new lines.

Example

```
- p := printer(display=T)  # use ghostview
- p.printvalues(1:100)     # show 1 2 ... 100 in it
```

1.6.9 help - Function

Package utility
Module misc

Provides user help.

Synopsis
help(help about, browser, server)
include "aips2help.g"

Description
help will either drive a browser (beta release supports Netscape only) to the requested help page or will scroll help text. If the argument contains a :, as in Refman:table, it will drive the browser to the table module in the reference manual. If instead you type, help('table') it would display the output on the local screen. You may time help() and see the following:

```
Please provide an argument to help.  If you type

 help('aips')
        will summarize the modules available in the aips package.
        Other packages available are synthesis and NRAO. You may also
        supply a function, tool, or module name in place of aips and
        have help about that argument printed.


 help('Refman:')
        it will give you the WWW address and drive your WWW browser
        to the \aipspp  User's Reference Manual


 help('Glish:')
        will give you the WWW address manual and drive your WWW browser
        to the Glish User Manual.


help('Refman:table.table')
        will drive your web browser the table tool constructor of the
        table tool in the \aipspp  User's Reference Manual.
```

Arguments

help about	Help a user is looking for Allowed:String
browser	Optional browser to drive Allowed:Optional: netscape or mosaic Default: netscape
server	Optional server to use Allowed:Optional: any valid AIPS++documentation server Default: file://localhost

Returns
F if driving a browser, but prints help text to the local screen if not using a browser.

Example
There are two ways to use help, one prints help text to the glish window, the second drives a web browser (only Netscape for the beta release) to the appropriate place in the User Reference Manual.

```
help('table.close')

displays help about table.close in the local glish window.  To have help
drive a browser to a web page

help('Refman:table.close');

or to drive a different browser or server you could

help('Refman:table.close', browser='mosaic', server='http://aips2.nrao.edu')

the new browser and server are remembered so you don't need to subsequently
set them again.
```

If displaying help in the glish window, the argument may contain wild cards i.e catalog.* or *.catalog. All matches found will be displayed in a list. If only one match is found then its help text will be printed in the glish window. As a short cut, web() maybe used to drive the web browser to the appropriate page in the User Reference Manual.
If using the web browser as the means to display help, you need to specify which document to look at. In the above example, Refman: tells the help system to use the AIPS++User Reference Manual, table.close tell the help system to goto the label table.close. Table here refers to the table tool, and close to the table function close. You must specify the volume you wish to search by making it first in the argument list and seperating it with a : from the rest of the help string. If you don't use the : help will try and print a summary to your glish window. You may use either . or a space to seperate the help string. The following are equivalent

- Refman:table:close
- Refman:table.close
- Refman:table close

The first item in the list is the volume to search.
Help may use two aipsrc variables to alter the default documentation server and browser. These are:

help.browser The browser to use for driving help, the default is Netscape and

help.server The address for the help server, the default is http://localhost.

See Also
web

1.6.10 web - Function
Package utility
Module misc

Drives a web browser to the last requested help

Synopsis
web()

Description
If you request help using:

```
help('table.close')
```

text will be displayed in the glish window. web() will drive the browser to that location in the User Reference Manual. If no help was found web() will drive the browser to the AIPS++search page (The beta release only supports driving Netscape to a requested help page).

See Also
help

1.6.11 bug - Function
Package utility
Module misc

Provides a GUI to for user's to report problems.

Synopsis
bug()
include "aips2help.g"

Description

Bug provides a GUI to report AIPS++problems. Problem reports are initially sent to one of three AIPS++sites, Australia (ATNF-Epping), Europe (NFRA), or North America (NRAO-Socorro). If the aipsrc variable system.aipscenter (or system.aipscentre) is not set, then the query is set to the north america site. Possible options for system.aipscenter are namerica, europe, or australia. For best service please choose a site that is near you.

1.6.12 ask - Function

Package utility
Module misc

Provides a GUI to for user's to ask questions.

Synopsis
ask()
include "aips2help.g"

Description
Ask provides a GUI similar to bug that the user may ask questions of thier local AIPS++guru. If the aipsrc variable system.local.contact is not set, then the query is set to aips2-help@nrao.edu. The intent is for a local contact person to screen questions answering those they can and passing those question that they can't answer to the AIPS++group. A local AIPS++contact maybe specified using the aipsrc variable system.local.contact.

1.6.13 register - Function

Package utility
Module misc

Provides a GUI to for user's to register AIPS++.

Synopsis
register()
include "aips2help.g"

Description
Register provides a GUI for registering your AIPS++installation.

1.6.14 faq - Function

Package utility
Module misc

Drives the user's web browser to the AIPS++FAQ

Synopsis
faq()
include "aips2help.g"

Description
Typing faq() will driver the user's specified web browser, to the AIPS++FAQ page.

1.6.15 sh - Tool

Package utility
Module misc

A persistent shell client
include "sh.g"

Constructors

sh	Construct persistent shell client tool

Functions

command	execute command
done	destroy the tool

Description

The sh class allows the user to create persistent instances of a shell client, so that the overhead of creating the client can be amortized over many different function calls. This can result in a considerable time savings for functions that iterate over, say, files in a directory. Unlike shellcmd, this is really meant for fairly low level use and there is no logging option. This might be fixed later, if there is demand for it.

Example

```
- mysh := sh()                          # create the client
- mysh
[command=<function>, done=<function>]
- mysh.command('ls -1')                 # The basic use
                                        # output in .lines
[lines=Glish.txt MS NUAGES_SOLARIS News Rapport.ps Refman.ps aips++ aipsvars
 aipsviewrc.list bin catalog.g catalogsink.g displaylistbox.g foo.g foo.ps
 gcal glish.comments ... , status=0, errlines=]
- mysh.command('ls sfajlds')            # stderr in .errlines
                                        # status in .status
[lines=, status=2, errlines=sfajlds: No such file or directory]
- mysh.command('jasdfjls')
[lines=, status=1, errlines=]
- files:=mysh.command('ls -1').lines  # typical composite use
- length(files)
63
- mysh.done()                         # must be killed explicitly
```

sh - Constructor

Package	**utility**
Module	**misc**
Tool	**sh**

Construct persistent shell client tool

Synopsis
sh()

Returns
persistent shell client tool

---

sh.command - Function

Package	**utility**
Module	**misc**
Tool	**sh**

execute command

Synopsis
command(command, timeout)

Description
Executes any Bourne shell command. Remember to escape any special characters which have meaning to the shell.

Arguments

command	Command line to be executed by Bourne shell
timeout	Timeout interval (seconds): 0 for no timeout Allowed:Int Default: 0

Returns
execution result record
Fields in the returned record are

lines String array of output written by command, one string per line of output.

status Integer valued exit status from command execution

errlines Array of lines written to stderr by command

---

sh.done - Function

Package	**utility**
Module	**misc**
Tool	**sh**

destroy the tool

Synopsis
done()

1.6.16 substitute - Tool

Package utility
Module misc

Functions to substitute Glish variables and expressions
include "substitute.g"

Constructors

Functions

substitute	substitute Glish variables and expressions
substitutename	return the value of a variable name
substitutestring	return the value of a string variable
substitutevar	return the value of a variable

Description
The substitute function allows the user to embed Glish variables and expressions in a string and have them substituted. It is, for instance, used in TaQL (Table Query Language). E.g.

```
table.query ("column > $glishvar")
table.query ("column > $(var1+var2)")
```

Substitute allows the user also to use a Glish variable representing a tool (e.g. a table tool). E.g.

```
t1:=table("table1.dat")
t2:=table("table2.dat")
t1.query ("time in [select time from $t2 where windspeed<10]")
```

Further below are some more detailed examples.
The following rules apply:

1. Glish variables can be substituted by preceeding their name with a $ (as shown in the examples above). A variable name must start with an underscore or alphabetic, followed by zero or more alphanumerics and underscores.
2. Parts enclosed in single or double quotes are left untouched. Furthermore a $ can be escaped by a backslash, which is useful when an environment variable is used. Note that Glish requires an extra backslash to escape the backslash. The output contains the quotes and backslashes.

3. When name has a vector value, its substitution is enclosed in square brackets and separated by commas. It means that the result can directly be used in a TaQL command (or in a Glish command).

4. A string value is enclosed in double quotes. When the value contains a double quote, that quote is enclosed in single quotes. This is in line with the TaQL way of specifying string constants.

5. When the variable has a record value representing a tool (e.g. table), it is substituted by `$n` (n is a sequence number), while the table-id is added to the argument `idrec`. The first field in `idrec` is `nr` containing the number of substituted table-id's. The other fields contain the table-id's at each sequence number. The fields are unnamed, but can be accessed with the index operator such that `idrec[n+1]` contains the table-id of `$n`.
   The argument type tells which tool types are recognized. The default is `''` meaning that no tools are recognized.
   The argument `startseqnr` tells the first sequence number to use.
   For a tool to be valid the following have to be met:

   The tool must have a function `type()` returning a string containing the tool type.

   The tool must have a function `id()` returning the object-id of the tool.

   The tool types `table` and `region` do not meet these requirements. However, they are recognized and handled specifically.

6. When the name is unknown or has an unknown type, it is left untouched (thus the result is simply `$name`).

Furthermore it substitutes `$(expression)` by the expression result. It correctly handles parentheses and quotes in the expression. E.g.

```
$(a+b)
$(a)
$((a+b)*(a+b))
$(len("ab cd( de"))
```

Similar escape rules as above apply.
Substitution is NOT recursive. E.g. if `a:=1` and `b:="$a"`, the result of `substitute("$a")` is `"$a"` and not `1`.
Substitute has one problem due to the rules used by the eval function of Glish. Eval searches a variable only in the global scope. So if the substitution mechanism is used in a function, one has to use global variables if they are to be substituted. At the of the function they should be deleted. One has to be sure to use a unique name for the variable, e.g. by using the function name a s a suffix. E.g.

```
myfunc := function() {
  tab := table('mytable')
  global coldata_in_myfunc;
  coldata_in_myfunc := tab.getcol ('col');
  seltab := tab.query ('col > $(sum(coldata)/len(coldata))')
  symbol_delete ('coldata_in_myfunc');
}
```

Example

The following examples shows how variables and expressions are substituted.

```
- a:=10
- b:=20
- substitute("$a+$b")          #substitute both variables
10+20
- substitute("$(a+b)")         #substitute the expression
30
- substitute("$a$b")           #substitute both variables
1020
- substitute("$c")             #c is unknown, thus $c is returned
$c
- substitute("'$a'$b")         #$a is quoted, thus not substituted
'$a'20
- substitute("\\$a$b")         #first $ is escaped, thus $a not substituted
                               #note that \\ is needed to escape \ in glish
\$a20
- substitute('ab $(len("ab cd ef")) cd')    #substitute the $(len...
ab 3 cd
- substitute('$("ab cd ef")')  #a vector is enclosed in [] and separated by ,
                               #strings are enclosed in ""
["ab","cd","ef"]
- str:='ab"cd'
- substitute('$str')           #a " in a string is enclosed in ''
"ab"'"'"cd"
```

The following example shows how substitute is used in TaQL.

```
#   Open 2 tables.
- t:=table("/aips++/gvandiep/9800617.MS")
- t1:=table("/aips++/gvandiep/9800618.MS")
#   Show their handles
- t.handle()
[type=table, id=0, file=/aips++/gvandiep/9800617.MS]
- t1.handle()
[type=table, id=1, file=/aips++/gvandiep/9800618.MS]
#   Create a record (substitute clears it if not empty).
```

```
- idrec:=[=]
#   Substitute the various variables.
#   The table tools are replaced by sequence numbers (1 is first one).
- substitute("select from $t1 $t where col>$a+1",'table',1,idrec)
select from $1 $2 where col>10+1
#   idrec contains the number of sequence numbers (2) and the table id
#   belonging to each sequence number. They are used by the table client
#   to get the correct table for $1 and $2 in the query command.
- idrec
[nr=2, *25=1, *26=0]
```

substitute.substitute - Function

Package	**utility**
Module	**misc**
Tool	**substitute**

substitute Glish variables and expressions

Synopsis
substitute(string, type, startseqnr, idrec)

Description
It analyzes a string and substitutes embedded Glish variables and expressions by their value as explained in the general description. If `var` is a valid Glish variable, `$var` is substituted by its value. If `expr` is a valid Glish expression, `$(expr)` is substituted by the result of the expression.

Arguments

string	in	string to be analyzed
type	in	tool type recognized (default none)
startseqnr	in	first sequence number to use for substituted tools
idrec	out	record containing id's of substituted tools

Returns
input string with appropriate parts substituted

---

substitute.substitutename - Function

Package	**utility**
Module	**misc**
Tool	**substitute**

return the value of a variable name

Synopsis
substitutename(name, type, idrec)

Description
If `name` is the name of a valid Glish variable, the value of the variable is returned as a string. (string values are enclosed in "", vectors are enclosed in `[]`).
When `name` is a recognized tool (i.e. its type matches), a sequence number is returned as `$n`. The tool id is added to `idrec`.

Arguments

name	in	name of variable to be substituted
type	in	tool type recognized (default none)
idrec	inout	record containing id's of substituted tools

Returns
string with value of Glish variable

---

substitute.substitutevar - Function

Package	**utility**
Module	**misc**
Tool	**substitute**

return the value of a variable

Synopsis
substitutevar(var)

Description
If `var` has a known value (i.e. numeric ,string, boolean), its value is returned as a string. (string values are enclosed in "", vectors are enclosed in `[]`).

Arguments

var	in	variable

Returns
string with value of Glish variable

---

substitute.substitutestring - Function

Package	**utility**
Module	**misc**
Tool	**substitute**

return the value of a string variable

Synopsis
substitutestring(var)

Description
Return the value of a string variable. The string is enclosed in "". When the string contains a `"`, it is enclosed in single quotes. If the variable is a vector, the strings are enclosed in `[]` and separated by commas.

Arguments

var	in	variable

Returns
string with value of Glish variable

1.6.17 itemcontainer - Tool

Package	**utility**
Module	**misc**

A generic closure toolcontainer
include "itemcontainer.g"

Constructors

itemcontainer	Construct an itemcontainer

Functions

add	Add the item to the container
delete	Remove the item from the container
done	Destroy this itemcontainer
fromrecord	Add the fields of this record
get	Recover value of specified item
has_item	Does this item exist
length	Returns the number of items
makeconst	Make this itemcontainer const
makeunconst	Make this itemcontainer unconst
names	Returns list of item names
set	Put the named item into the container
torecord	Return the contents of the container as a record
type	Return the type of this tool

Description

This generic closure tool is designed to be used as a container of small items. Once created, the container can be made immutable. There is functionality to put items into the container, remove them and recover the contents of the container. `Items` can be of any type, and different types can be mixed in the one container.
Region of interest tools are actually `itemcomntainer` tools.

Example

We make a tool containing some smoothing kernel information. After invoking the `makeconst` function, you can no longer add items to the container or delete from it.

```
- include 'itemcontainer.g';
T
- include 'quanta.g';
T
- kernelType := 'gaussian';
- kernelWidth := dq.quantity(20,'arcsec');
- const i1 := itemcontainer();
- i1.set('type', kernelType);
T
```

```
- i1.set('width', kernelWidth);
T
- i1.makeconst();
T
- i1.set('comment', 'fish');
<fail>: This is a const object, you cannot write to it
        File:   itemcontainer.g, Line 228
        Stack:  .()
```

itemcontainer - Constructor

Package	**utility**
Module	**misc**
Tool	**itemcontainer**

Construct an itemcontainer

Synopsis
itemcontainer()

Returns
itemcontainer tool

Description

This is the only `itemcontainer` constructor.

Example

```
- include 'itemcontainer.g'
- i1 := itemcontainer()
```

---

itemcontainer.type - Function

Package	**utility**
Module	**misc**
Tool	**itemcontainer**

Return the type of this tool

Synopsis
type()

Returns
String

Description

This tool function returns the string "itemcontainer". If you have a tool, but don't know what kind it is, this function usually exists to tell you who the tool is.

Example

```
- myic := itemcontainer();
- if (is_defined('myic') && is_record(myic) &&
       has_field(myic, 'type') && myic.type() == 'itemcontainer') {
     print 'yippee';
  } else {
     print 'itemcontainers are not impressive enough for my work';
  }
```

Note the use of the short cut "&&" operator. A statement is only evaluated if the statement to the left is true.

---

itemcontainer.done - Function

Package	**utility**
Module	**misc**
Tool	**itemcontainer**

Destroy this itemcontainer

Synopsis
done()

Returns

Bool

Description

This tool function destroys the state of the current `itemcontainer` tool. The tool still exists, but it is no longer an `itemcontainer` !

Example

```
- const myic := itemcontainer();
- myic.done();
T
- print 'myic=', myic;
myic=F
```

---

itemcontainer.has_item - Function

Package	**utility**
Module	**misc**
Tool	**itemcontainer**

Does this item exist

Synopsis

has_item()

Returns

Bool

Description

This tool function indicates whether the named item exists in the `itemcontainer` or not.

Example

```
- const i1:= itemcontainer();
- i1.set('doggies', "retriever spaniel siamese");
T
- i1.has_item('doggies');
T
- i1.has_item('fish');
F
```

---

itemcontainer.names - Function

Package	**utility**
Module	**misc**
Tool	**itemcontainer**

Returns list of item names

Synopsis
names()

Returns
Vector of strings

Description

This tool function returns the list of item names. Note that any items added with the add will have a field name like "*n" where "n" is a number.

Example

```
- const fish := itemcontainer();
- fish.set('types', "salmon trout gold");
T
- fish.set('fish1', 'trout');
T
- fish.add('salmon');
3
- print 'names=', fish.names();
names=types fish1 *1
```

---

itemcontainer.makeconst - Function

Package	**utility**
Module	**misc**
Tool	**itemcontainer**

Make this itemcontainer const

Synopsis
makeconst()

Returns
Boolean

Description

After application of this tool function, you can no longer change the internal state of the `itemcontainer` tool.

Example

```
- i1 := itemcontainer();
T
- i1.set('par1', 20);
T
- i1.set('par2', 30);
T
- i1.makeconst();
T
- i1.set('par3', 40);
This is a const tool, you cannot overwrite it
<fail>: This is a const object, you cannot overwrite it
        File:   itemcontainer.g, Line 56
        Stack:  .()
```

Note that this is a different error from

```
- const i1 := itemcontainer()
- i1 := 'fish'
<fail>: <fail>: 'const' values cannot be modified.
```

Here Glish is complaining that it can't reassign the tool `i1`. The previous example is illustrating that after application of the `makeconst` function, you can't change the internal state of the `itemcontainer` tool (nothing to do with trying to overwrite it).

---

itemcontainer.makeunconst - Function

Package	**utility**
Module	**misc**
Tool	**itemcontainer**

Make this itemcontainer unconst

Synopsis
makeunconst()

Returns
Boolean

Description

After application of this tool function, you can change the internal state of the `itemcontainer` tool again.

Example

```
- i1 := itemcontainer();
T
- i1.set('par1', 20);
- i1.makeconst();
- i1.set('par3', 40);
This is a const tool, you cannot overwrite it
<fail>: This is a const object, you cannot overwrite it
        File:   itemcontainer.g, Line 56
        Stack:  .()
- i1.makeunconst();
- i1.set('par3', 40);
```

---

itemcontainer.length - Function

Package	**utility**
Module	**misc**
Tool	**itemcontainer**

Returns the number of items

Synopsis
length(showdeleted)

Arguments

showdeleted	yes or no Allowed:Bool Default: F

Returns
Integer

Description

This tool function returns the number of items in the list of item names. `itemcontainer`. If `showdeleted` is set to T, illegal fields are counted.

Example

```
- const fish := itemcontainer();
- fish.set('types', "salmon trout gold");
T
- fish.set('fish1', 'trout');
T
- fish.add('salmon');
3
- print 'number=', fish.length();
number=3
```

---

itemcontainer.torecord - Function

Package	**utility**
Module	**misc**
Tool	**itemcontainer**

Return the contents of the container as a record

Synopsis
torecord(showdeleted)

Arguments

showdeleted	yes or no Allowed:Bool Default: F	

Returns
Record

Description

This tool function recovers the contents of the container into a record where each item has its own named (or numbered) field. If `showdeleted` is T, even "illegal" fields are copied.

Example

```
- const fish := itemcontainer();
- fish.set('types', "salmon trout gold");
T
- fish.set('fish1', 'trout');
T
- fish.add('salmon');
3
- x := fish.torecord();
- x
[types=salmon trout gold, fish1=trout, *1=salmon]
- is_record(x);
T
```

---

itemcontainer.fromrecord - Function

Package	**utility**
Module	**misc**
Tool	**itemcontainer**

Add the fields of this record

Synopsis
fromrecord()

Returns
Bool

Description

This tool function makes one item per field in the supplied record.

Example

```
- const fish := itemcontainer();
- rec := [=];
- rec.types := "salmon trout gold";
- rec.fish1 := 'trout';
- rec.fish2 := 'salmon';
- fish.fromrecord(rec);
T
- fish.names();
types fish1 fish2
- fish.torecord();
[types=salmon trout gold, fish1=trout, fish2=salmon]
```

---

itemcontainer.set - Function

Package	**utility**
Module	**misc**
Tool	**itemcontainer**

Put the named item into the container

Synopsis
set(item, value)

Arguments

item	Item name Allowed:String
value	Item Allowed:Any

Returns
Bool

Description

This tool function puts the specified item into the container with the given name.

Example

```
- include 'quanta.g';
T
- const i1 := itemcontainer();
- i1.set('type', 'boxcar');
T
- i1.set('width', dq.quantity(10,'arcsec'));
T
```

---

itemcontainer.add - Function

Package	**utility**
Module	**misc**
Tool	**itemcontainer**

Add the item to the container

Synopsis
add(value)

Arguments

value	Item Allowed:Any

Returns
Integer

Description

This tool function adds the specified item into the container. The item is given a number one higher than the number of existing items in the container. This tool function returns the item number for the new item.

Example

```
- include 'quanta.g';
T
- const i1 := itemcontainer();
- idx := i1.add(dq.quantity(10,'arcsec'));
- x := i1.get(idx);
- x;
[value=10, unit=arcsec]
```

---

itemcontainer.delete - Function

Package	**utility**
Module	**misc**
Tool	**itemcontainer**

Remove the item from the container

Synopsis
delete(item)

Arguments

item	Item Allowed:String or Integer

Returns

Description

This tool function removes the specified `item` from the container. Item can be an item name or the number of an item. (It does not remove the field from the container to preserve indexing, but sets the item's value to illegal.)

Example

```
- include 'quanta.g';
T
- const i1 := itemcontainer();
- idx := i1.add(dq.quantity(10,'arcsec'));
- x := i1.get(idx);
- x;
[value=10, unit=arcsec]
- i1.delete(idx);
T
- i1.get(idx);
<fail>: Requested item "1" is not present
        File:   ./itemcontainer.g, Line 152
        Stack:  .()
-
```

---

itemcontainer.get - Function

Package	**utility**
Module	**misc**
Tool	**itemcontainer**

Recover value of specified item

Synopsis
get(item)

Arguments

item	Item name or number Allowed:String or integer

Returns

Glish variable

Description

This tool function recovers the specified variable into a Glish variable. You can specify the `item` either by its name or by its number.

Example

```
- include 'quanta.g';
T
- const i1 := itemcontainer();
- i1.set('type', 'boxcar');
T
- idx := i1.add(dq.quantity(10,'arcsec'));
- type := i1.get('type');
- type;
boxcar
- type := i1.get(1);
- type;
boxcar
- width := i1.get(2);
- width;
[value=10, unit=arcsec]
- x := i1.get(idx);
- x;
[value=10, unit=arcsec]
```

1.6.18 is_itemcontainer - Function

Package utility
Module misc

Is this an itemcontainer tool?

Synopsis
is_itemcontainer(thing)

Arguments

thing	The thing to test Allowed: Any

Returns
Bool

Description

This global function determines whether this is an `itemcontainer` tool.

Example

```
- include 'regionmanager.g';
T
- r1 := drm.box();
- is_itemcontainer(r1);     # Regions are made from itemcontainers
T
- i1 := itemcontainer();
- is_itemcontainer(i1);
T
- x := [20];
- is_itemcontainer(x);
F
```

There are four required fields, Name, Email, Brief Description and Detail Description. If data is missing in any of these fields your bug report will not be sent.

1.6.19 ask - Function
Package utility
Module misc

Displays the AIPS++ question submission form

Synopsis
ask()

Description
Ask allows you to ask a question of the AIPS++ Project directly from Glish/Tk. You fill out the requested information and press the send button. The question is then sent to aips2bugs@aoc.nrao.edu and put into the AIPS++ bug reporting and tracking system. You will receive an acknowledgment once the question is in the system.

1.6.20 assay - Tool

Package	**utility**
Module	**misc**

Assay test and demo functions
include "assay.g"

Constructors

Functions

demos	Returns names of the standard demonstrations
tests	Returns names of the standard tests
try	Try each of a list of functions
trydemos	Tries the standard demonstrations
trytests	Tries the standard tests

Description
The assay class is used to track the results of running the test and demonstration functions. It is useful for verifying an installation. It can also assay any list of functions. The functions must not require arguments.
The functions to be tested should use the Glish fail mechanism for assay to be meaningful.
The standard tests and demonstrations are those that are expected to not fail in a working installation.

Example

```
- include "assay.g"
T
- assay().trytests()
```

assay.try - Function

Package	**utility**
Module	**misc**
Tool	**assay**

Try each of a list of functions

Synopsis
try(functionlist)

Description
For each function in the list, run the function (*without arguments*) and trap any fails. At the end, the failures and corresponding messages are logged, and the number of failures is returned.
Note that the argument is a list of function *names*, not the functions themselves.

Arguments

functionlist	The list of functions

Returns
Int - Number that failed

Example

```
- include "assay.g"
T
- assay().try('foo bar tabletest')
foo failed:  argument must be defined
bar failed:  argument must be defined
2
```

---

assay.tests - Function

Package	**utility**
Module	**misc**
Tool	**assay**

Returns names of the standard tests

Synopsis
tests()

---

assay.demos - Function

Package	**utility**
Module	**misc**
Tool	**assay**

Returns names of the standard demonstrations

Synopsis
demos()

---

assay.trytests - Function

Package	**utility**
Module	**misc**
Tool	**assay**

Tries the standard tests

Synopsis
trytests(tests)

Arguments

tests	Names of tests Allowed:String Default: All tests

Returns
Int - Number that failed

---

assay.trydemos - Function

Package	**utility**
Module	**misc**
Tool	**assay**

Tries the standard demonstrations

Synopsis
trydemos(demos)

Arguments

demos	Names of demos Allowed:String Default: All demos

Returns
Int - Number that failed

1.6.21 stopwatch - Tool

Package	**utility**
Module	**misc**

stopwatch style timer functions
include "stopwatch.g"

Constructors

stopwatch	Construct stopwatch tool

Functions

delete	destroy the tool
fmtvalue	formatted string value of timer with units
help	display help
reset	reset time value to zero
show	show value and run state of timer
start	start timer running

state	get run state of timer
stop	stop timer
timefunc	time a function
value	get numerical value of timer in seconds
zero	stop and reset timer

Description

The stopwatch class provides a set of timer functions with the functionality of a traditional stopwatch. Any number of stopwatches may be created, which can then individually be started, stopped, reset, displayed and interrogated. A function can be passed to a stopwatch, which is then run and timed. This is particularly useful for benchmarking purposes. Due to the distributed process nature of AIPS++, it is very difficult correctly to account for all the resources used by a given calculation. Thus only wall clock time is implemented in this class, as opposed to the "system, user and wall clock" accounting often found in simple single threaded applications. Multiple stopwatch timers will not interfere with each other.

Example

```
- include "stopwatch.g"              # no timer is created by default
T
- tim := stopwatch()
- tim.show()                         # timer created running
timer is RUNNING, value is 3.56s
F
- tim.stop()                         # stop and start
timer stopped at 9.31s
F
- tim.show()
timer is STOPPED, value is 9.31s
F
- tim.start()                        # restartable
timer restarted from 9.31s
F
-tim.reset()                         # reset doesn't change state
timer reset to zero
F
- tim.show()
timer is RUNNING, value is 12.35s
F
```

```
- tim.stop()                        # values can be retrieved
timer stopped at 1m 8.73s
F
- tim.value()
68.733093
- tim.fmtvalue()
1m 8.73s
- tim.state()
STOPPED
- tim.start(quiet=T)                # suppress messages
- split_time := tim.value()         # a value while running is a split time
- split_time
74.526214
```

See timefunc for examples of how to time functions. stopwatch tools also provide standard functions `ok()` and `delete()`.

stopwatch - Constructor

Package	**utility**
Module	**misc**
Tool	**stopwatch**

Construct stopwatch tool

Synopsis
stopwatch()

Description
Creates a stopwatch tool, initialized to zero and running. Any number of stopwatches may be created without interfering with each other.

Returns
stopwatch tool

---

stopwatch.start - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

start timer running

Synopsis
start(quiet)

Description
In the running state, the timer advances at real time. The value of the timer is not reset to zero, and the timer may be restarted after it has been stopped.

Arguments

quiet	Suppress status output? Default: F

A timer is created zeroed and running, so you may not need to call `start()` explicitly. Similarly, there is no need to directly set the state of the timer when timing functions,

---

stopwatch.stop - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

stop timer

Synopsis
stop(quiet)

Description
In the stopped state, the timer does not advance, though it can be interrogated for the value in different formats. It may be restarted.

Arguments

quiet	Suppress status output? Default: F

---

stopwatch.reset - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

reset time value to zero

Synopsis
reset(quiet)

Description
The timer value is reset to zero, but the run state is not changed.

Arguments

quiet	Suppress status output? Default: F

---

stopwatch.zero - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

stop and reset timer

Synopsis
zero(quiet)

Description
The timer is stopped and the value is reset to zero.

Arguments

quiet	Suppress status output? Default: F

---

stopwatch.value - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

get numerical value of timer in seconds

Synopsis
value()

Returns
numerical value of timer in seconds

---

stopwatch.fmtvalue - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

formatted string value of timer with units

Synopsis
fmtvalue(value)

Arguments

value	Time in seconds to be formatted Default: Current value()

Returns
formatted time string value with units

Description
If no explicit value is passed to the function, the current value of the stopwatch will be used. Leading fields with zero value are suppressed.

Example
An extreme example might be

```
- tim.fmtvalue()
27y 306d 20h 45m 54.44s
```

Or when passing in an explicit value

```
- tim.fmtvalue(90)
1m 30.00s
```

---

stopwatch.state - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

get run state of timer

Synopsis
state()

Returns
either 'RUNNING' or 'STOPPED'

---

stopwatch.show - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

show value and run state of timer

Synopsis
show()

Example

```
- tim.show()
timer is RUNNING, value is 1m 8.73s
F
```

---

stopwatch.timefunc - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

time a function

Synopsis
timefunc(func, niter, name, quiet)

Description
Run a function, and display the runtime. The function to be timed should take no arguments, but constant arguments may be passed in by defining a function expression in the call. After the function is run, the numerical value of the runtime can be retrieved from the time with value.

Arguments

func	Function to be timed
niter	Number of iterations Default: 1
name	Output label Default: ” or func
quiet	Suppress status output? Default: F

Example

```
                                     # a function that might take a while
- function iter_sin (iter=10,len=10000) {
+ v:=[1:len]; while (iter>0) { iter:=iter-1; v := sin(v); } v; }

- tim.timefunc(iter_sin)             # time it
Timing function...
return value is [0.462957899 0.471936887 0.137069347 -0.448762012 ... ]
Function took 0.15s  to run
F
-                                    # can supply args this way
+ tim.timefunc(function () { iter_sin(1000) })
Timing function...
```

```
return value is [0.0545929715 0.0546066474 0.0510562464 -0.0545696198 ... ]
Function took 9.31s  to run
F
-                                    # or this way
+ function tmpfunc() { iter_sin(1000) }
tim.timefunc(tmpfunc,quiet=T)
F
- tim.show()
timer is STOPPED, value is 9.21s      # timings are reasonably consistent
F
-                                    # can average multiple runs
+ t.timefunc(function() { iter_sin(1000)}, 3, 'big_demo')
Timing function big_demo 3 times...
return value is [0.0545929715 0.0546066474 0.0510562464 -0.0545696198 ... ]
Function big_demo took 9.20s to run
return value is [0.0545929715 0.0546066474 0.0510562464 -0.0545696198 ... ]
Function big_demo took 9.26s to run
return value is [0.0545929715 0.0546066474 0.0510562464 -0.0545696198 ... ]
Function big_demo took 9.31s to run

Function big_demo took an average of 9.26s to run, over 3 iterations
T
```

The func argument may also be be a string containing the name of the function to be timed. In this case, the name argument will default to the function name.

---

stopwatch.help - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

display help

Synopsis
help()

Example

```
- tim.help()
Stopwatch style timing.  Available member functions:
```

```
tim := stopwatch()  # timer is created running and zeroed

tim.start()       # timer returns mutable values
tim.stop()        # timer returns fixed values
tim.reset()       # zeros time w/o state change
tim.zero()        # stop followed by reset
tim.state()       # returns 'RUNNING' or 'STOPPED'
tim.value()       # return current value in seconds
tim.fmtvalue(v)   # return string formatted value, with units
tim.show()        # print state and value of timer
tim.timefunc(func,niter,label)
                  # time function f.  Should take no parameters
tim.delete()      # destroy the tool

Most functions take an optional quiet argument to suppress informational
messages.  Notice that tim.timefunc(function () {foo(5)}) is a valid call
if you wish to avoid defining a new function to provide arguments.  After
timing a function, the numerical interval can be retrieved with tim.value()
F
```

---

stopwatch.delete - Function

Package	**utility**
Module	**misc**
Tool	**stopwatch**

destroy the tool

Synopsis
delete()

1.6.22 edit - Function

Package	**utility**
Module	**misc**

Edit and optionally execute a file

Synopsis
edit(file, execute, editor)
include "edit.g"

Synopsis
edit(file, execute=T, editor='')(file, execute, editor)

Description
Edit 'file'. If 'editor' is set, use it, otherwise use the editor specified by the 'EDITOR' environment variable if set, otherwise use 'vi'. If 'execute' is T, execute (i.e. include), the file.

Arguments

file	File to edit Allowed:string
execute	Read the file into glish? Allowed:T or F Default: T
editor	Editor to use, if unset use EDITOR Allowed:string Default: ''

Returns
T if OK, fail or F on failure

Example

```
ok := edit('myfile.g');
ok := edit('myfile.g', execute=F);    # Edit, but don't execute
ok := edit('myfile.g', editor='vi');  # Edit with vi
```

1.6.23 note - Function
Package utility
Module misc

Send a message to the logger or terminal

Synopsis
note(. . . , priority, origin, time, postcli)
include "note.g"

Synopsis
edit(..., priority='NORMAL', origin='Glish', time='')(. . . , priority, origin, time, postcli)

Description
Use `note` to send a message to the logger if AIPS++is running, otherwise it sends it to the screen. This is useful if you are writing Glish functions that should function in AIPS++and non-AIPS++environments. In particular, sending a message doesn't start AIPS++, but if AIPS++is already started then the message is logged.
The priority defaults to `NORMAL` and the origin to `Glish`, however these can be overridden if desired.

Arguments

. . .	in	components of the message Allowed:string
priority	in	Message priority Allowed:DEBUGGING,NORMAL,WARN,SEVERE Default: NORMAL
origin	in	source of the message Allowed:string Default: 'Glish'
time	in	time as a string Allowed:string Default: Current time
postcli	in	Show message in CLI window too? Allowed:Bool Default: F

Example

```
- note('The value of x is', x);
```

See Also

1. The AIPS++logger
2. throw

1.6.24 throw - Function
Package utility
Module misc

Send a SEVERE message, then fail

Synopsis
throw(. . . , origin, postcli)
include "note.g"

Synopsis
throw(..., origin='Glish')(. . . , origin, postcli)

Description
It is often useful to be able to both print an error message (to the logger if it is defined, otherwise to the terminal), and then return a fail. This function implements that idiom.
The priority is always set to `SEVERE`.

Arguments

. . .	in	components of the message Allowed:string
origin	in	source of the message Allowed:string Default: 'Glish'
postcli	in	Show message in CLI window too? Allowed:Bool Default: F

Returns
fail with the message in it

Example

```
myfunc := function(x) {
    if (!is_numeric(x))
        return throw('I can only deal with numeric values of x, not ',
                     type_name(x), origin='myfunc');
}
```

See Also

1. The AIPS++logger
2. note

1.6.25 plugins - Tool

Package utility
Module misc

attach code to tools at runtime
include "plugins.g"

Constructors

Functions

attach	Attach plugins for a type to an tool
loaded	list which plugins have been loaded
loadplugins	load plugin (.gp) files for some class
rescan	Find all plugin (.gp) files

Description

`plugins` is the technique by which additional Glish functions are attached to tools when that tool is created. For example, a user might want to attach some code that automatically fits models to data to a pgplotter, or a site might wish to attach telescope-specific calibration functions to standard data reduction classes.
The function is a simple one. When `plugins.g` is first included, it searches all directories in its include path (Glish variable `system.path.include`) for files of the form `class_*.gp`, where `.gp` means "Glish plugin", "class" is the tool type that the plug refers to, and the "`*`" is a category for the types of functionality within that plugin. For example, `pgplotter_fitting.gp` might

be the name of a plugin for attaching fitting functionality to a pgplotter.
The files are not actually loaded at this stage, they are just located. Then when an tool of a type that has plugins is actually created, the plugin code is attached to the tool just before the tool is returned to the user.
The plugin system is careful to do as little work as possible. In particular, it loads files only on demand, and does not require any services from the rest of AIPS++that might take more than a brief time to start.
`plugins` is a single tool. Not more than one such tool can ever be created (automatically via the inclusion of `plugins.g`) in any Glish session.

Example
If you want to plugin-enable your classes, you merely have to:

1. include `plugins.g` at the beginning of your `.g` file; and

2. include Call `plugins.attach` before returning the tool from your constructor

That's it! It is so simple and low-cost, that generally speaking all user-level classes should be plugin-enabled.
Here is a simple example.of a plugin-enabled tool that just keeps track of a counter:

```
pragma include once
include 'plugins.g'                         ### 1

counter := function() { # constructor
    private := public := [=];
    private.count := 0;

    # Member functions
    public.bump := function() {
        wider private;
        private.count +:= 1;
        return T;
    }

    public.total := function() {
        wider private;
        return private.count;
    }

    plugins.attach('counter', public);     ### 2
    return ref public; # tool return
}
```

The numbered lines are all that is necessary to plugin-enable the counter class. The unaugmented class can be used like this:

```
- c := counter()
- c.bump(); c.total();
T
1
```

Now, suppose we are a user who wants to customize this simple counter so that we can "bump" the counter an arbitrary number of times. We could create the following plugin to attach a "multibump" function.

```
# counter_mychanges.gp
pragma include once;

counter_mychanges := [=];

counter_mychanges.attach := function(ref public) {
    # Multibump function
    public.multibump := function(count) {
        for (i in 1:count) { public.bump(); }
        return T;
    }

    return T; # Say that the attach worked
}
```

Note how we can both extend the list of functions in the plugin by just adding a function to public, and how we can use other public functions. There is no access to the private state of the class through a plugin – it preserves encapsulation.
A careful programmer would check to see whether or not a "multibump" function already exists before adding one, and returning a "fail" (probably via the throw function) if it does.
Now we can just use the multibump function as if it was defined in the original class, even though it has been added after the fact:

```
+ c.multibump(5); c.total()
T
6
```

If there was any one-time initialization code that was needed by any of the functions in the file, you would implement them in a function:

```
counter_mychanges.init := function() # no arguments
{
    ...        # initialization code
    return T;  # or fail if there is an error
}
```

plugins.rescan - Function

Package	**utility**
Module	**misc**
Tool	**plugins**

Find all plugin (.gp) files

Synopsis
rescan()

Description
This function is used to find all plugin files. It is called automatically when the `plugins` tool is constructed, so you generally never need to call it. You might call it if you know that some new `*.gp` files have been created since you started your Glish session.
The rules are that it searches all directories in `system.path.include` for `.gp` files that are of the form `class_something.gp`, where "class" is the classname this plugin is to be attached to, and something is a general description (category) of what functionality is provided by the plugin (for example, "demo").

---

plugins.loadplugins - Function

Package	**utility**
Module	**misc**
Tool	**plugins**

load plugin (.gp) files for some class

Synopsis
loadplugins(type)

Description
This function loads all the plugin files for some class (*i.e.*, type). You should never have to call it, since `plugins` itself will call it the first time it is needed for that class.
If a plugin has an "init" function, `loadplugins` will call it after the file has been included.

Arguments

type	type (class) for which we want to load plugins Allowed:string

---

plugins.loaded - Function

Package	**utility**
Module	**misc**
Tool	**plugins**

list which plugins have been loaded

Synopsis
loaded()

Description
This function returns a record with information about what plugins have been loaded. It is probably not of any general interest.

Returns
record

---

plugins.attach - Function

Package	**utility**
Module	**misc**
Tool	**plugins**

Attach plugins for a type to an tool

Synopsis
attach(type, objpublic)

Description
This is the only function that most programmers will have to call. You call it from the constructor(s) of your class. Its arguments are the class name (type),

and the tool you are about to return to the user (that is, you pass it "public" if you are using the usual closure-tool idiom).
If necessary, it loads all the plugin (`.gp`) files for the requested types, and runs any plugin `init()` functions it finds. It then calls all the `attach()` functions on the loaded plugins. Those attach functions add functions to the tool.

Arguments

type	type (class) for which we want to load plugins Allowed:string
objpublic	the tool the constructor will return to the caller Allowed:record

Returns
reference to objpublic (so you can return it directly)

1.6.26 timer - Tool
Package utility
Module misc

Execute functions periodically, wait for a specified time
include "timer.g"

Constructors

Functions

execute	execute a function periodically
remove	remove a callback function from the timer
wait	wait a specified amount of time

Description
`timer` is an tool which is used to execute a function periodically (or once after a timeout), or to wait for a specified period of time.
`timer` is a single tool, only one will be created in a given glish session. The timer is created the first time `timer.g` is included in a glish session.

Example

We can implement a simple "reminder" as follows:

```
- reminder := function(interval, name) { print 'Do not forget your meeting!'; }
-                                                # Run reminder every 5 minutes
- id := timer.execute(reminder, interval=300, oneshot=F)
- ... time passes ... the following message comes up every 5 minutes
Do not forget your meeting!
- ...
- timer.remove(id);   # Turn off reminder
```

timer.execute - Function

Package	**utility**
Module	**misc**
Tool	**timer**

execute a function periodically

Synopsis
execute(callback, interval, oneshot)

Description
`execute` takes a function which you supply (a "callback"), and executes it one or more times after a time that you specify has passed. The function returns a string, which you can use in the `remove` function to deactivate execution of the function.

The callback function you supply must take two arguments, which will be passed to the function when it is called by `timer`.

interval The time interval that was supplied to `execute()`.

name The name of the timing event, *i.e.*, the value returned by `execute()`.

Often the callback will just ignore these arguments

You can specify that the function is to be executed only once ("oneshot"), or "forever" (until deactivated with `remove`).

Arguments

callback	function to be executed Allowed:two-argument function
interval	how long to wait before executing function (seconds) Allowed:float Default: 60
oneshot	execute function once only? Allowed:boolean Default: T

Returns
string identifier, for remove, or fail

---

timer.remove - Function

Package	**utility**
Module	**misc**
Tool	**timer**

remove a callback function from the timer

Synopsis
remove(name)

Description
`remove` removes a callback function from `timer`. You use the value which returned from the `execute` function to determine which callback to remove. It returns `T` on success, `F` if there is no such callback (for example, it might have already been removed), or "fail" on error.

Arguments

name	identifier returned from execute Allowed:string

Returns
T if callback removed, F if no callback, or fail on error

---

timer.wait - Function

Package	**utility**
Module	**misc**
Tool	**timer**

wait a specified amount of time

Synopsis
wait(interval)

Description
`wait` merely waits the specified period of time, and then returns. C and C++ programmers might think of it as "sleep."

Arguments

interval	how long to wait (seconds) Allowed:float

Returns
T

1.6.27 serverexists - Function

Package	**utility**
Module	**misc**

Does the specified server exist

Synopsis
serverexists(symbol, name, server)
include "serverexists.g"

Arguments

symbol	Symbol of interest Allowed:String
name	The name of the type of server Allowed:String
server	The server tool Allowed:Glish closure tool

Returns
T or F

Description

This function can be used to see whether an AIPS++ server tool exists or not. It relies upon the Glish tool having the function `type` which returns the `name` of the type of server. For example, "quanta" for a `quanta` server, or "regionmanager" for a `regionmanager` server. All of the AIPS++ servers (or tools) should have this function.

Example

```
- include 'quanta,g'
- include 'serverexists.g'
- serverexists('dq', 'quanta', dq);
T
- symbol_delete('dq');
- serverexists('dq', 'quanta', dq);
F
```

1.6.28 aips2loc - Function

Package utility
Module misc

Plot lines of code in AIPS++

Synopsis
aips2loc(file)
include "aips2loc.g"

Constructors

Functions

execute	execute a function periodically
remove	remove a callback function from the timer
wait	wait a specified amount of time

Description

Plot the lines of code in AIPS++ as a function of date. This only works at the AOC unless you wish to ftp the file from
ftp://aips2.nrao.edu/../linecount/source_lines.

Arguments

file	Name of text file containing source lines Allowed:String Default: /export/aips++/linecount/source_lines

Example

```
include 'aips2loc.g'
aips2loc();
```

1.6.29 ftp - Tool

Package utility
Module misc

Tool to ftp files
include "ftp.g"

Constructors

ftp	Construct an ftp tool

Functions

ascii	Use ascii mode for transfer
binary	Use binary mode for transfer
cd	Change directory

connect	Connect to the host
disconnect	Disconnect from the host
get	Get a file
send	Send a file
type	Return the type of this tool

Description

This tool gets and sends files using ftp. While it is not a particularly good interface to ftp, it is adequate for most simple applications.

ftp - Constructor

Package	**utility**
Module	**misc**
Tool	**ftp**

Construct an ftp tool

Synopsis
ftp(host, user, pass, command, prompt, verbose)

Returns
ftp tool

Description
This is the only `ftp` constructor.

Arguments

host	Host to connect to Allowed:String Default: aips2.nrao.edu
user	User name Allowed:String Default: anonymous
. pass	Password Allowed:String Default: e-mail address
command	ftp command string Allowed:String Default: ftp -n -v -i
. prompt	ftp command prompt Allowed:String Default: ftp¿
. verbose	Report lots of information? Allowed:Boolean Default: T—F

ftp.type - Function

Package	**utility**
Module	**misc**
Tool	**ftp**

Return the type of this tool

Synopsis
type()

Description
This tool function returns the string "ftp". If you have a tool, but don't know what kind it is, this function usually exists to tell you who the tool is.

Returns
String

---

ftp.connect - Function

Package	**utility**
Module	**misc**
Tool	**ftp**

Connect to the host

Synopsis
connect()

Description
This tool function connects to the host.

Returns
Boolean

---

ftp.disconnect - Function

Package	**utility**
Module	**misc**
Tool	**ftp**

Disconnect from the host

Synopsis
disconnect()

Description
This tool function disconnects from the host.

Returns
Boolean

---

ftp.binary - Function

Package	**utility**
Module	**misc**
Tool	**ftp**

Use binary mode for transfer

Synopsis
binary()

Description
Tell ftp to use binary mode when transferring files.

Returns
Boolean

---

ftp.ascii - Function

Package	**utility**
Module	**misc**
Tool	**ftp**

Use ascii mode for transfer

Synopsis
ascii()

Description
Tell ftp to use ascii mode when transferring files.

Returns
Boolean

---

ftp.get - Function

Package	**utility**
Module	**misc**
Tool	**ftp**

Get a file

Synopsis
get(file)

Description
This tool function gets a file.

Arguments

file	File name Allowed:String

Returns
Boolean

---

ftp.send - Function

Package	**utility**
Module	**misc**
Tool	**ftp**

Send a file

Synopsis
send(file)

Description
This tool function sends a file.

Arguments

file	File name Allowed:String

Returns
Boolean

---

ftp.cd - Function

Package	**utility**
Module	**misc**
Tool	**ftp**

Change directory

Synopsis
cd(dir)

Description
This tool function changes the current directory.

Arguments

dir	Directory name Allowed:String

Returns
Boolean

1.6.30 ftptest - Function

Package utility
Module misc

Test the ftp tool

Synopsis
ftptest(verbose)

Description
This global function tests the ftp tool by anonymous connection to aips2.nrao.edu from where the file source_lines is retrieved.

Arguments

verbose	Report lots of information? Allowed:Boolean Default: T—F

Returns
Boolean

1.6.31 make - Function
Package utility
Module misc

Function for a production system

Synopsis
make(target, args, makefile, Execute, script, debug)
include "make.g"

Description

`make`
`make` implements a *production system.* A production system makes a product or products according to some well-specified rules. It is of very widespread applicability but it's best suited to automating a set of steps that must be carried out in sequence but with some variation if, for example, a given product is sometimes available and sometimes not. The principle is familiar to users of the Unix make system, on which make is built. `make` keeps track of which files have changed and knows to remake dependent products.
`make` needs a *makefile* that specifies the production rules.

```
.SUFFIXES: .image .ms .fits

.fits.ms :
glish include 'ms.g';m:=fitstoms('$@', '$<');m.done();

.ms.image :
glish include 'imager.g';myimgr:=imager('$<');myimgr.advise(takeadvice=T,fieldofview='

clean:
shell rm -rf *.fits *.ms *.image

data:
shell cp ${AIPSROOT}/data/demo/3C273XC1.fits .
```

By using this makefile, make knows how to make the image, 3C273XC1.image, from the MeasurementSet, 3C273XC1.ms, and it knows how to make the MeasurementSet from the original FITS file. The glish commands are:

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

The arguments allowed are:

```
-D <VARIABLE>
    define <VARIABLE> to be 1

-I <DIR>
    Specify a directory in which to search for makefiles
    and included makefiles.

-d flags
    Turn on debugging, and specify which portions of make are to
    print debugging information.  Flags is one or more of the following:

       A       Print all possible debugging information; equivalent to
               specifying all of the debugging flags.

       d       Print debugging information about directory searching and
               caching.

       g1      Print the input graph before making anything.

       g2      Print the input graph after making everything, or before
               exiting on error.

       m       Print debugging information about making targets,
               including modification dates.

       s       Print debugging information about suffix-transformation
               rules.

       t       Print debugging information about target list
               maintenance.

       v       Print debugging information about variable assignment.

-e      Specify that environmental variables override macro assignments
        within makefiles.
```

If a script is specified, then `make` generates the commands and writes these to the script file. If the argument `execute` is set to T then the commands are also executed. If one of the command fails then the make is terminated at that point.

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();
```