1.8 tasking - Module

Running tools and functions within AIPS++
include "tasking.g"

Description

The `tasking` module contains tools that help with the execution of other tools and functions within AIPS++.

`toolmanager` allows execution of tools and functions from the Glish command line and also from Graphical User Interfaces.

`inputsmanager` allows saving and retrieval of inputs to the tools and functions.

`logger` is used for logging messages from tools and functions.

`scripter` allows execution of scripts built up from the toolmanager.

`clipboard` — allows cut-and-paste processing of Glish variables.

`types` defines types of variables in the tasking system.

`plugins` allows commands to be augmented by user-defined plug-in commands.

`asynceval` Allows asynchronous execution of an eval().

Tools

asynceval	Asynchronous eval
clipboard	clipboard object
inputsmanager	Tool for handling inputs for other tools
logger	Tool for logging
plugins	attach code to tools at runtime
scripter	scripter tool
servers	A tool for sending events between glish and precompiled executables
toolmanager	Tool for using AIPS++ tools and functions
types	Tool for defining types known to the tasking system

Functions

is_tool	Is this a tool
toolmanagertest	Test the tool manager
typestest	Test the types definition

1.8.1 toolmanager - Tool

Package utility
Module tasking

Tool for using AIPS++ tools and functions
include "toolmanager.g"

Description

`toolmanager` is the prime interface to the AIPS++ tools and functions. It provides graphical and command line interfaces for using tools and functions.

Using the toolmanager

The use of the `toolmanager` is documented in Getting Started in AIPS++.

Making a tool or function

If you have created an tool that you wish to show up in the toolmanager then the tool must obey the following rules:

- Be a record containing only functions
- Have a function *done* to terminate instances of the tool.
- AND:

  Either have a function `gui` that starts a GUI,

  Or have a meta information file containing a description of the GUI to be built automagically.

If the tool is to be classified correctly, then a help file must be in the system, and built into the documentation. Otherwise the tool will be shown under unclassified (Note: this is considered a serious wart and will be fixed soon). An example of a trival compliant tool is:

```
const foo := function() {
  public := [=];
  public.type := function() {return 'foo'};
  # A do-nothing-much function
  public.gui := function() {
    f:=frame(title='foo.gui');
    b:=button(f, 'Press here to continue');
    await b->press;
    note('foo.gui() finished');
```

```
      f:=F;
      public:=F;
      return T;
    };
    # Another do-nothing-much function
    public.another := function() {
      print "foo.another";
      return T;
    };
    # Another do-nothing-much cli
    public.anothercli := function() {
      print "foo.anothercli";
      return T;
    };
    # Another do-nothing-much gui
    public.anothergui := function() {
      f:=frame(title='foo.anothergui');
      b:=button(f, 'Press here to continue');
      await b->press;
      note('foo.anothergui() finished');
      f:=F;
      public:=F;
      return T;
    }
    return ref public;
  }
  # Add the required meta-information
  types.class('foo').function('ctor_foo');
```

An example of a more detailed meta information file follows:

```
pragma include once; # Include once only

include 'types.g'; # Required

# Name the include file that defines the tool
types.class('imager').includefile('imager.g');

# Constructors are preceeded by ctor_. Then follows the
# list of arguments.  Each argument is defined in this form:
#
# type(arg, default=unset, dir='in', checkeval=T)
#
# where type is defined in type.g and is one of:
#
# file|table|string|integer|float|double|record|boolean|vector_string|complex
```

```
#
# arg is the name of the argument
# default sets default values (may be an executable string)
# dir sets the direction of the argument
# checkeval=T means that a string value is eval'ed and checked,
# otherwise no such check is done.
#
# Define two constructors, one with just a table name, and the other
# with more arguments
types.function('ctor_imager').table('filename').
    function('ctor_imagertester').string('filename', '3C273XC1.ms').
    integer('size', 256).string('cell', '0.7arcsec').string('stokes', 'IV').
    string('coordinates', 'b1950');

# Functions
# Open with just a table name
types.function('open').table('thems');

# Plain close
types.function('close');

# Get the name back
types.function('name').string('return');

# Summary
types.function('summary');

# Set various image parameters. Note the use of the measures
# tool to set the phasecenter.
types.function('setimage').integer('nx',128).integer('ny',128).
    string('cellx','1arcsec').string('celly','1arcsec').
    string('stokes', 'I').
    boolean('doshift', F).
    record('phasecenter', 'dm.direction(\'b1950\', \'0d\', \'0d\')',checkeval=F).
    string('shiftx','0arcsec').string('shifty','0arcsec').
    string('mode', 'mfs').integer('nchan', 1).integer('start', 1).
    integer('step', 1).string('mstart', '0km/s').string('mstep', '0km/s').
    integer('spwid', 1).integer('fieldid', 1);
```

The toolmanager's own constructor is called during startup. Since only one toolmanager can be created, you should never have to call this constructor. The *only* toolmanager that exists has the conventional abbreviation `tm`.

Minimizing use of the toolmanager

If you wish, the toolmanager may be prevented from doing very much by putting the following in your .aipsrc file

```
toolmanager.fullgui: F
```

toolmanager.usegui - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Use GUIs if possible

Synopsis
usegui()

Description
Note that the GUI can only be used if Glish has access to a GUI!
An alternate function of specifying this is via the aipsrc mechanism: add the following to your .aipsrc file

```
toolmanager.gui.auto: T
```

Returns
Bool

---

toolmanager.usecli - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Use CLI

Synopsis
usecli()

Description
The CLI can always be used.

An alternate function of specifying this is via the aipsrc mechanism: add the following to your .aipsrc file

```
toolmanager.gui.auto: F
```

Returns
Bool

---

toolmanager.show - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Show one of: the toolmanager, a tool or function

Synopsis
show(what, prefergui)

Description
`show` can be used in four ways:

- To show the toolmanager:

  ```
  tm.show()
  ```

- Or, to show a GUI or CLI for a specific tool:

  ```
  tm.show('imgr')
  ```

- Or, to show a GUI or CLI for a specific global function:

  ```
  tm.show('imagerwizard')
  ```

- Or, to show a GUI or CLI for a tool constructor:

  ```
  tm.show('imager')
  ```

Arguments

what	Name of thing to be displayed Allowed:String Default: ”
prefergui	Use a GUI if possible Allowed:Bool Default: T

Returns
Bool

---

toolmanager.showconstructor - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Show the constructors of a tool

Synopsis
showconstructor(type, prefergui)

Description
Show the constructors of a type of tool. Before a type of tool *e.g.* `imager` can be used, it must be constructed into an actual tool *e.g.* `myimager`. There are usually a number of ways of constructing a tool. `showconstructor` allows one of these to be selected, the arguments set, and the constructor executed.

Arguments

type	Type of the tool to be constructed Allowed:String Default: ”
prefergui	Use a GUI if possible Allowed:Bool Default: T

Returns
Bool

Example

```
- tm.showconstructor('imager')
```

---

toolmanager.showtool - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Show a user interface for a tool

Synopsis
showtool(type, prefergui, forceauto)

Description
Show a user interface (GUI or CLI) for a tool. This interface will allows functions of the tool to be executed.

Arguments

type	Type of the tool to be constructed Allowed:String Default: ”
prefergui	Use a GUI if possible Allowed:Bool Default: T
forceauto	Force use of the auto GUI Allowed:Bool Default: F

Returns
Bool

Example

```
- tm.showtool('myimager')
```

---

toolmanager.showglobalfunction - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Show a user interface for a global function

Synopsis
showglobalfunction(ofn, prefergui, forceauto)

Description
Show a user interface (GUI or CLI) for a global function.

Arguments

ofn	Type of the tool to be constructed Allowed:String Default: ”
prefergui	Use a GUI if possible Allowed:Bool Default: T
forceauto	Force use of the auot GUI Allowed:Bool Default: F

Returns
Bool

Example

```
- tm.showglobalfunction('imagertest')
```

---

toolmanager.showitemmanager - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Show a user interface for a tool function

Synopsis
showitemmanager(ofn, prefergui, forceauto)

Description
Show a user interface (GUI or CLI) for an itemmanager.

Arguments

ofn	Type of the tool to be constructed Allowed:String Default: ”
prefergui	Use a GUI if possible Allowed:Bool Default: T
forceauto	Force use of the auot GUI Allowed:Bool Default: F

Returns
Bool

Example

```
- tm.showitemmanager('defaultmodelmanager')
```

---

toolmanager.where - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Find package, module and description of a type of tool

Synopsis
where(type)

Description
This returns a record containing the package *e.g.* `aips` or `synthesis`, module *e.g.* `images` or `utility`, and description of a type of tool.

Arguments

type	Type of tool Allowed:String Default: ”

Returns
Bool

Example

```
- print tm.where('image')
[package='aips', module='images', description='']
```

---

toolmanager.istool - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Is this a tool?

Synopsis
istool(tool)

Description
Is the named Glish variable a tool? For example, `imager` is a type of tool that can be created, whereas `myimager` is an actual tool.

Arguments

tool	Name of candidate Allowed:String Default: ”

Returns
Bool

Example

```
- tm.istool('imager')
F
- tm.istool('myimager')
T
```

---

toolmanager.istooltype - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Is this a tool type?

Synopsis
istooltype(tool)

Description
Is the named Glish variable a tool? For example, `imager` is a type of tool that can be created, whereas `myimager` is an actual tool.

Arguments

tool	Name of candidate Allowed:String Default: ”

Returns
Bool

Example

```
- tm.istooltype('imager')
T
- tm.istooltype('myimager')
F
```

---

toolmanager.tooltype - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Return tool type

Synopsis
tooltype(tool)

Description
Show the type of the named Glish variable *e.g.* `imager`, `ms`, `regionmanager`.

Arguments

tool	Name of candidate Allowed:String Default: ”

Returns
Bool

Example

```
- tm.tooltype('myimager')
imager
- tm.tooltype('drm')
regionmanager
```

---

toolmanager.registertool - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Register a tool

Synopsis
registertool(tool, status, description)

Description
Register a tool with the tool manager. This is performed by the toolmanager itself but may also be performed otherwise.

Arguments

tool	Name of tool Allowed:String Default: None
status	Status of tool Allowed:String Default: '-'
description	Description of tool Allowed:String Default: unset

Returns
Bool

Example

```
- tm.registertool('myimager')
T
```

---

toolmanager.settoolstatus - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Set the tool status

Synopsis

settoolstatus(tool, status)

Description
The status of a tool is informative only. Recommended use is the name of an executing function or the string 'Idle'.

Arguments

tool	Name of tool Allowed:String Default: None
status	Status of tool Allowed:String Default: '-'

Returns
Bool

Example

```
- tm.settoolstatus('myimager', 'clean')
T
- tm.settoolstatus('myimager', 'Idle')
T
```

---

toolmanager.deletetool - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Delete a tool

Synopsis
deletetool(tool)

Description
Delete a tool: first unregister the tool with the tool manager, and then delete

the tool. This is performed by the toolmanager itself but may also be performed otherwise.

Arguments

tool	Name of tool Allowed:String Default: None

Returns
Bool

Example

```
- tm.deletetool('myimager')
T
```

---

toolmanager.killtool - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Kill a tool

Synopsis
killtool(tool)

Description
Kill the server associated with a given tool. Note that multiple tools may be present in one server so that if you have to kill a given tool, you may end up killing its cousins as well. This is a current limitation of the tasking system in AIPS++ that will probably be fixed in version 1.5 or 1.6.

Arguments

tool	Name of tool Allowed:String Default: None

Returns
Bool

Example

```
- tm.killtool('myimager')
T
```

---

toolmanager.unregistertool - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Unregister a tool

Synopsis
unregistertool(tool)

Description
Unregister a tool with the tool manager. This is performed by the toolmanager itself but may also be performed otherwise.

Arguments

tool	Name of tool Allowed:String Default: None

Returns
Bool

Example

```
- tm.unregistertool('myimager')
T
```

---

toolmanager.isregistered - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Is this tool registered?

Synopsis
isregistered(tool)

Description
Is this tool registered?

Arguments

tool	Name of tool Allowed:String Default: None

Returns
Bool

Example

```
- tm.isregistered('myimager')
T
```

---

toolmanager.toolinfo - Function

Package	**utility**
Module	**tasking**
Tool	**toolmanager**

Show information about a registered tool

Synopsis
toolinfo(tool)

Description
Show information about a registered tool.

Arguments

tool	Name of tool Allowed:String Default: None

Returns
Record

Example

```
- print tm.toolinfo('myimager')
[type='imager', status='Idle', description='Synthesis imaging and deconvolution']
```

1.8.2 is_tool - Function
Package utility
Module tasking

Is this a tool

Synopsis
is_tool(tool)

Description
Check to see if the Glish variable is a valid tool. To be a tool, a variable must be a record, with a field named `type` that is a function returning a string.

Arguments

tool	Variable to be tested Allowed:Glish variable Default: None

Returns
Bool

1.8.3 toolmanagertest - Function

Package utility
Module tasking

Test the tool manager

Synopsis
toolmanagertest()

Description
Run a set of tests on the toolmanager.

Returns
Number of failed tests (Int)

Example

```
- toolmanagertest()
```

1.8.4 inputsmanager - Tool

Package utility
Module tasking

Tool for handling inputs for other tools

Description
`inputsmanager` is a tool for handling the inputs (function arguments) for AIPS++ tools. It provides command line interfaces for saving and retrieving all the inputs to a function. It is primarily used by toolmanager to save and restore inputs, but it can also be used directly from the glish command line. Note that inputs are not automatically saved in this manner when a function

is invoked directly from the glish command line, although the command line history buffer can be used in that case.
The values of the inputs are stored as a record by `inputsmanager` for each tool function. For example, to retrieve the function arguments used last time that imager.setimage was run from the toolmanager:

```
- include 'inputsmanager.g'
- inputs.show()
Inputs:

  Row                 Time   Keyword
-------------------------------------
    1   2000/02/18/13:54:24  lastsave
T
- inputs.getvalues('imager', 'setimage')
[nx=300, ny=300, cellx=[value=1.2, unit=arcsec], celly=[value=1.2,
unit=arcsec], stokes=IQUV, doshift=T, phasecenter=[m0=[value=0,
unit=deg], m1=[value=90, unit=deg], type=direction, refer=JUPITER],
shiftx=[value=0, unit=arcsec], shifty=[value=0, unit=arcsec],
mode=mfs, nchan=1, start=1, step=1, mstart=[value=0, unit=km/s],
mstep=[value=0, unit=km/s], spwid=1, fieldid=1, facets=1]
```

The current inputs may be saved to a specific slot named by the keyword chosen from the toolmanager. The default is 'lastsave'.
The values of the inputs are stored permanently in an AIPS++ table. The name of the file is *aips++.inputs.table* and thus a separate table will be created for every directory in which AIPS++ is run. This default file location may be overridden using an `aipsrc` variable. For example, to put consolidateall inputs in one place:

```
inputsmanager.file: $HOME/aips++.inputs.table
```

Note that inputs are only saved from the toolmanager, and that the values are written to a disk-based table only when a save function is executed. A save is performed automatically on exit from glish. This means that you should always exit normally from glish if you wish the current inputs to be saved.

Tools

asynceval	Asynchronous eval
clipboard	clipboard object
inputsmanager	Tool for handling inputs for other tools
logger	Tool for logging
plugins	attach code to tools at runtime
scripter	scripter tool
servers	A tool for sending events between glish and precompiled executables
toolmanager	Tool for using AIPS++ tools and functions
types	Tool for defining types known to the tasking system

inputsmanager - Constructor

Package	**utility**
Module	**tasking**
Tool	**inputsmanager**

Construct inputsmanager tool

Synopsis
inputsmanager()

Description
This constructor is called on inclusion of inputsmanager.g. Since only one inputsmanager can be created, you should never have to call this constructor. The *only* inputsmanager that exists has the conventional name `inputs`.

---

inputsmanager.getvalues - Function

Package	**utility**
Module	**tasking**
Tool	**inputsmanager**

Get inputs for a tool and function

Synopsis
getvalues(tool, method, keyword)

Description
Get the inputs for a given tool, function, and keyword. This returns a record containing the argument name and value.

Arguments

tool	Name of tool Allowed:String Default: None
method	Name of function Allowed:String Default: None
keyword	Name of keyword Allowed:String Default: 'lastsave'

Returns
Record of values

Example

```
- include 'inputsmanager.g'
- inputs.show()
Inputs:

  Row                  Time   Keyword
------------------------------------
    1   2000/02/18/13:54:24  lastsave
T
- inputs.getvalues('imager', 'setimage')
[nx=300, ny=300, cellx=[value=1.2, unit=arcsec], celly=[value=1.2,
unit=arcsec], stokes=IQUV, doshift=T, phasecenter=[m0=[value=0,
unit=deg], m1=[value=90, unit=deg], type=direction, refer=JUPITER],
shiftx=[value=0, unit=arcsec], shifty=[value=0, unit=arcsec],
mode=mfs, nchan=1, start=1, step=1, mstart=[value=0, unit=km/s],
mstep=[value=0, unit=km/s], spwid=1, fieldid=1, facets=1]
```

---

inputsmanager.savevalues - Function

Package	**utility**
Module	**tasking**
Tool	**inputsmanager**

Save inputs for a tool and function

Synopsis

savevalues(tool, method, values, keyword, dosave)

Description
Save the inputs for a given tool, function, and keyword. The record must contain the argument names and values.

Arguments

tool	Name of tool Allowed:String Default: None
method	Name of function Allowed:String Default: None
values	Record of inputs Allowed:Record Default: [=]
keyword	Name of keyword Allowed:String Default: 'lastsave'
dosave	Force a save to disk-based table Allowed:Bool Default: F

Returns
Bool

Example

```
- include 'inputsmanager.g'
- inputs.show()
Inputs:

  Row                  Time   Keyword
-------------------------------------
    1   2000/02/18/13:54:24  lastsave
T
- rec := inputs.getvalues('imager', 'setimage')
- print rec
[nx=300, ny=300, cellx=[value=1.2, unit=arcsec], celly=[value=1.2,
unit=arcsec], stokes=IQUV, doshift=T, phasecenter=[m0=[value=0,
```

```
unit=deg], m1=[value=90, unit=deg], type=direction, refer=JUPITER],
shiftx=[value=0, unit=arcsec], shifty=[value=0, unit=arcsec],
mode=mfs, nchan=1, start=1, step=1, mstart=[value=0, unit=km/s],
mstep=[value=0, unit=km/s], spwid=1, fieldid=1, facets=1]
- rec.stokes:='IV';
- inputs.savevalues('imager', 'setimage', rec, 'IV only', dosave=F)
- inputs.show()
Inputs:

  Row                 Time   Keyword
-------------------------------------
    1   2000/02/18/14:04:25  lastsave
    2   2000/02/18/14:04:25   IV only
T
```

---

inputsmanager.show - Function

Package	**utility**
Module	**tasking**
Tool	**inputsmanager**

Show all sets of available inputs

Synopsis
show(keyword)

Description
Shows all the available inputs on the screen.

Arguments

keyword	Name of keyword Allowed:String Default: 'default'

Returns
Bool

Example

```
- include 'inputsmanager.g'
- inputs.show()

  Row                  Time   Keyword
------------------------------------
    1   2000/02/18/14:04:25  lastsave
    2   2000/02/18/14:04:25   IV only
T
```

---

inputsmanager.list - Function

Package	**utility**
Module	**tasking**
Tool	**inputsmanager**

List all sets of available inputs

Synopsis
list(keyword)

Description
Returns a record of all available inputs. Use this function with caution: the returned record is very large.

Arguments

keyword	Name of keyword Allowed:String Default: 'default'

Returns
Record

Example

```
- include 'inputsmanager.g'
- field_names(inputs.list())
version lastsave mysave1 mysave2
```

---

inputsmanager.delete - Function

Package	**utility**
Module	**tasking**
Tool	**inputsmanager**

Delete rows from the table

Synopsis
delete(rows)

Description
Deletes numbered rows from the inputs table.

Arguments

rows	Vector of row numbers Allowed:Vector of Ints Default: None

Returns
Record

Example

```
- include 'inputsmanager.g'
- inputs.show()
Inputs:

  Row                  Time   Keyword
-------------------------------------
    1   2000/02/18/14:04:25  lastsave
    2   2000/02/18/14:04:25   IV only
T
- inputs.delete(2)
NORMAL: Successfully deleted row 2
T
- inputs.show()
Inputs:
```

```
  Row                  Time   Keyword
-------------------------------------
    1   2000/02/18/14:04:25  lastsave
T
```

---

inputsmanager.deletetable - Function

Package	**utility**
Module	**tasking**
Tool	**inputsmanager**

Delete the inputs table

Synopsis
deletetable()

Description
Deletes the inputs table.

Returns
Bool

Example

```
- include 'inputsmanager.g'
- inputs.show()
Inputs:

  Row                  Time   Keyword
-------------------------------------
    1   2000/02/18/14:04:25  lastsave
T
- inputs.deletetable()
NORMAL: deleted table aips++.inputs.table
F
```

---

inputsmanager.get - Function

Package	**utility**
Module	**tasking**
Tool	**inputsmanager**

Get all inputs for a given keyword

Synopsis
get(keyword)

Description
Get the inputs for a given keyword from the table. The values returned by `getvalue` are then drawn from this set.

Arguments

keyword	Name of keyword Allowed:String Default: 'lastsave'

Returns
Record of values

Example

```
- include 'inputsmanager.g'
- inputs.get('lastsave')
```

---

inputsmanager.save - Function

Package	**utility**
Module	**tasking**
Tool	**inputsmanager**

Save all inputs for a given keyword

Synopsis
save(keyword)

Description
Save the current inputs to a given keyword in the table.

Arguments

keyword	Name of keyword Allowed:String Default: 'lastsave'	

Returns
Bool

Example

```
- include 'inputsmanager.g'
- inputs.save('lastsave')
```

1.8.5 clipboard - Tool

Package utility
Module tasking

clipboard object

Description
`clipboard` is a simple tool for saving and restoring glish variables. It's mainly of use for interacting with GUIs. Values can be copied to and from entry widgets. For example, to transfer a direction measure from the measures direction GUI to an entry in a tool, first use the Copy button on the direction GUI to place the value in the clipboard, and then use the Paste button in the Wrench menu of the direction entry to copy it into the entry.
A paste into the Glish CLI can be performed using the paste function. Similarly a copy from the Glish CLI can be done using the copy. For example, to multiply the value in the clipboard by the $\sqrt{2}$, do:

```
include 'clipboard.g'
a: = dcb.paste()
a *:= sqrt(2.0)
dcb.copy(a);
```

This clipboard can only contain one value at a time, so you must copy and then paste one value at a time. Note that any glish variable can be placed in the clipboard. In the future, we may develop multiple entry clipboards if these are thought to be useful.

clipboard - Constructor

Package	**utility**
Module	**tasking**
Tool	**clipboard**

Construct clipboard tool

Synopsis
clipboard()

Description
This constructor is called during startup. The *default* clipboard has the conventional name `dcb`. Normally only this clipboard is used.

---

clipboard.copy - Function

Package	**utility**
Module	**tasking**
Tool	**clipboard**

Copy something to the clipboard

Synopsis
copy(parameter)

Description
Copy any glish variable to the clipboard.

Arguments

parameter	Name of Glish parameter Allowed:Any glish variable Default: None

Returns
Bool

Example

```
include 'clipboard.g'
a := sqrt(2.0)
dcb.copy(a);
```

---

clipboard.paste - Function

Package	**utility**
Module	**tasking**
Tool	**clipboard**

Paste the clipboard value

Synopsis
paste()

Description
The entry currently in the clipboard is returned.

Returns
Glish variable

Example

```
include 'clipboard.g'
print dcb.paste();
```

1.8.6 scripter - Tool

Package	**utility**
Module	**tasking**

scripter tool

Description
The scripter is a tool for logging commands to a file to be run at a later time. It is intended to be used for storing commands that can then be saved to a file or executed directly.

toolmanager can be directed, via its Options menu, to send commands to the scripter. These commands will correspond to any subsequent actions invoked through its gui.

imagerwizard sends glish commands to the scripter. These commands can be saved to a file and executed to reproduce exactly the operations performed by the wizard.

Constructors

scripter	Construct scripter tool

Functions

getcommands	Returns a record containing the commands
gui	Display logged commands in a text pane
load	Load command(s) from a file.
log	Log one or more commands
reset	Removes all the logged commands
save	Save the logged command(s) to a file.

scripter - Constructor

Package	**utility**
Module	**tasking**
Tool	**scripter**

Construct scripter tool

Synopsis
scripter(logfile, run, guititle, widgetset)

Description
This constructs a scripter tool. The *default* scripter is abbreivated `ds`. Only one scripter is usually needed so this constructor is seldom called.

Arguments

logfile	File containing logged commands Allowed:String Default: scripter.log_PID
run	Flag to determine whether commands are "eval'ed" Allowed:Boolean Default: F
guititle	Title for the scripter Allowed:String Default: 'Scripter gui (AIPS++)'
widgetset	Widget set to be used Allowed:Any valid widgetset Default: dws

Returns
tool

---

scripter.gui - Function

Package	**utility**
Module	**tasking**
Tool	**scripter**

Display logged commands in a text pane

Synopsis
gui()

Description
This will place the current commands in a text window for editing, saving, and submission to glish or a batch queue.
The scripter gui has an autosave feature which is turned on by default `scripter.autosave`. You may disable the autosave feature by setting the `.aipsrc` variable `scripter.autosave` to F or off. There are two additional `.aipsrc` variables that let you control the autosave feature, `scripter.autosavetime` and `scripter.autosavefile`. The default auto-save time is 30s and the default auto-save filename is `.scripter_autosave`. The auto-save file will stick around unless the scripter-gui edits are saved explicitly. If the scripter-gui detects an auto-save file it will prompt, whether to load it or not.

The buttons act as follows:

Submit The script is executed using the command shown above. The default command `unset DISPLAY;glish -l` causes the scripter to be run in a separate instance of glish with no graphical output.

Save The command is saved to the named file. Use the File-¿Save As menu to save with a different name.

Clear Clears the scripter of all commands.

Dismiss Dismisses the scripter. It may be recalled using the `gui` function.

The submit command may be overridden using the `.aipsrc` variable `scripter.submitcommand`.

Returns
Bool

---

scripter.log - Function

Package	**utility**
Module	**tasking**
Tool	**scripter**

Log one or more commands

Synopsis
log(command, run)

Description
Log one or more command(s). The commands are stored in the scripter. The commands will be executed directly if the flag run is set to T.

Arguments

command	Command(s) to be logged Allowed:String
run	Flag to determine if command(s) are run immediately Allowed:Boolean Default: F

Returns
Bool

Example

```
include 'scripter.g'
ds.log('myimager:=imager(\'3C273XC1.ms\')');
ds.log('myimager.setimage(nx=300,ny=300,cellx=\'0.7arcsec\'',celly=\'0.7arcsec\')')
ds.log('myimager.clean()');
```

---

scripter.save - Function

Package	**utility**
Module	**tasking**
Tool	**scripter**

Save the logged command(s) to a file.

Synopsis
save(filename)

Description
Saves the command(s) in the internal logged commands record to a file.

Arguments

filename	File to save commands Allowed:Any valid file name or F Default: F

Returns
Bool

Example

```
include 'scripter.g'
ds.log('myimager:=imager(\'3C273XC1.ms\')');
```

```
ds.log('myimager.setimage(nx=300,ny=300,cellx=\'0.7arcsec\'',celly=\'0.7arcsec\')')
ds.log('myimager.clean()');
ds.save('imager_scripter.g');
```

---

scripter.load - Function

Package	**utility**
Module	**tasking**
Tool	**scripter**

Load command(s) from a file.

Synopsis
load(filename, append, run)

Description
Load commands from a file to the scripter

Arguments

filename	File from which to load commands Allowed:Any valid file name
append	Flag to determine whether to append commands to exsisting command list Allowed:Boolean Default: F
run	Flag to determine whether commands are executed Allowed:Boolean Default: F

Returns
Bool

Example

```
include 'scripter.g'
ds.load('imager_scripter.g', run=T);
```

---

scripter.getcommands - Function

Package	**utility**
Module	**tasking**
Tool	**scripter**

Returns a record containing the commands

Synopsis
getcommands()

Description
Return the commands currently in the scripter in a record.

Returns
Record

Example

```
include 'scripter.g'
ds.load('imager_scripter.g');
print ds.getcommands()
```

---

scripter.reset - Function

Package	**utility**
Module	**tasking**
Tool	**scripter**

Removes all the logged commands

Synopsis
reset()

Description
Removes all current commands. The scripter is then empty.

Returns
Bool

Example

```
include 'scripter.g'
ds.load('imager_scripter.g');
print ds.getcommands()
ds.reset();
```

1.8.7 logger - Tool

Package utility
Module tasking

Tool for logging
include "logger.g"

Description

The `logger` is the tool managing messages from scripts, clients, etc.. In general it writes the messages into a log file and shows them on the screen in a text window or a GUI.
Log messages are produced by binary executable programs and Glish scripts. You may also create log messages yourself through the "note" member function of logger. You will almost always be interacting with `defaultlogger`, which is also known as `dl` to save you some typing.
Note: Users should use the global note function to create log messages. This has sufficient flexibility to do everything that the "log" and "note" member function of logger can do. note checks for the existance of the defaultlogger and sends messages to the screen if the defaultlogger is not present.
The log messages are sent both to your screen (normally to a GUI, but you can have them come out in your terminal window instead if you prefer) and to a file (an AIPS++table). This file will continue to accumulate messages until you purge them.
By default the log file is `~/aips++/aips++.log`, however you may override this default location by setting some variables in your `~/.aipsrc` file as described below.
The first column is the time that the message was produced. The default is to display the GMT times. If you want to see local times instead, you can set the

time zone offset with another `.aipsrc` variable.
The second panel gives the priority of the message. Messages may presently be of `DEBUGGING`, `NORMAL`, `WARN`, and `SEVERE` priority. When a `SEVERE` level message is shown a pop-up window with a dismiss button also comes up to make sure you don't miss it. In the log window `WARN` and `SEVERE` level messages are highlighted and shown in red instead of green to make them stand out.
The third panel contains the (hopefully!) informative messages. Note that messages may be multiple lines, the other column values are only shown at the start of the message, not for every line in it.
The final panel contains the location where the message came from. This may be of little interest to you.
You can control the panels (size, order, and which to see) by `.aipsrc` variables and by the `Options` menu. They can also be used to control whether a few buttons will be displayed at the bottom. It makes it possible to make the log window as big or small as you like.
It is possible to execute a query on the log window; i.e. to select or sort messages. They will be displayed in a separate log window which is similar to the main window, but also shows the select and sort command used in the query. One can execute another query on that new window, etc..

The `File` menu contains various entries to deal with log messages.

- Query makes it possible to query the log messages as described above. It uses taqlwidget to let you form the select and/or sort command.

- Purge removes old messages from the log file. The last `n` messages will be kept which can be set using a slider. Alternatively a TaQL expression can be given to remove the matching messages. You should purge your log file from time to time.
  This entry is only available in the main log window.

- Refresh refreshes a window resulting from a query. New messages are only added to the main log window. It is possible to add them to a sub window too by pressing `refresh`. It simply re-executes the query.
  The refresh button at the bottom-left of the window does the same.

- Script creates a glish script from logged glish commands. It gets all messages starting with the special prefix and stores them in a file after removing the prefix. That file can be saved and/or be executed. A slider lets you specify the maximum number of commands to be written (counted from the end).

- Print writes the messages into a file and optionally prints them. A slider lets you specify the maximum number of messages to be written (counted from the end). It is also possible to tell which columns have to be printed.

- Dismiss to dismiss this window and all its children.
  The dismiss button at the bottom-right of the window does the same.

Note that the commands above are always executed on the log file and not on the messages in the window. For example, a print is done in a window resulting from a query. This may print more messages than there are in the window, because in the mean time new messages may have been added to the log file.

A dismiss of a window also dismisses its child windows resulting from queries. If you dismiss the main log window, log messages will come out on your terminal instead of the GUI. When you get tired of this, you can type `dl.gui()` at the Glish command prompt to restart the GUI.

The only logger functions you might perhaps call are `loginput()` and `logoutput()` to control what is logged from Glish.

Aipsrc Variables

user.aipsdir	AIPS++info directory Allowed: Any writable directory. If it doesn't exist it will be created Default: ~/aips++
tzoffset	Timezone offset in hours Allowed: e.g., Socorro is -7 Default: 0
logger.file	Log file name (no path!) Allowed: any (none means temporary log file is used) Default: aips++.log
logger.default	Where to write log messages Allowed: gui or screen Default: gui
logger.showbuttons	Show buttons at bottom of the GUI? Allowed: Bool Default: T
logger.glish	What to log from Glish Allowed: input, output, both, or none Default: input
logger.guishow	Which panels to show in GUI and in which order Allowed: any of time,priority,message,origin Default: time,priority,message
logger.height	Height of GUI panels Allowed: Integer Default: 8
logger.timewidth	Width of time panel (if shown) Allowed: Integer Default: 20
logger.prioritywidth	Width of priority panel (if shown) Allowed: Integer Default: 6
logger.messagewidth	Width of message panel (if shown) Allowed: Integer Default: 75
logger.originwidth	Width of origin panel (if shown) Allowed: Integer Default: 25
logger.autoscroll	Scroll GUI to the end after appending a message Allowed: Bool Default: T
logger.background	Normal background color of the GUI Allowed: String Default: xing

Constructors

logger   Construct logger tool

Functions

attach	Attach to another log table
brief	Write fewer fields in log messages
getformatted	Get a formatted representation of the log file
gui	Use the GUI
log	Add a message to the logging system
loginput	Log input to Glish?
logoutput	Log output from Glish?
nmessages	How many log messages are in the file
note	Add a note to the logging messages
printtofile	Write some or all of the log file to a text file
purge	Delete some or all messages from the log file.
screen	Use the screen for logging messages
verbose	Write more fields in log messages

logger - Constructor

Package	**utility**
Module	**tasking**
Tool	**logger**

Construct logger tool

Synopsis
logger(use_gui)

Description
The GUI will be used, unless you have your aipsrc variable `logger.default` set to something besides "gui", or X-windows is not enabled for your login session (e.g., the DISPLAY environment variable is not set).

Arguments

use_gui	in	Use GUI for logging messages Allowed:Bool Default: global_use_gui

Returns
tool

logger.gui - Function

Package	**utility**
Module	**tasking**
Tool	**logger**

Use the GUI

Synopsis
gui()

Description
An alternate method of specifying the display of logging messages is via the aipsrc mechanism: add the following to your .aipsrc file

```
logger.default: gui
```

Returns
Bool

---

logger.screen - Function

Package	**utility**
Module	**tasking**
Tool	**logger**

Use the screen for logging messages

Synopsis
screen()

Description
An alternate method of specifying the display of logging messages is via the aipsrc mechanism: add the following to your .aipsrc file

```
logger.default: screen
```

Actually, anything besides “gui” will send log messages to the screen.

Returns
Bool

---

logger.attach - Function

Package	**utility**
Module	**tasking**
Tool	**logger**

Attach to another log table

Synopsis
attach(logfile)

Description
Attach the logger to another log table. If the argument is not a string, the logger is attached to the default log table. If the argument is an empty string, it is attached to a temporary log table.

Arguments

logfile	in	Name of new log table Allowed:String Default: unset

Returns
Bool

Example

```
  dl.attach('my.log');
# T
  dl.attach();
# T
```

The defaultlogger is temporarily attached to table my.log. Thereafter it is reattached to the default log table.

Returns
Bool

---

logger.note - Function

Package	**utility**
Module	**tasking**
Tool	**logger**

Add a note to the logging messages

Synopsis
note(..., origin, postglobally, postlocally, postcli)

Description
Add a note to the logging messages with optional origin

Arguments

...	in	Strings to be logged Allowed:Any strings Default: Empty string
origin	in	Origin of note Allowed:String Default: 'logger::note()'
postglobally	in	Write into log table? Allowed:Bool Default: T
postlocally	in	Write to screen? Allowed:Bool Default: T
postcli	in	Show message in CLI window too? Allowed:Bool Default: F

Returns
Bool

Example

```
include 'logger.g';
dl.note('Gone to lunch', 'back in about 1 hour')
```

---

logger.log - Function

Package	**utility**
Module	**tasking**
Tool	**logger**

Add a message to the logging system

Synopsis

log(timeString, priority, message, origin, postglobally, postlocally, postcli)

Description

Add a message to the logging system. This is the preferred mechanism for Glish scripts to add messages to the logging message file. Note that although priority=SEVERE is allowed, we prefer that in general, the Glish fail mechanism is used for irrecoverable conditions.

Arguments

timeString	in	time Allowed:String Default: Current time
priority	in	Priority of message Allowed:NORMAL—DEBUGGING—WARN—SEVERE
message	in	Message Allowed:String
origin	in	source of the message Allowed:string Default: [=]
postglobally	in	Write into log table? Allowed:Bool Default: T
postlocally	in	Write to screen? Allowed:Bool Default: T
postcli	in	Show message in CLI window too? Allowed:Bool Default: F

Returns

Bool

Example

```
include 'logger.g';
dl.log('', 'SEVERE', 'File does not exist', 'file.open');
```

---

logger.loginput - Function

Package	**utility**
Module	**tasking**
Tool	**logger**

Log input to Glish?

Synopsis

loginput(dolog)

Description

This function turns on (or off) input logging from Glish. That is, if it is called with `T` things that you type into Glish will appear as log messages.

Arguments

dolog	in	Do logging? Allowed:Bool Default: T

Returns

Bool - the current state

---

logger.logoutput - Function

Package	**utility**
Module	**tasking**
Tool	**logger**

Log output from Glish?

Synopsis
logoutput(dolog)

Description
This function turns on (or off) input output from Glish. That is, if it is called with `T` things that Glish prints on the screen will also be entered as log messages (with comment signs in front so that they can be rerun).

Arguments

dolog	in	Do logging? Allowed:Bool Default: T

Returns
Bool - the current state

---

logger.nmessages - Function

Package	**utility**
Module	**tasking**
Tool	**logger**

How many log messages are in the file

Synopsis
nmessages()

Description
This function (no arguments) returns the number of messages stored in the log file. If you are using the GUI you probably don't need to know this, however if you are calling any of `purge()`, `printtofile()`, or `getformatted()` by hand for some reason you might want to call this function

Returns
Integer

---

logger.purge - Function

Package	**utility**
Module	**tasking**
Tool	**logger**

Delete some or all messages from the log file.

Synopsis
purge(keeplast, expr)

Description
While you will normally want to delete log messages using the GUI, you can do it "by hand" from this function, or you might want to use this function to manage messages in your own scripts.
If an expression is given all matching messages will be removed. Otherwise the last N messages will be kept.

Arguments

keeplast	in	How many messages to keep (from end) Allowed:Int Default: 500
expr	in	Delete messages matching this TaQL expression Allowed:String Default: "

Returns
Bool - function ok

---

logger.printtofile - Function

Package	**utility**
Module	**tasking**
Tool	**logger**

Write some or all of the log file to a text file

Synopsis

printtofile(num, filename, colwidth, expr, ascommand)

Description

Again, while you will normally invoke printing from the GUI, you can make a text representation of the log file using this function. It is your responsiblity to actually print the file and delete it when done. The printer tool can do this for you.
The `filename`argument can be given to specify the name of the file to be created. A blank name means that a file `/tmp/aips_NNN` will be created, where NNN is a number to make the name unique.
The `colwidth` argument lets you define if and how wide each column has to be printed. 0 means do not print. -1 means use default width. The order of the columns in this 4-element vector is `time, priority, message, origin`.
It is possible to filter and/or sort the log messages by giving the `WHERE` and `ORDERBY` parts of a TaQL expression in the `expr` argument. This has to be a vector of 0, 1, or 2 strings. If the first string is present, it has to be the `WHERE` expression. If the second string is present, it has to be the `ORDERBY` sort list.
The `ascommand` argument can be used to tell that only logged glish commands should be written (after removing the special prefix). This is, for example, useful to create a glish script from interactively given commands.

Arguments

num	in	How many messages to write (from end) Allowed:Int Default: -1 (all)
filename	in	Name of the output file Allowed:String Default: ”
colwidth	in	Width of each column (0 = do not print; -1 = use default) Allowed:Int vector Default: [-1,-1,-1,-1]
expr	in	select and/or sort command Allowed:String vector Default: ”
ascommand	in	Only write logged glish commands Allowed:Int Default: F)

Returns

String - file name

logger.getformatted - Function

Package	**utility**
Module	**tasking**
Tool	**logger**

Get a formatted representation of the log file

Synopsis
getformatted(time, priority, messages, origin, howmany, expr, concat)

Description
This is principally intended to be used by the GUI to refresh itself from the log file (e.g. at startup).

Arguments

time	out	Allowed:String vector
priority	out	Allowed:String vector
messages	out	Allowed:String vector
origin	out	Allowed:String vector
howmany	in	How many to get (from end) Allowed:Int Default: -1 (all)
expr	in	select and/or sort command (see printtofile) Allowed:String vector Default: ”
concat	in	Concatenate consecutive messages with the same priority into one String? Allowed:Bool Default: F

Returns
Bool - success

---

logger.verbose - Function

Package utility

Module	**tasking**
Tool	**logger**

Write more fields in log messages

Synopsis
verbose(show_time, show_priority, show_origin)

Description
Verbose by default prints all fields in log messages (time, priority, message, origin).

Arguments

show_time	in	Allowed:boolean Default: T
show_priority	in	Allowed:boolean Default: T
show_origin	in	Allowed:boolean Default: T

Returns
Bool

---

logger.brief - Function

Package	**utility**
Module	**tasking**
Tool	**logger**

Write fewer fields in log messages

Synopsis
brief(show_time, show_priority, show_origin)

Description
Brief by default prints only the message, and suppresses time, priority, and origin.

Arguments

show_time	in	Allowed: boolean Default: F
show_priority	in	Allowed: boolean Default: F
show_origin	in	Allowed: boolean Default: F

Returns
Bool

1.8.8 types - Tool

Package utility
Module tasking

Tool for defining types known to the tasking system
include "types.g"

Description

`types` is the means for defining the user interface (i.e., the functions and arguments) of other AIPS++tools.
An AIPS++tool or function usually takes various types of arguments. Glish itself is type-less and doesn't require that the type of a variable be defined explicitly or fixed, which has advantages and disadvantages. In the tasking system, it is important to impose some typing of variables so that, for example, the toolmanager knows whether a particular argument is a scalar or a measure or a file name. The `types` tool provides the mechanism for defining the type to be used in the `toolmanager`'s user interface.
This works as follows:

- The developer specifies the interface in a meta information file.

- In the AIPS++startup, the `types` tool reads these meta information files, and stores the results in a cached file.
- The functions of `types` may be used to recall information on the interface.
- User interfaces for tools and functions are constructed by the toolmanager using the meta information and passed to the autogui or autocli.
- The user interface does parsing as appropiate.
- If the tool is linked to C++, arguments are passed appropriately.

The net result for the developer is that a user interface for any tool or function may be generated easily merely by provision of a meta information file.
An example of a meta information file follows. This defines the interface for a tool called imager and hence should be located in a file called imager_meta.g.

```
pragma include once; # Include once only

include 'types.g'; # Required

#
# Name the include file that defines the tool
types.class('imager').includefile('imager.g');
#
# Constructor names are preceeded by ctor_. Then follows the
# list of arguments.  Each argument is defined with a
# function call of this form:
#
# type(arg, default=unset, dir='in', checkeval=T)
#
# where type is defined in type.g and is one of:
#
# file|table|string|integer|float|double|record|boolean|vector_string|complex
#
# arg is the name of the argument
# default sets default values (may be an executable string)
# dir sets the direction of the argument
# checkeval=T means that a string value is eval'ed and checked,
# otherwise no such check is done.
#
# The functions may be assigned to groups using the
# group('mygroup') function. A group() call resets the
# group to the default, which is 'basic'.
#
# Define two constructors, one with just a table name, and the other
```

```
# with more arguments
types.method('ctor_imager').table('filename').
    method('ctor_imagertester').string('filename', '3C273XC1.ms').
    integer('size', 256).string('cell', '0.7arcsec').string('stokes', 'IV').
    string('coordinates', 'b1950');

# Methods
# Open with just a table name
types.group('basic').method('open').table('thems');

# Plain close
types.method('close');

# Get the name back
types.method('name').string('return');

# Summary
types.method('summary');

# Set various image parameters. Note the use of the measures
# tool to set the phasecenter.
types.group('setup').method('setimage').integer('nx',128).integer('ny',128).
    string('cellx','1arcsec').string('celly','1arcsec').
    string('stokes', 'I').
    boolean('doshift', F).
    record('phasecenter', 'dm.direction(\'b1950\', \'0d\', \'0d\')',checkeval=F).
    string('shiftx','0arcsec').string('shifty','0arcsec').
    string('mode', 'mfs').integer('nchan', 1).integer('start', 1).
    integer('step', 1).string('mstart', '0km/s').string('mstep', '0km/s').
    integer('spwid', 1).integer('fieldid', 1);
```

Constructors

Functions

addmeta	Copy meta information to a class
all	Show all classes
antennas	Define a list of antennas
attachhelp	Attach help from the help system
baselines	Define a list of baselines
boolean	Define a boolean
check	Choose one or more from a list of strings
choice	Choose one from a list of strings
class	Define a class or type of tool
classes	Return names of the classes

complex	Define a complex
direction	Define a direction measure
directory	Define a directory
double	Define a double
epoch	Define an epoch measure
exists	Does the class, method, arg exist?
fields	Define a list of field ids
file	Define a file
float	Define a float
getincludefile	Get the include file for a class
group	Define the group name for a function
includefile	Specify the include file for a class
includemeta	Include all meta information files
integer	Define an integer
is_tool	Is this a tool?
list	Define a list of other types
measure	Define a measure
measurecodes	Define the reference code for a measure
meta	Return meta information for a type of tool
method	Define a method or function for a class
msselect	Define a taql string for a MeasurementSet
position	Define a position measure
quantity	Define a quantity
record	Define a record
region	Define a region
spectralwindows	Define a list of spectralwindows
string	Define a string
table	Define a table
taql	Define a Table Query Language string
untyped	Define an untyped variable
vector_boolean	Define a vector of booleans
vector_complex	Define a vector of complexs
vector_double	Define a vector of doubles
vector_float	Define a vector of floats
vector_integer	Define a vector of integer
vector_string	Define a vector of strings

types.exists - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Does the class, method, arg exist?

Synopsis
exists(class, method, arg)

Description
Does the class, method, arg exist? class may be composite.

Arguments

class	Class or type of tool Allowed:String
method	Method Allowed:String Default: unset
arg	Argument Allowed:String Default: unset

Example

```
- types.exists('imager','advise','cell')
T
- types.exists('imager.advise')
T
- types.exists('imager.integrate')
F
```

---

types.classes - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Return names of the classes

Synopsis
classes()

---

types.class - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a class or type of tool

Synopsis
class(class)

Arguments

class	Class or type of tool Allowed:String

---

types.group - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define the group name for a function

Synopsis
group(group)

Description
Functions may be assigned to names groups in order to bring some structure to the user interface. The default name is 'basic'.

Arguments

group	Group name Allowed:String Default: basic

---

types.includefile - Function

Package	**utility**

Module	**tasking**
Tool	**types**

Specify the include file for a class

Synopsis
includefile(file, class)

Arguments

file	File name Allowed:String
class	Class or type of tool Allowed:String Default: unset

---

types.getincludefile - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Get the include file for a class

Synopsis
getincludefile(class)

Arguments

class	Class or type of tool Allowed:String Default: unset

---

types.method - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a method or function for a class

Synopsis
method(method, title, label, category)

Arguments

method	Name of method Allowed:String
title	Title Allowed:String Default: unset
label	label Allowed:String Default: unset
category	Category Allowed:String Default: unset

---

types.list - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a list of other types

Synopsis
list(arg, types, options, name, names, default, dir, checkeval, help, allowunset)

Description
This widget allows the assembly of types containing zero or more composite types, where the composite type can be made of one or more other types. The user may then add or delete these composite types.
Any number of types may be included in a composite type. The corresponding arguments are given in either vector string or record form.
The test function typestest demonstrates a list type.

Arguments

arg	List of arguments Allowed:String
types	List of types Allowed:String
options	Record of options Allowed:Record
name	Name of the composite type Allowed:Vector String
names	Names of individual types Allowed:Vector String
default	Default values Allowed:Region Default: unset
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

Example

```
# The following defines a list containing two other types: string
# and direction.
    list('Sources', types=['string', 'direction'], names="Source Direction",
 name='Source', default=[Name='3C273', Direction=dm.direction()],
 help=['Source name', 'Source direction'],
 allowunset=T,
 options=unset);
```

---

types.integer - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define an integer

Synopsis
integer(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Integer Default: 1
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.vector_integer - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a vector of integer

Synopsis
vector_integer(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Vector of Int Default: []
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.float - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a float

Synopsis
float(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Float Default: 0.0
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.vector_float - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a vector of floats

Synopsis
vector_float(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Vector of floats Default: []
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.double - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a double

Synopsis
double(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Double Default: 0.0
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.vector_double - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a vector of doubles

Synopsis

vector_double(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Vector of doubles Default: []
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.complex - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a complex

Synopsis
complex(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Complex Default: 0+0i
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.vector_complex - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a vector of complexs

Synopsis

vector_complex(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Vector of complexs Default: []
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.string - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a string

Synopsis
string(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:String Default: ”
dir	Direction of variable Allowed:String: ’in’—’inout’—’out’ Default: ’in’
checkeval	Check eval’ability Allowed:Boolean Default: F
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.vector_string - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a vector of strings

Synopsis
vector_string(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Vector of strings Default: [”]
dir	Direction of variable Allowed:String: ’in’—’inout’—’out’ Default: ’in’
checkeval	Check eval’ability Allowed:Boolean Default: F
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.measure - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a measure

Synopsis
measure(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Measure Default: 'dm.direction('b1950', '0deg', '0deg')'
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.measurecodes - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define the reference code for a measure

Synopsis
measurecodes(arg, default, dir, options, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Measure Default: 'dm.direction('b1950', '0deg', '0deg')'
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
options	List of options Allowed:Vector of strings Default: unset
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.epoch - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define an epoch measure

Synopsis
epoch(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Measure Default: dm.epoch('utc', 'today')
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.position - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a position measure

Synopsis
position(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Measure Default: dm.observatory('vla')
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.direction - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a direction measure

Synopsis
direction(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Measure Default: dm.direction('b1950', '0deg', '0deg')
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.quantity - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a quantity

Synopsis
quantity(arg, default, dir, checkeval, help, options, allowunset)

Description
The type of the quantity is fixed on construction. It is determined by the type of the default, or if that is unset, from the options string.

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Quantity Default: '0deg'
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
options	Default type Allowed:Vector of strings Default: angle—time—flux—unnormalizedfluxdensity—vel—long—lat—len—temp
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.choice - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Choose one from a list of strings

Synopsis
choice(arg, default, options, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:String Default: ”
options	List of options Allowed:Vector of strings Default: unset
dir	Direction of variable Allowed:String: ’in’—’inout’—’out’ Default: ’in’
checkeval	Check eval’ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.check - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Choose one or more from a list of strings

Synopsis
check(arg, default, options, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:String Default: ”
options	List of options Allowed:Vector of strings Default: unset
dir	Direction of variable Allowed:String: ’in’—’inout’—’out’ Default: ’in’
checkeval	Check eval’ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.boolean - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a boolean

Synopsis
boolean(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Boolean Default: T
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.vector_boolean - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a vector of booleans

Synopsis
vector_boolean(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Vector of booleans Default: []
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.record - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a record

Synopsis
record(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Record Default: [=]
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.table - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a table

Synopsis
table(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:String Default: ”
dir	Direction of variable Allowed:String: ’in’—’inout’—’out’ Default: ’in’
checkeval	Check eval’ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.taql - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a Table Query Language string

Synopsis
taql(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:String Default: ”
dir	Direction of variable Allowed:String: ’in’—’inout’—’out’ Default: ’in’
checkeval	Check eval’ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.msselect - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a taql string for a MeasurementSet

Synopsis
msselect(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:String Default: ”
dir	Direction of variable Allowed:String: ’in’—’inout’—’out’ Default: ’in’
checkeval	Check eval’ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.file - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a file

Synopsis
file(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:String Default: ”
dir	Direction of variable Allowed:String: ’in’—’inout’—’out’ Default: ’in’
checkeval	Check eval’ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.directory - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a directory

Synopsis
directory(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:String Default: ”
dir	Direction of variable Allowed:String: ’in’—’inout’—’out’ Default: ’in’
checkeval	Check eval’ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.region - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a region

Synopsis
region(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Region Default: unset
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.antennas - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a list of antennas

Synopsis
antennas(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Region Default: unset
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.baselines - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a list of baselines

Synopsis
baselines(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Region Default: unset
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.fields - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a list of field ids

Synopsis
fields(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Region Default: unset
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.spectralwindows - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define a list of spectralwindows

Synopsis

spectralwindows(arg, default, dir, checkeval, help, allowunset)

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Region Default: unset
dir	Direction of variable Allowed:String: 'in'—'inout'—'out' Default: 'in'
checkeval	Check eval'ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.untyped - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Define an untyped variable

Synopsis
untyped(arg, default, dir, checkeval, help, options, allowunset)

Description
This is a catch-all for variables that may be of many different types (such as return from the table tool. Don't use it in place of defining the correct type. The behavior is that:

- In determining the type of a given input string or insert argument, various conversions are tried, and the first successful one retained.
- The search list of conversions can be set using the options string.
- The default search list is "measure array coordinates region record string untyped"
- Valid search list items are the entry types that the Entryparser `tool` recognizes.

Arguments

arg	List of arguments Allowed:String
default	Default value Allowed:Any type Default: ”
dir	Direction of variable Allowed:String: ’in’—’inout’—’out’ Default: ’in’
checkeval	Check eval’ability Allowed:Boolean Default: T
help	Help string: if unset, help file is used Allowed:String Default: unset
options	Search order string Allowed:Vector of strings Default: unset
allowunset	Allow unset values? Allowed:Boolean Default: F

---

types.meta - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Return meta information for a type of tool

Synopsis
meta(class, ctors, addhelp)

Description
The type information read from the meta-information file is returned in a record.

Arguments

class	Class or type of tool Allowed:String
ctors	Show constructors Allowed:Boolean Default: F
addhelp	Add the help information? Allowed:Boolean Default: T

Example

```
- types.meta('imager', T, F)
[imager=[data=[toolname=[type=string, default=myimager,
parameters=[ptype=string, allowunset=F, default=myimager,
value=myimager, dir=in, dlformat=toolname, listname=toolname],
help=[text=Glish name for the constructed tool]],
filename=[type=table, default=, parameters=[ptype=table,
popt=Measurement Set, allowunset=F, default=, value=, dir=in,
dlformat=filename, listname=filename]]], gui=autogui, cli=autocli,
group=basic, title=Construct an imager tool],
imagertester=[data=[toolname=[type=string, default=myimager,
parameters=[ptype=string, allowunset=F, default=myimager,
value=myimager, dir=in, dlformat=toolname, listname=toolname],
help=[text=Glish name for the constructed tool]],
filename=[type=table, default=3C273XC1.ms, parameters=[ptype=table,
popt=Measurement Set, allowunset=F, default=3C273XC1.ms,
value=3C273XC1.ms, dir=in, dlformat=filename, listname=filename],
help=[text=Name of MS to create]], size=[type=integer, default=256,
parameters=[ptype=scalar, allowunset=F, default=256, value=256,
dir=in, dlformat=size, listname=size], help=[text=Number of pixels on
a side]], cell=[type=string, default=0.7arcsec,
parameters=[ptype=quantity, popt=[i_am_unset=i_am_unset],
allowunset=F, default=0.7arcsec, value=0.7arcsec, dir=in,
dlformat=cell, listname=cell]], stokes=[type=string, default=IV,
parameters=[ptype=choice, popt=I IV IQUV, allowunset=F, default=IV,
value=IV, dir=in, dlformat=stokes, listname=stokes], help=[text=Stokes
parameters to image (e.g. ' IQUV' )]], coordinates=[type=string,
default=b1950, parameters=[ptype=choice, popt=b1950 gal, allowunset=F,
default=b1950, value=b1950, dir=in, dlformat=coordinates,
listname=coordinates], help=[text=Coordinate system to use e.g. '
j2000,' gal' ]]], gui=autogui, cli=autocli, group=basic,
title=Construct an imager tool for testing]]
- field_names(types.meta('imager', F, F).advise)
data gui cli group title
```

Returns
Record

---

types.addmeta - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Copy meta information to a class

Synopsis
addmeta(class, meta)

Arguments

class	Class or type of tool Allowed:String
meta	Meta-information Allowed:Record

Returns
Boolean

---

types.all - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Show all classes

Synopsis
all(class, meta)

Arguments

class	Class or type of tool Allowed:String
meta	Meta-information Allowed:Record

Returns
String

---

types.is_tool - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Is this a tool?

Synopsis
is_tool(obj, type)

Description
Is this a tool? It must be a record of public functions only. If the type is specified, it will be tested. Then consistency with the meta-information is checked: all functions in the meta information must appear in the object (but not the other way around).

Arguments

obj	Possible object Allowed:Variable
type	Type of variable Allowed:String Default: 'unset'

---

types.attachhelp - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Attach help from the help system

Synopsis
attachhelp(type)

Description
Help information may be defined in the meta-information file or may be copied from that in the .help files.

Arguments

type	Type of tool Allowed:String

---

types.includemeta - Function

Package	**utility**
Module	**tasking**
Tool	**types**

Include all meta information files

Synopsis
includemeta(includepath)

Description
Force inclusion of the meta information files so that the global types function is up to date.

Arguments

includepath	Path for include files Allowed:String Default: unset

---

1.8.9 typestest - Function

Package	**utility**
Module	**tasking**

Test the types definition

Synopsis
typestest()

Description
This defines a class called typestest, a global function called typestester, and the arguments for typestester. The toolmanager is then called to display the interface for typestester, which may then be executed to see the arguments passed in.

1.8.10 plugins - Tool

Package	**utility**
Module	**tasking**

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	**tasking**
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	**tasking**
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	**tasking**
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	**tasking**
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)

1.8.11 asynceval - Tool

Package utility
Module tasking

Asynchronous eval

Description

`asynceval` is a simple tool for asynchronous evals. It is mainly of use in Glish programming. You give a command that is to be eval()'ed and then you can wait for the result. It's particularly useful for commands that may fail for some unexpected reason. The Glish interpreter is not then blocked.

The events recognized are:

run (accepted) the argument is a command string to be eval()'ed.

result (emitted) The argument is the result of the eval().

Constructors

asynceval	Construct asynceval tool

Functions

done	Kill this asynceval

asynceval - Constructor

Package	**utility**
Module	**tasking**
Tool	**asynceval**

Construct asynceval tool

Synopsis
asynceval()

Description
This makes an asynceval tool. It is a good idea to make one of these when needed and to kill it (via done) when finished.

---

asynceval.done - Function

Package	**utility**
Module	**tasking**
Tool	**asynceval**

Kill this asynceval

Synopsis
done()

Description
The asynceval is terminated. All existing whenevers are deactivated and the subsequence itself is destroyed.

Returns
Bool

1.8.12 servers - Tool

Package utility
Module tasking

A tool for sending events between glish and precompiled executables
include "servers.g"

Description
servers communicates between glish distributed objects (tools) and C++ executables. It provides the following services:

- Creation, execution and destruction of C++ servers
- Parameter passing to and from servers
- Plotting, progress meters, *etc.* (implemented in Glish or via a Glish layer)
- Synchronous and asynchronous execution of functions
- Logging of messages

There is only one servers tool running at any one time called defaultservers. This *must* be running if the tool system is to work at all.
Users normally need not interact with defaultservers. Its functions will largely be used by programmers writing tools and most of the interactions are hidden inside the tool framework (this is described in Note 197). The only major exceptions are:

- When one wants to retrieve values returned by a function that was running asynchronously. This is described in the result function.
- When one wants the name of a plotter used by a tool. This is returned by the getplotter function

The terminology used here needs some explanation:

server A server is a C++ executable that runs as a process.

object A server may contain one or more objects. An object is equivalent to a Glish tool. It usually has multiple functions that may be executed.

Some servers are pre-started to minimize AIPS++ startup time. The default pre-started clients are: timer, misc, and quanta. If you wish to prevent this, set the following in your .aipsrc file:

```
user.prestart: none
```

If you wish to extend or change the list, set the following in your .aipsrc file:

```
user.prestart: timer misc quanta measures
```

Integers are used to index into lists of servers and objects. Thus to specify a server requires one integer, and to specify an object requires two: one for the server and one for the object. Thus the functions of servers require one or both of these, depending on whether the operation is server or object-specific.

Constructors

Functions

activate	Create a new server
add	Add an object to an existing server
agents	Return a record of all agents
alerter	Relay events on completion
busy	Is the server busy?
create	Create an object
done	Delete a tool
getplotter	Get the plotter used by a tool
info	Print information on the existing servers.
init	Initialize servers
isvalid	Is this a valid server?
kill	Kill a given agent
memory	Return the memory usage of each server
methods	Return the methods of an object
result	Get the result(s) from an asynchronous function.
run	Run a function in an object
running	Determine if an asynchronous function is running
sendmemoryevents	Enable the generation of memory events
settimeout	Set event timeouts
stopmemoryevents	Disable the generation of memory events
suspend	Suspend certain clients
timeouts	Return record of timeouts
trace	Trace events related to certain clients

servers.activate - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Create a new server

Synopsis
activate(server, host, forcenewserver, async, terminateonempty)

Description

The activate function creates a new server of the specified type, if necessary. A server is an executable piece of code that has been written to communicate with glish. This function will start the specified process on the specified computer or, by default, on the current computer ie., the one running glish. This function will only create new server process if one is not already running on the specified computer. However this can be overridden.
This function returns a positive integer that can be used by other functions in this tool indicate which server process you are referring to.
Normally servers are terminated when empty ie., contains no tools. One can override this using the terminateonempty flag. Use this only for servers that are likely to be restarted soon to avoid the overhead for reloading the executable.

Arguments

server	The name of the server to create Allowed:a string Default: no default
host	The name of the computer on which to start the server Allowed:any computer name Default: an empty string
forcenewserver	Start a new server even if one is running Allowed:T or F Default: F
async	Start the server asynchronously Allowed:T or F Default: F
terminateonempty	Terminate servers when empty Allowed:Boolean Default: T

Returns
A positive integer

Example

```
include 'servers.g';
private := [=];
private.serverid := defaultservers.activate('numerics');
private.toolid := defaultservers.create(private.serverid, 'fftserver');
```

This will start up the numerics process on the current machine. It then creates a fftserver tool, using the default constructor, that uses this server. See also the example for the create function.

---

servers.create - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Create an object

Synopsis
create(id, type, creator, invokerecord)

Description

The create function creates an object. The object that is created must be one of those inside the specified server.
Every C++ object that can be created with this function has a className member function. The string returned by this function is the string that must be specified in the type argument to create an instance of that class.
Some objects may be constructed only using a default constructor. For these objects the creator and invokerecord arguments should be left at their default values.
Objects which can be constructed in multiple ways will have a "factory" class associated with them. This factory class will map the string specified in the creator argument into the appropriate constructor.
Some constructors may require additional parameters and these are specified using fields in the invokerecord argument. The parameters that are relevant are defined using Parameter objects in the appropriate factory class.

Arguments

id	Identifies which server to use Allowed: A positive integer obtained from the activate function. Default: no default
type	The object to create. Allowed: A string that corresponds to the output of the className function in the C++ class. Default: no default
creator	A string indicating which constructor to use. Allowed: An empty string or one that matches a string in the "factory" class. Default: An empty string
invokerecord	Additional construction parameters Allowed: A record with fields containing additional construction parameters. Default: An empty record

Returns
A record whose fields?????

Example

```
include 'servers.g';
private := [=];
private.serverid := defaultservers.activate('componentlist');,
private.toolid := defaultservers.create(private.serverid, 'componentlist',
                                        'readcomponentlist',
                                        [filename='crux.cl', readonly=T]);
```

This will start up the componentlist process on the current machine. It then creates a componentlist object, using the "readcomponentlist" constructor. This constructor has two arguments the file to open and whether it should be opened read-only.

---

servers.add - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Add an object to an existing server

Synopsis
add(agentid, id)

Description

The add function adds an object to an existing server. The id that is specified is presumed to have been returned by a function returning a ID.

Arguments

agentid	Identifies which server to use Allowed: A positive integer obtained from the activate function. Default: no default
id	Identifies the object (as returned from a DO) Allowed: A record returned from a DO function Default: no default

Returns
A record

Example

```
id := defaultservers.run(private.serverid, private.invokerec);
toolid := defaultservers.add(private.serverid, id);
return _define_componentlist(private.serverid, toolid);
```

The run returns an id record. The add function is then used to add this id to the list of objects stored in private.serverid. Finally a new componentlist tool is returned using this server and id.

---

servers.run - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Run a function in an object

Synopsis
run(id, invokerecord, async)

Description
The run function runs a member function in the specified C++ object.
The server that will be used is specified by the id parameter. The object, function within that object, and parameters within that function are specified using the invokerecord argument.
There are two special fields in the invokerecord argument. These are "_method" and "_sequence". The first of these indicates which function to run and hence should be a string that matches one returned by the methods member function in the C++ object. I do not know what the "_sequence" argument does but I guess that it indicates which C++ object to use. The value to use for the second argument is the _sequence field in the record returned by the create function as shown in the example below.
Other fields in the invokerecord function are only required if the specified function needs user supplied parameters. The field names must match the name given to the "Parameter" object associated with the specified function.
The async argument controls whether or not to wait for the execution of C++ function to complete. Functions that are quick to compute are normally run synchronously. When a function is run synchrounously the return value of this function is the return value of the C++ function, or T if the C++ function returns a void.
When the function is run asynchronously the return value is a small integer that identifies the job. This job id can be used in conjunction with the running function to determine when the function has finished. When a function has finished the job id can be used in conjunction with the result function to get the return value of the function.
This function can also return a fail if something bad has happened.

Arguments

id	Identifies which server to use Allowed: A positive integer obtained from the activate function. Default: no default
invokerecord	Parameters for the function that will be run Allowed: A record with the required fields Default: no default
async	Return before the function has completed? Allowed: T or F Default: F

Returns

T, an integer or a fail

Example

```
private.getlabelRec := [_method = 'getlabel',
_sequence = private.toolid._sequence];
public.getlabel := function(which) {
  wider private;
  private.getlabelRec.which := which;
  return defaultservers.run(private.serverid, private.getlabelRec);
}
```

This example shows how to use the run function to call a C++ function that has one parameter, the which argument. This function is always run synchronously and hence the return value of the run function is the return value of the C++ function ie., the label string.

---

servers.done - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Delete a tool

Synopsis
done(id, objectid)

Description

The done function closes ie., it calls the destructor, of the C++ object associated with the specified tool. When all the objects that use a server have been shutdown the done function will also shutdown the server.
Once a object has been shutdown it is generally advisable to diasble the tool associated with the C++ object. An example of how this is done is given below.

Arguments

id	Identifies which server to use Allowed: A positive integer obtained from the activate function. Default: no default
objectid	The object to delete. Allowed: The objectid field of the record returned by the create function. Default: no default

Returns
T or a fail

Example

```
public.done  := function() {
  wider public, private;
  ok := defaultservers.done(private.serverid, private.toolid.objectid);
  if (ok) {
    val private := F;
    val public := F;
   }
  return ok;
}
```

Assuming the toolid and serverid variables have been defined as shown in the examples for the activate or create functions, it is possible to shutdown the tool and perhaps the server using this function.

---

servers.running - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Determine if an asynchronous function is running

Synopsis
running(jobid)

Description

The running function determines whether the specified job, or asynchronous invokation of a C++ function, has finished. It returns T if the job is still running and F otherwise. If a bad job id is passed it returns a fail.

Arguments

jobid	Which job id we are inquiring about Allowed: A positive integer obtained from the run function. Default: no default

Returns
T, F or a fail

Example
See the example for the results function.

---

servers.result - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Get the result(s) from an asynchronous function.

Synopsis
result(jobid, clear)

Description
The results function is used to get the return values after an asynchronous C++ function has finished.
The results for the specified function are obtained using the jobid that is returned when the function is started with the `run` function. Normally these results are cleared when `result` is called; however, it is possible to keep them for future access by setting the `clear` argument to `F`.
The results are provided in a record format with fields corresponding to each of the input parameters. This is to allow some input parameters to also act as output values. If the C++ function returns a value then the record will also contain a `returnval` field that contains the value returned by the C++ function.

Arguments

jobid	Which job id we are inquiring about Allowed:A positive integer obtained from the run function. Default: no default
clear	Delete the results from the internal buffer Allowed:T or F Default: T

Returns
T, F or a fail

Example

```
private.simulateRec := [_method = 'simulate',
     _sequence = private.id._sequence];
public.simulate := function(howmany=1, async=F) {
  wider private;
  private.simulateRec.howmany := howmany;
  retval := defaultservers.run(private.serverid,
                               private.simulateRec, async);
  if (is_fail(retval)) return retval;
  if (async) {
    t := client("timer", 1.0);
    whenever t->ready do {
   if (!defaultservers.running(retval)) {
        t->terminate();
     deactivate;
     note('Added ', howmany, ' simulated component(s) to the list.',
          origin='componentlist.simulate');
   }
    }
  } else {
    note('Added ', howmany, ' simulated component(s) to the list.',
      origin='componentlist.simulate');
  }
  return T;
}
```

This example illustrates how a function can be made to work both synchronously or asynchronously. In either the former case the user waits for the function to complete does not receive a glish prompt until then. In the latter case the functions returns quickly and executation proceeeds in the backgroud. In either case the note is only printed when everything has been done.

---

servers.getplotter - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Get the plotter used by a tool

Synopsis
getplotter(id, name)

Description
When a tool uses a plot command, a pgplotter is created. The plotter is stored by `defaultservers`, and a reference to it may be retrieved using the getplotter function. Since it is a glish reference to the `pgplotter`, commands that change the state of the plotter may be issued. The main use fot this is to allow script to issue print commands to the pgplotter.
The function needs the agentid field in the record returned from the `id()` function of the tool. All tools should have this function, but unfortunately some may not.

Arguments

id	agentid field from record returned by tool id() function Allowed:A positive integer Default: no default
name	Optional name of plotfile Allowed:String Default: ”

Returns
A record containing pgplotters or a fail

Example

```
myimager := imager('3C273XC1.ms');
myimager.plotuv();
imagerplotter := defaultservers.getplotter(myimager.id().agentid)
imagerplotter[1].postscript('3C273XC1.uvplot.ps');
      }
```

This example illustrates how to retrieve and use the `pgplotter`. Note that We assume in this example that the only plotter is the first one. In more complex cases, one might have to use a different element in the returned record.

---

servers.isvalid - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Is this a valid server?

Synopsis
isvalid(agentid)

Arguments

agentid	server id Allowed:Int Default: no default

Returns
Bool

---

servers.busy - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Is the server busy?

Synopsis

busy(agentid)

Arguments

agentid	server id Allowed:Int Default: no default

Returns
Bool

---

servers.settimeout - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Set event timeouts

Synopsis
settimeout(action, timeout)

Description
Normally some crucial events are just awaited. It is possible to set a timeout for some of the events. Normally this should not be needed except in debugging. The events are:

create Create a server

close Close a tool

done Close a server

makeplot Make a plot server

Arguments

action	event name Allowed:String Default: create
timeout	Time out in seconds Allowed:Numeric Default: 60

Returns
Bool

---

servers.timeouts - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Return record of timeouts

Synopsis
timeouts()

Description
Normally some crucial events are just awaited. It is possible to set a timeout for some of the events. Normally this should not be needed except in debugging. The events are:

create Create a server

close Close a tool

done Close a server

makeplot Make a plot server

Returns
Record of timeouts

---

servers.info - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Print information on the existing servers.

Synopsis
info()

Returns
Bool

---

servers.memory - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Return the memory usage of each server

Synopsis
memory()

Description
servers keeps track of how much memory each server is using. This function will return a record containing the amount of memory, in mega-bytes, used bt each server. For convienience the amount of memory used by Glish is also returned.

Returns
record

Example

```
include 'servers.g';
print defaultservers.memory();
# [misc=0.290412903, quanta=0.0799331665, Glish=3.77894211]
```

---

servers.sendmemoryevents - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Enable the generation of memory events

Synopsis
sendmemoryevents()

Description
servers can send a memory event whenever it detects that the memory used by a server has changed. The value associated with the memory event is a record identical to the one returned by the memory function.
By default servers does not send events and this function must be called to initiate this behaviour. The complimentary function is stopmemoryevents.

Returns
nothing

Example

```
include 'servers.g';
whenever defaultservers->memory do {
  local memory := $value;
  total := 0;
  for (i in ind(memory)) {
     total +:= memory[i];
  }
  print 'aips++ memory usage is', total;
}
defaultservers.sendmemoryevents();
# At this point the whenever may receive events
defaultservers.stopmemoryevents();
# At this point the whenever will not receive any events.
```

This example creates a "whenever" statement that will print the total memory used by aips++. However the defaultsaervers will not emit memory events until the sendmemoryevents function is called.

---

servers.stopmemoryevents - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Disable the generation of memory events

Synopsis
stopmemoryevents()

Description
servers can send a memory event whenever it detects that the memory used by a server has changed. This function can be used to prevent this from happening. It is complementry to the sendmemoryevents function.
Using this function does does not guarentee that servers will stop sending memory events. servers counts how many times the sendmemoryevents function has been called and the stopmemoryevents function must be called an equal number of times to actually stop the memory events. This is to ensure that one client does not stop events for other clients that have requested them.

Returns
nothing

Example
See the example for the sendmemoryevents function.

---

servers.init - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Initialize servers

Synopsis
init()

Description
This function initializes servers after construction. It should not be called other than in servers.g.

Returns
don't know

---

servers.alerter - Function

Package **utility**
Module **tasking**
Tool **servers**

Relay events on completion

Synopsis
alerter()

Description
servers works internally by sending and receiving events. When running synchronously these events are converted into return values as appropriate. However, when running asynchronously, one needs a mechanism to emit events that can be caught outside. The alerter function provides this mechanism. The events emitted are:

error The job ended in error.

fail The server failed.

note A note was issued.

'jobid' The job with id 'jobid' ended successfully.

The value returned by the event is a record with fields:

agent The agent used in execution.

value The value.

Here is an example of how to execute a function asynchronously and set up a whenever on the result:

```
im:=imager('3C273XC1.ms')
jobid:=im.plotvis(async=T);
whenever defaultservers.alerter()->[as_string(jobid)] do {
  print 'Plot finished';
  deactivate;
}
```

Returns
Agent record for receiving events and their results

---

servers.agents - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Return a record of all agents

Synopsis
agents()

Description
Needed for debugging only.

Returns
Record

---

servers.suspend - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Suspend certain clients

Synopsis
suspend(suspend_clients)

Description
Needed for debugging only.

Arguments

suspend_clients	Clients to suspend Allowed:String Default: no default

Returns
Bool

servers.trace - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Trace events related to certain clients

Synopsis
trace(trace_clients)

Description
Needed for debugging only.

Arguments

trace_clients	Clients to trace Allowed:String Default: no default

Returns
Bool

---

servers.kill - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Kill a given agent

Synopsis
kill(agentid)

Description
Kill a given agent.