Synopsis
infowindow(sometext, title, selfdestruct, timeout, ws)
include "infowindow.g"

Description
The infowindow is a simple dialog window that lets one display an information message in a window. If selfdestruct is specified, the window will dismiss itself after the timeout period specified. Depending on the length of the text (more than 200 characters) the message will either appear as a message widget or in a scrolling text widget.

Arguments

sometext	Text of message you wish to display Allowed:string
title	Title to put in window Allowed:string
selfdestruct	Do you want this window to automatically disappear? Allowed:Boolean Default: F
timeout	specifies the number of ticks in .2s for the window to automatically dismiss itself Allowed:integer Default: 150
ws	widget server Allowed:agent Default: dws

Returns
T (or fail)

Example

```
include 'infowindow.g';
iw := infowindow('Something good will happen to you.');
```

1.2 plotter - Module

Module for plotting

Tools

pgplotmanager	tool used in application development to send plot commands to a plot device
pgplotter	stand alone pgplotter tool
pgplotwidget	tool used in application development to embed a pgplot canvas into some GUI

Functions

pgplottertest	Test pgplotter by drawing the examples

Description

The plotter module has a hierarchy of plotting tools layered on top of the Glish/PGPLOT client.

- pgplotmanager – a generic tool for plotting to any pgplot device. Most interactive users would not use this low-level tool. It is meant to help application developers build applications that can draw plots to either to the screen or to an output file. It adds convenience plotting functions to the basic Glish/PGPLOT client along with functions to manage complex plots via display lists.
- pgplotwidget – An on-screen plotting canvas adds to the functionality of the pgplotmanager tool with functions that assist with interactions with the GUI device. This tool is oriented toward application developers that wish to add specialized plotting to their applications.
- pgplotter – A general-purpose, stand-alone display tool designed for interactive users, allowing them to draw line and raster plots from the Glish command line. It has all the functionality of the pgplotwidget and adds additional functions convenient for the interacitve user.

You might want to use the direct Glish/PGPLOT binding if plotting efficiency is of concern. However, the above tools are usually sufficiently fast.

1.2.1 pgplotmanager - Tool

Package display
Module plotter

tool used in application development to send plot commands to a plot device
include "pgplotmanager.g"

Constructors

pgplotmanager	Manage a previously created pgplot agent
pgplotps	Create a pgplotmanager attached a file

Functions

addredrawfunction	Add a function to redraw a command in the displaylist
canplay	return whether a command can be executed
clear	empty the display list
displaylist	return the displaylist currently in use
done	shutdown the manager
getagent	return the PGPLOT agent being managed by this tool
lastchange	Version of the last change
maskline	Draw line segments based on input vectors and a mask
play	Execute a series of plot commands
plotfile	Save the plot in native format
plotxy	Plot X,Y vectors as lines or points with auto-scaling
record	set whether drawing commands are saved
recording	return whether plot commands are being saved
refresh	replay the display list
resetplotnumber	reset the internal plot number counter to 0
restore	Retrieve a plot
setagent	set the PGPLOT agent to be managed by this tool
settings	Set many PGPLOT attributes at once

Description

The `pgplotmanager` is tool that is useful for building Glish applications that do specialized plotting. Many applications will prefer instead to use the pgplotwidget which is an extension of the pgplotmanager tool, specialized for plotting to a graphical widget. (Interactive users will normally use the pgplotter tool.) In contrast, pgplotmanager is device-independent; that is, its capabilities are relevent to plotting either to a widget or to a file. As a result, applications can:

- use the same code for plotting to both graphical and hardcopy devices
- open a hardcopy device without opening a graphical widget (as is necessary with the pgplotwidget tool) to do non-interactive plotting.

See the Example below for details on how one can accomplish this using `pgplotmanager` tool.

pgplotmanager is built on top of the "raw" Glish/PGPLOT agent, providing access to most of its capabilities through its tool functions. (That is, instead accessing pgplot function through the event syntax, e.g. `pg->line()`, you access it like any tool function: `pg.line()`.) In addition, a few higher level functions have been added:

- `settings()` — sets many PGPLOT attributes (e.g. line width, character height) all at once.

- `plotxy()` — a simple function for plotting a "y" array versus an "x" array.
- `maskline()` — similar to the the "raw" function `line()`, but uses a mask array to determine with points plot.

An important added feature is pgplotmanager's maintenance of an internal *display list*—a record of plot commands (i.e. calls made to the plotmanager's plotting functions) stored inside the tool that can be played back at any time. This enables several useful capabilities:

- when the GUI is resized, the plot commands can be replayed to refresh the plot, filling the new size.
- plot commands can be save to a "plot" file, which can be opened and edited later (for example, by a user via pgplotter).
- plot commands sent to a GUI can be replayed to a PostScript file.
- recording of commands can be turned "on" and "off." This allows you to "erase" parts of the plot. For example, you could interactively draw ellipses on the plot widget with recording turned off, and then remove the ellipses by doing a replaying the commands that were recorded.

Only those functions that add functionality beyond that of the Glish pgplot agent are described explicitly in this reference manual section. For descriptions of the "raw" pgplot agent functions, refer to the Glish/PGPLOT description found in the Glish User's Manual. Those functions are as follows:

arro – draw an arrow

ask – control new page prompting

bbuf – begin batch of output (buffer)

beg – begin PGPLOT, open output device

bin – histogram of binned data

box – draw labeled frame around viewport

circ – draw a filled or outline circle

clos – close the selected graphics device

conb – contour map of a 2D data array, with blanking

conl – label contour map of a 2D data array

cons – contour map of a 2D data array (fast algorithm)

cont – contour map of a 2D data array (contour-following)

ctab – install the color table to be used by PGIMAG

draw – draw a line from the current pen position to a point

ebuf – end batch of output (buffer)

end – terminate PGPLOT

env – set window and viewport and draw labeled frame

eras – erase all graphics from current page

errb – horizontal or vertical error bar

errx – horizontal error bar

erry – vertical error bar

gray – gray-scale map of a 2D data array

hi2d – cross-sections through a 2D data array

hist – histogram of unbinned data

iden – write username, date, and time at bottom of plot

imag – color image from a 2D data array

lab – write labels for x-axis, y-axis, and top of plot

ldev – list available device types

len – find length of a string in a variety of units

line – draw a polyline (curve defined by line-segments)

move – move pen (change current pen position)

mtxt – write text at position relative to viewport

numb – convert a number into a plottable character string

open – open a graphics device

page – advance to new page

panl – switch to a different panel on the view surface

pap – change the size of the view surface

pixl – draw pixels

pnts – draw one or more graph markers, not all the same

poly – fill a polygonal area with shading

pt – draw one or more graph markers

ptxt – write text at arbitrary position and angle

qah – inquire arrow-head style

qcf – inquire character font

qch – inquire character height

qci – inquire color index

qcir – inquire color index range

qcol – inquire color capability

qcr – inquire color representation

qcs – inquire character height in a variety of units

qfs – inquire fill-area style

qhs – inquire hatching style

qid – inquire current device identifier

qinf – inquire PGPLOT general information

qitf – inquire image transfer function

qls – inquire line style

qlw – inquire line width

qpos – inquire current pen position

qtbg – inquire text background color index

qtxt – find bounding box of text string

qvp – inquire viewport size and position

qvsz – find the window defined by the full view surface

qwin – inquire window boundary coordinates

rect – draw a rectangle, using fill-area attributes

rnd – find the smallest 'round' number greater than x

rnge – choose axis limits

sah – set arrow-head style

save – save PGPLOT attributes

unsa – restore PGPLOT attributes

scf – set character font

sch – set character height

sci – set color index

scir – set color index range

scr – set color representation

scrn – set color representation by name

sfs – set fill-area style

shls – set color representation using HLS system

shs – set hatching style

sitf – set image transfer function

slct – select an open graphics device

sls – set line style

slw – set line width

stbg – set text background color index

subp – subdivide view surface into panels

svp – set viewport (normalized device coordinates)

swin – set window

tbox – draw frame and write (DD) HH MM SS.S labelling

text – write text (horizontal, left-justified)

updt – update display

vect – vector map of a 2D data array, with blanking

vsiz – set viewport (inches)

vstd – set standard (default) viewport

wedg – annotate an image plot with a wedge

wnad – set window and adjust viewport to same aspect ratio

Example
This example illustrates the pattern one can use to support both interactive, GUI-oriented plotting and non-interactive, batch plotting.

```
# In this tool, the pgplot device is either a widget or a file,
# depending on whether the psfile argument is set.  If it is, we use
# pgplotps() to construct our pgplot tool; otherwise, we use the
# pgplotwidget() constructor.  All plot commands are collected in the
# tool function, drawplot().  This works because, in terms of its
# interface, a pgplotwidget is a pgplotmanager.
#
pipelineplotter := function(psfile=unset) {
    public := [=];
    private := [file=psfile, topframe=F];

    if (is_unset(psfile)) {

        # no filename given, so create a pgplot GUI
include 'widgetset.g';
include 'pgplotwidget.g';
private.topframe := dws.frame();

        private.pgp := pgplotwidget(private.topframe);

# do further setup of GUI as necessary; e.g. setting callbacks

    }
    else {

        # record plot to a file
include 'pgplotwidget.g';
```

```
private.pgp := pgplotps(psfile);
    }

    # This function draws the plot to the pgplot device (be it a GUI
    # or a file.
    public.drawplot := function() {

        # PGPLOT commands go here.  Only those functions supported by
        # the pgplotmanager tool should be used via the tool held in the
# private.pgp variable.  GUI related commands should go elsewhere.

private.pgp.box('BC', 0, 0, 'BC', 0, 0);
# etc....
    }

    # Other functions...

}
```

pgplotmanager - Constructor

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

Manage a previously created pgplot agent

Synopsis

pgplotmanager(pgpagent, interactive, record, playlist, ownplaylist, askfunction, widgetset)

Description

This constructor creates a pgplotmanager tool that will manage an arbitrary pgplot agent. This agent should have already been created with either a graphical widget or hardcopy device (see examples below). The agent can be created either via the widgetserver tool or directly witht the Glish pgplot() global function (see the PGPLOT section of the Glish Manual for details).
Note that for many applications, this constructor is not needed: the pgplotps constructor can be used to attach to a file, and the pgplotwidget tool can be used to attach to a widget.
It is possible to create a pgplotmanager that is attached to neither a widget nor real file device by passing F to this constructor's `pgpagent` parameter.

(This actually causes the output device to be set to the /dev/null file.) This allows one to use the pgplotmanager to record commands into its internal display-list without sending them to any real output device; these commands could be played back to a real device later. This could be done by setting the agent later with the setagent() function. Alternatively, the display list could be save to a file via the plotfile() function and then reloaded later (e.g. in another session) into a new `pgplotmanager` or pgplotter tool.

Care should be taken by the caller of this constructor when the agent is attached to a graphical widget. Widgets are destroyed only when all references to its agent are eliminated. When this tool's done() function is called, the tool will eliminate its internal reference; however, the widget will not disappear if the caller of this constructor still has a reference. See examples below.

Arguments

pgpagent	the pgpagent to manage. A pgpagent is usually constructed either from a widgetset tool's pgplot() function or directly through glish with the pgplot() global function. Alternatively, it can "borrowed" from another pgplotmanager via its getagent() function (but see caveat described above). Allowed:pgplot agent or F Default: F – causes the device to be set to a null file (/dev/null; see explanation above.)
interactive	sets controls whether interactive prompting of the user should be allowed. This is usually set to F if the PGPLOT agent is attached to a file. Allowed:boolean; T – the pgplot ask() function is enabled, allowing the user to turn prompting on and off; F – prompting is not allowed and the ask() function has no effect. Default: T
record	sets whether recording is initially turned on Default: F
playlist	a displaylist tool to use to store plot commands. This allows pre-record commands to be attached to this pgplotmanager. Default: F – a new display list will be created internally
ownplaylist	if true, the displaylist tool passed by the playlist parameter will be shut down when this tool is shut down. This is only used when the playlist parameter is used. Allowed:boolean Default: T
askfunction	the function to call when prompting the user to advance to the next plot. This can either one of the global functions pgplotaskviaprompt() or pgplotaskviagui(), or it can be a user provided function. The interactive parameter must be set to T for this function to get used. Allowed:function Default: F – pgplotaskviaprompt() will be used by default
widgetset	the widgetserver tool to use when creating an internal displaylist. This is used only when a displaylist is not provided via the playlist parameter. Allowed:widgetserver tool Default: dws – the default widgetserver tool

Returns
pgplotmanager tool or fail

Example
In this example, note how the reference to the pgplot agent, in this case, attached to a GUI device, is handled.

```
f := frame();
pgpa := pgplot(f);
pgpm := pgplotmanager(pgpa);
pgpa := F;          # Now pgpm has the only reference to the agent
```

Here, the pgplotmanager is attached to a PostScript file. Note that the pgplotps constructor is the simpler way to do this.

```
pgpa := pgplot('plot.ps/PS');
pgpm := pgplotmanager(pgpa, interactive=F);
pgpa := F;
```

Here, the pgplot agent is set after the construction of the `pgplotmanager` tool.

```
pgpm := pgplotmanager(record=T);

# do some plotting
pgpm.env(0, 2 * pi, -1, 1, 0, 0);

# now attach an agent and replay the plotting done thus far
f := frame();
pgpa := pgplot(f);
pgpm.setagent(pgpa);
pgpm.refresh();
pgpa := F;          # Now pgpm has the only reference to the agent
pgpm.done();        # widget should disappear
```

---

pgplotps - Constructor

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

Create a pgplotmanager attached a file

Synopsis
pgplotps(psfile, overwrite, color, landscape, playlist, record, widgetset)

Description

This constructor creates a pgplotmanager tool attached to an output PostScript file that can be sent to a printer. Interactive prompting will be disabled, so the pgplot function `ask()` will have no effect.

Arguments

psfile	the name of output file to write plot to Allowed: string representing a file name Default: 'pgplot.ps'
overwrite	whether to allow a previously existing file with the same name to be overwritten Allowed: boolean; T – if psfile already exists, it will be overwritten; F – if psfile already exists, this constructor will return a fail value Default: F
color	if true, output is color postscript; otherwise, colors are converted to black-and-white or greyscale Allowed: boolean Default: T
landscape	if true, plot is written in landscape mode (i.e. with the bottom axis of the plot oriented along the long axis of the paper); otherwise, portrait mode is used. Allowed: boolean Default: T
playlist	a displaylist tool to use to store plot commands. This allows pre-record commands to be attached to this pgplotmanager. Default: F – a new display list will be created internally
record	sets whether recording is initially turned on Default: F
widgetset	the widgetserver tool to use when creating an internal displaylist. This is used only when a displaylist is not provided via the playlist parameter. Allowed: widgetserver tool Default: dws – the default widgetserver tool

Returns
pgplotmanager tool or fail

Example

```
pgpm := pgplotps('plot.ps');
```

```
# do some plotting
pgpm.env(0, 2 * pi, -1, 1, 0, 0);

pgpm.done();    # plot file is closed
```

---

pgplotmanager.getagent - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

return the PGPLOT agent being managed by this tool

Synopsis
getagent()

Description
This function simply returns a reference to the PGPLOT agent currently being managed by this tool. This function is rarely needed in most applications; however, it does allow an application to temporarily pass the agent to another pgplotmanager in order, for example, playback its displaylist to the agent. Care should be taken by the caller of this function, especially when the agent is attached to a graphical widget. Widgets are destroyed only when all references to its agent are eliminated. When this tool's done() function is called, the tool will eliminate its internal reference; however, the widget will not disappear if the caller of this function still has a reference. Note in the example below how the caller takes care to destroy its reference to the agent when it's done with it.

Returns
pgplot agent

Example
In this example, the pgplot agent is passed to another pgplotmanager tool. This is, admittedly, a somewhat contrived example.

```
# this will record commands for drawing a special plot
drawplot := pgplotmanager(record=T);
drawplot.env(0, 2 * pi, -1, 1, 0, 0);
```

```
# this will display the plot to the user
f := frame();
pgpm := pgplotmanager(pgplot(f));

# draw plot to widget
pgpa := pgpm.getagent();
drawplot.setagent(pgpa);
drawplot.refresh();
drawplot.setagent(close=F);  # reset agent to null, but don't close widget
pgpa := F;                   # Now pgpm has the only reference to the agent

# now continue plotting
pgpm.arro(1, -1, 5, -0.5)
```

---

pgplotmanager.setagent - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

set the PGPLOT agent to be managed by this tool

Synopsis
setagent(pgpagent, close)

Description
This function makes it possible to switch in a new pgplot client for management by this tool. This is most useful after constructing pgplotmanager tool that isn't attached to an output device.
Care should be taken by the caller of this function, especially when the new agent is attached to a graphical widget. Widgets are destroyed only when all references to its agent are eliminated. When this tool's done() function is called, the tool will eliminate its internal reference; however, the widget will not disappear if the caller of this function still has a reference. Note in the example below how the caller takes care to destroy its reference to the agent after passing it to this function.

Arguments

pgpagent	the pgpagent to manage. A pgpagent is usually constructed either from a widgetset tool's pgplot() function or directly through glish with the pgplot() global function. Alternatively, it can "borrowed" from another pgplotmanager via its getagent() function (but see caveat described above). Allowed:pgplot agent or F Default: F – causes the device to be set to a null file (/dev/null; see explanation above.)
close	if true, the previous agent will be explicitly closed before its replacement is set. Set this to F if agent is attached to a GUI being shared with another pgplotmanager Allowed:boolean Default: T

Returns
T or fail if pgpagent is not an agent

Example
In this example, the pgplot agent is set after the construction of the `pgplotmanager` tool.

```
pgpm := pgplotmanager(record=T);

# do some plotting
pgpm.env(0, 2 * pi, -1, 1, 0, 0);

# now attach an agent and replay the plotting done thus far
f := frame();
pgpa := pgplot(f);
pgpm.setagent(pgpa);
pgpm.refresh();
pgpa := F;          # Now pgpm has the only reference to the agent
pgpm.done();        # widget should disappear
```

In the next example, the pgplot agent is passed to another pgplotmanager tool. This is, admittedly, a somewhat contrived example.

```
# this will record commands for drawing a special plot
drawplot := pgplotmanager(record=T);
drawplot.env(0, 2 * pi, -1, 1, 0, 0);

# this will display the plot to the user
f := frame();
pgpm := pgplotmanager(pgplot(f));
```

```
# draw plot to widget
pgpa := pgpm.getagent();
drawplot.setagent(pgpa);
drawplot.refresh();
drawplot.setagent(close=F);  # reset agent to null, but don't close widget
pgpa := F;                   # Now pgpm has the only reference to the agent

# now continue plotting
pgpm.arro(1, -1, 5, -0.5)
```

---

pgplotmanager.resetplotnumber - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

reset the internal plot number counter to 0

Synopsis
resetplotnumber()

Description
This function resets the internal plot number counter to 0. The pgplotwidget maintains an internal counter which is used to determine when to prompt the user (when function `ask(T)` has been called). When prompting is desired, it's usually only necessary to do so after the first plot has been presented. This function resets to 0 the counter that tracks how many plots have been presented so far; thus, for the next plot after calling this function, the user won't be prompted.

Example

```
pgpm.ask(T);
pgpm.page();              # user may get prompted to confirm start of new plot
pgpm.resetplotnumber();
pgpm.page();              # user will not be prompted to confirm
```

---

pgplotmanager.displaylist - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

return the displaylist currently in use

Synopsis
displaylist()

Description
The pgplotmanager tool maintains a record of plot commands that have been issued (when recording is turned on, see record() and recording()) in the form of a displaylist tool. This function returns a reference to that tool. Because a reference is returned, you should be careful that you don't damage it.
The most common use of this tool might be to temporarily pass the displaylist to another pgplotmanager tool. In this case, `ownplaylist` parameter to the constructor should be set to `F` to prevent the second `pgplotmanager` tool does not shut it down. (See example below.)
You could also use this function to, for example, go backwards through the plot command to find the last data plotted, extract it, fit it, and plot the fit to the data.

Returns
displaylist tool

Example
This example shows how the commands sent to a pgplot widget could be replayed to a file. In practice, this can be done more straightforwardly with the pgplotwidget tool's postscript function.

```
# create the widget
f := frame();
pgpa := pgplot(f);
pgpw := pgplotmanager(pgpa);
pgpa := F;

# do some plotting
pgpm.env(0, 2 * pi, -1, 1, 0, 0);

# create a manager of a postscript file, passing in the displaylist
```

```
# from the widget
dl := pgpw.displaylist();
pgpf := pgplotps('plot.ps', playlist=dl, ownplaylist=F);

# replay plotting commands to file
pgpf.refresh();
pgpf.done();      # PostScript file is now closed

# do some more plotting in the widget
pgpw.arro(1, -1, 5, -0.5);
```

pgplotmanager.record - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

set whether drawing commands are saved

Synopsis
record(newstate)

Description
This function gives you control of whether or not the plotting commands are saved in the internal display list or not. You might want to do this for efficiency concerns (e.g. the commands are coming frequently and contain a lot of pixels), or so that you can "undraw" some temporary plotting with a refresh.
The return value is the *previous* state of the function to make it easy to get back to the previous state. Alternatively, the recording() function can be used to determine the current state.

Arguments

newstate	if true, recording is turned on; otherwise, it is turned off Allowed:boolean

Returns
boolean – the former recording state

Example

```
# Turn on saving of plot commands
pg.record(T);

# Do "permanent" part of plot
pg.env(0, 2 * pi, -1, 1, 0, 0);

# Turn off saving of plot commands
pg.record(F);

# Add "temporary" parts of plot
pg.arro(1, -1, 5, -0.5);

# Make the temporary stuff go away by replaying "permanent" commands
pg.refresh();
```

---

pgplotmanager.recording - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

return whether plot commands are being saved

Synopsis
recording()

Description
The pgplotmanager tool maintains a record of plot commands that have been issued in the form of a displaylist tool. This allows this tool to replay the commands later to, for example, refresh the screen or send a currently viewed plot to a PostScript file.
This function simply returns whether plot commands are currently being saved internally for later replay. A returned value of `T` indicates that recording is currently turned on and commands are being saved. The recording state can be changed via the record() function.

Returns
boolean – the current recording state

---

pgplotmanager.canplay - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

return whether a command can be executed

Synopsis
canplay()

Description
This function returns true if the given command is recognized as a supported plot command. To be recognized, the `command` parameter must be the name of a plotting function of this tool (either a "raw" plotting command supported by the Glish/PGPLOT agent or higher level command, like settings() or plotxy()) or a custom command that has been added via the addredrawfunction() function (usually as part of a plug-in).

Example
Consider this session in which no custom commands have been added:

```
- print pg.canplay('plotxy');
T
- print pg.canplay('gray');
T
- print pg.canplay('grayscale');
F
```

---

pgplotmanager.play - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

Execute a series of plot commands

Synopsis
play(commands, record)

Description
This function is used to execute a series of plot commands in this plotter. It is used internally (e.g. by refresh), but it can also be of use to programmers. For example, you could keep external "decorations" in a list, and show or don't show them by calling `play`.
The format of the commands is that it is either a `displaylist` from a `pgplotwidget`, or it is a record, each field of which is a plot command of the form, e.g.:

```
[_function='line', xpts=1:10, ypts=1:10]
```

Arguments

commands	list of plot commands to execute Allowed:displaylist or record
record	save commands in our internal displaylist? Allowed:boolean

Returns
T or fail

---

pgplotmanager.plotfile - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

Save the plot in native format

Synopsis
plotfile(file)

Description
This function saves the plot as an AIPS++ Table. The plot can then be reopened and new commands added or old commands changed. Basically it saves the internal display list, so any commands that have not been recorded will not be saved. If the file already exists (and is writable), it will be overwritten without any request for confirmation.

Arguments

file	file name Allowed:string Default: aips++plot.plot	

Returns
T or fail (e.g., file not writable)

---

pgplotmanager.refresh - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

replay the display list

Synopsis
refresh()

Description
This function redraws the plot. The most common use for it is to redraw the plot after the frame size has been changed.

Example

```
- f := frame()
- pg := pgplotmanager(pgplot(f));
- whenever f->resize do pg.refresh();
```

Arguments

Returns
T or fail

---

pgplotmanager.restore - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

Retrieve a plot

Synopsis
restore(file)

Description
This function retrieves a plot file that has been saved by `plotfile`. The current plot is replaced with the plot in the file. If you want to merge two plot files you can use `play` on the displaylist of another `pgplotwidget`.

Arguments

file	file name Allowed:string Default: aips++plot.plot

Returns
T or fail (e.g., file not readable)

---

pgplotmanager.done - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

shutdown the manager

Synopsis
done()

Description
This function will shut down this `pgplotmanager` tool. If the tool has the sole

reference to its pgplot agent, the agent's device will be closed. For widget devices, the widget will be destroyed. If the device is a file, it will be closed.

---

pgplotmanager.plotxy - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

Plot X,Y vectors as lines or points with auto-scaling

Synopsis
plotxy(x, y, plotlines, newplot, xtitle, ytitle, title, linecolor, ptsymbol, mask)

Description
`plotxy` is defined to allow you to make line or scatter plots. The routine will scale and draw the axes appropriately, or you can specify to plot over a previous plot using the existing scaling. Axis labels and a plot title can be specified. You can also specify a mask, if desired, to blank out bad data points from the plot. When a mask is given, the unmasked data are plotted in line segments.

Example
Note that the examples below can be applied to pgplotwidget and pgplotter tools with greater ease since both support all of pgplotmanager's functions.

```
include 'pgplotmanager.g'
f := frame();
pg := pgplotmanager(pgplot(f));
x := 1:100;
y := x*x;
pg.plotxy(x,y,xtitle='XX', ytitle='YY', title='TITLE');
pg.plotxy(x+10, y, F, F);
```

Example

```
include 'pgplotmanager.g'
f := frame();
pg := pgplotmanager(pgplot(f));
x := 1:100;
```

```
y := sqrt(x);
pgmask := array(T,100)
pgmask[20:30] := F
pgmask[50:55] := F
pg.plotxy(x,y,mask=pgmask);
```

Arguments

x	X values Allowed:non-complex numeric array
y	y values Allowed:non-complex numeric array
plotlines	Plot lines (T) or points (F) Allowed:boolean Default: T
newplot	Start a new plot (T) or overplot (F) Allowed:boolean Default: T
xtitle	x axis title (ignored if newplot is F) Allowed:string Default: none
ytitle	y axis title (ignored if newplot is F) Allowed:string Default: none
title	plot title (ignored if newplot is F) Allowed:string Default: none
linecolor	color index for lines or points Allowed:integer Default: 2
ptsymbol	symbol index (used if plotlines=F) Allowed:integer Default: 2
mask	mask for eliminating bad data points (T for good data, F for masked data) Allowed:boolean array Default: none

Returns
T or fail (illegal argument)

---

pgplotmanager.maskline - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

Draw line segments based on input vectors and a mask

Synopsis
maskline(x, y, mask, decimate)

Description

`maskline` is analogous to `line` except that it takes a mask (an array of boolean elements) and draws a series of line segments with the masked data eliminated.
The argument `decimate` is used to speed up the search for masked pixels. It breaks the spectrum into `decimate` segments. It uses built in *Glish* functions to detect whether there are any masked pixels in each segment. If there are, then those masked pixels are found, one by one with the *Glish* indexing operator (slow). If you set `decimate==1` then each pixel of the spectrum is searched for masked pixels with the *Glish* indexing operator (very slow). The default decimation of 10 is safe for any length spectrum (i.e. it will realize if there aren't enough pixels for 10 segments).

Example

```
include 'pgplotmanager.g'
f := frame();
pg := pgplotmanager(pgplot(f));
x := 1:100;
y := sqrt(x);
pgmask := array(T,100)
pgmask[20:30] := F
pgmask[50:55] := F
pg.env(0,100,0,10,0,0)
pg.maskline(x,y,pgmask)
```

Arguments

x	X values Allowed:non-complex numeric array
y	y values Allowed:non-complex numeric array
mask	mask for eliminating bad data points (T for good data, F for masked data) Allowed:boolean array Default: none
decimate	Number of segments to decimate into Allowed:Integer Default: 10

Returns
T or fail

---

pgplotmanager.clear - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

empty the display list

Synopsis
clear()

Description
This function empties the display list. This is unlike `eras`, which merely clears the current page. This function erases the screen and removes all accumulated plot commands.

---

pgplotmanager.settings - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

Set many PGPLOT attributes at once

Synopsis
settings()

Description
This function is used to set (possibly) many PGPLOT attributes at once. It is always inserted at the beginning of every plot's display list to make it easy to change settings in a global way. All values have their PGPLOT defaults.

```
ask                              : page prompting
nxsub, nysub                     : plot panelling
arrowfs, arrowangle, arrowvent   : arrow style
font                             : text font
ch                               : character height
ci                               : color index
fs                               : fill style
hsangle, hssepn, hsphase         : hatching style
ls                               : line style
lw                               : line width
tbci                             : text background color
```

---

pgplotmanager.lastchange - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

Version of the last change

Synopsis
lastchange()

Description
This function returns a vaule that changes every time the internal display list changes. It can be used, for example, to see if the plot has changed since the last time it was inspected.

Returns
integer - changes every time the displaylist changes

---

pgplotmanager.addredrawfunction - Function

Package	**display**
Module	**plotter**
Tool	**pgplotmanager**

Add a function to redraw a command in the displaylist

Synopsis
addredrawfunction(name, redrawfunction)

Description
This function is only of interest if you want to add functions via plugins. Suppose you create via a plugin a high-level plot command that you want to show up in the displaylist as a high-level command (*i.e.*, not broken up into lower level PGPLOT commands). In this case, in your plugin you would enter into the displaylist a record with a field per argument, and a field `_function` that contains the name of the function.
To be able to redraw the function, you would use `addredrawfunction` to add a function which, when passed the record the plugin inserted in the display list, would ensure that the draw commands are executed.

Arguments

name	name of the draw function Allowed:string
redrawfunction	function to redraw a command from the displaylist Allowed:function

Returns
T or fail

Boolean

1.2.2 pgplotwidget - Tool

Package	**display**
Module	**plotter**

tool used in application development to embed a pgplot canvas into some GUI include "pgplotwidget.g"

Constructors

pgplotwidget	Create a pgplot widget in a supplied frame

Functions

activatecallback	activate a callback
addredrawfunction	Add a function to redraw a command in the displaylist
canplay	return whether a command can be executed
clear	empty the display list
curs	Get a key and position
cursor	Set the cursor
deactivatecallback	deactivate a callback
displaylist	Access the displaylist plot commands are kept in
done	destroy the widget, release resources
getcallbacks	Recover callbacks
lastchange	Version of the last change
maskline	Draw line segments based on input vectors and a mask
message	write in the message line
play	Execute a series of plot commands
plotfile	Save the plot in native format
plotxy	Plot X,Y vectors as lines or points with auto-scaling
postscript	Save the plot as a PostScript file
record	Save the drawing commands
recording	return whether plot commands are being saved
refresh	replay the display list
resetplotnumber	Reset internal plot number counter to 0
restore	Retrieve a plot
setcallback	set a function to be called for mouse or keyboard events
settings	Set many PGPLOT attributes at once
shortnametofullname	Get a longer description for pgplotwidget commands
size	size in pixels of the pgplotwidget

Description

The pgplotwidget is a PGPLOT style widget to embed in a larger gui, such as pgplotter. It has a number of advantages over using a "raw" Glish/PGPLOT agent.

1. It can automatically resize itself when it is resized.
2. It can save its contents in a "plot" file, which can be opened and edited later (for example, by a User via pgplotter).
3. It can save itself itself as postscript (colour or black and white, landscape

or portrait).

4. It has a one-line "message" window for status reports or instructions to the user.

5. The architecture allows for "higher level" than PGPLOT commands to be added to the widget (either directly or via plugins). At the moment, we have implemented a `settings` function to allow you to set many PGPLOT defaults (line width, character height) at once, `plotxy` to allow you to conveniently plot x,y vectors with auto scaling, and a number of commands to emulate an old plotter `gplot1d`.

Except for the last two items, these advantages accrue by keeping a "display" list internally. That is, the commands sent to the widget are stored internally, and then when, e.g., the frame size is changed the list is replayed. This advantage comes at a cost of requiring more memory and slowing the display somewhat. Nevertheless, for most applications the advantages probably favor this approach.
One interesting capability is the ability to turn on and off the recording of plot commands. This allows you to "lift" drawings off one another. For example, you could interactively draw ellipses on the plot widget but not keep the ellipses in the display list, and then remove the ellipses by doing a refresh.
pgplotwidget features a zoom capability through the Middle Mouse Button (Button 2). Clicking on this and dragging the mouse opens up a zoom window; clicking again selects this as the zoom region, and redisplays. To unzoom, use CNTRL-Middle Mouse Button.
The `pgplotwidget` is built on top of the more generic pgplotmanager tool, which is in turn built on top of the Glish/PGPLOT client. Thus, almost all the basic drawing and inquiry commands described in the Glish/PGPLOT section of the Glish manual are available in the `pgplotwidget`. Calling these commands is different, though: instead of using the event operator, `->`, one uses the normal `.` for accessing tool functions.
Only those functions that add functionality beyond that of the Glish pgplot agent are described explicitly in this reference manual section, specifically:

1. Refreshing the screen (e.g. when the frame resizes).

2. Saving and restoring to and from files.

3. Instead of registering "whenever" statements for interactions (e.g., button pushes) you register callback functions.

4. Standard PGPLOT `curs()` emulation.

5. General manipulation of the displaylist.

6. A "done" function for cleanup.

7. A one line message window.

8. A function for changing many settings (e.g. line width) at once.

Note that the pgplotwidget requires no AIPS++ binaries, so that including it in a non-AIPS++ Glish application is not "expensive" in startup time.
The following standard PGPLOT commands are described in the Glish manual:

arro – draw an arrow

ask – control new page prompting

bbuf – begin batch of output (buffer)

beg – begin PGPLOT, open output device

bin – histogram of binned data

box – draw labeled frame around viewport

circ – draw a filled or outline circle

clos – close the selected graphics device

conb – contour map of a 2D data array, with blanking

conl – label contour map of a 2D data array

cons – contour map of a 2D data array (fast algorithm)

cont – contour map of a 2D data array (contour-following)

ctab – install the color table to be used by PGIMAG

draw – draw a line from the current pen position to a point

ebuf – end batch of output (buffer)

end – terminate PGPLOT

env – set window and viewport and draw labeled frame

eras – erase all graphics from current page

errb – horizontal or vertical error bar

errx – horizontal error bar

erry – vertical error bar

gray – gray-scale map of a 2D data array

hi2d – cross-sections through a 2D data array

hist – histogram of unbinned data

iden – write username, date, and time at bottom of plot

imag – color image from a 2D data array

lab – write labels for x-axis, y-axis, and top of plot

ldev – list available device types

len – find length of a string in a variety of units

line – draw a polyline (curve defined by line-segments)

move – move pen (change current pen position)

mtxt – write text at position relative to viewport

numb – convert a number into a plottable character string

open – open a graphics device

page – advance to new page

panl – switch to a different panel on the view surface

pap – change the size of the view surface

pixl – draw pixels

pnts – draw one or more graph markers, not all the same

poly – fill a polygonal area with shading

pt – draw one or more graph markers

ptxt – write text at arbitrary position and angle

qah – inquire arrow-head style

qcf – inquire character font

qch – inquire character height

qci – inquire color index

qcir – inquire color index range

qcol – inquire color capability

qcr – inquire color representation

qcs – inquire character height in a variety of units

qfs – inquire fill-area style

qhs – inquire hatching style

qid – inquire current device identifier

qinf – inquire PGPLOT general information

qitf – inquire image transfer function

qls – inquire line style

qlw – inquire line width

qpos – inquire current pen position

qtbg – inquire text background color index

qtxt – find bounding box of text string

qvp – inquire viewport size and position

qvsz – find the window defined by the full view surface

qwin – inquire window boundary coordinates

rect – draw a rectangle, using fill-area attributes

rnd – find the smallest 'round' number greater than x

rnge – choose axis limits

sah – set arrow-head style

save – save PGPLOT attributes

unsa – restore PGPLOT attributes

scf – set character font

sch – set character height

sci – set color index

scir – set color index range

scr – set color representation

scrn – set color representation by name

sfs – set fill-area style

shls – set color representation using HLS system

shs – set hatching style

sitf – set image transfer function

slct – select an open graphics device

sls – set line style

slw – set line width

stbg – set text background color index

subp – subdivide view surface into panels

svp – set viewport (normalized device coordinates)

swin – set window

tbox – draw frame and write (DD) HH MM SS.S labelling

text – write text (horizontal, left-justified)

updt – update display

vect – vector map of a 2D data array, with blanking

vsiz – set viewport (inches)

vstd – set standard (default) viewport

wedg – annotate an image plot with a wedge

wnad – set window and adjust viewport to same aspect ratio

pgplotwidget - Constructor

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Create a pgplot widget in a supplied frame

Synopsis
pgplotwidget(parentframe, size, foreground, background, padx, pady, mincolors, maxcolors, havemessages, widgetset)

Description
This constructor creates a PGPLOT widget in a supplied frame. It either returns a tool, or a "fail" if construction isn't possible (for example, a "fail" is returned if no DISPLAY environment variable is set).
The supplied frame should have been created with `side='top'` to make sure that the message line is above the plot area.
If the minimum requested number of colors `mincolors` cannot be supplied, then a fail will be returned. In this case you may wish to ensure that

`parentframe` has a private colormap installed (set `newcmap=newcmap` in the `frame` constructor) and try again.
If you want to, you can suppress the messages window by setting `havemessages` to `F`. You can still call the `message()`, function, however it has no effect.

Arguments

parentframe	frame to attach to Allowed:frame
size	Size in pixels for the drawing area Allowed:integer[2] Default: [600,450]
foreground	Foreground colour Allowed:string (valid colour name Default: white
background	Background colour Allowed:string (valid colour name) Default: black
padx	x border padding in pixels Allowed:float Default: 2
pady	y border padding in pixels Allowed:float Default: 2
mincolors	Minimum number of colors we require Allowed:integer Default: 2
maxcolors	Maximum number of colors we will take Allowed:integer Default: 100
havemessages	Display messages on the screen? Allowed:boolean Default: T
widgetset	The widgetset to use Allowed:A widgetserver tool Default: dws

Returns
pgplotwidget tool or fail

---

pgplotwidget.plotxy - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Plot X,Y vectors as lines or points with auto-scaling

Synopsis
plotxy(x, y, plotlines, newplot, xtitle, ytitle, title, linecolor, ptsymbol, mask)

Description
`plotxy` is defined to allow you to make line or scatter plots. The routine will scale and draw the axes appropriately, or you can specify to plot over a previous plot using the existing scaling. Axis labels and a plot title can be specified. You can also specify a mask, if desired, to blank out bad data points from the plot. When a mask is given, the unmasked data are plotted in line segments.

Example

```
include 'pgplotter.g'
pg := pgplotter();
x := 1:100;
y := x*x;
pg.plotxy(x,y,xtitle='XX', ytitle='YY', title='TITLE');
pg.plotxy(x+10, y, F, F);
```

Example

```
include 'pgplotter.g'
pg := pgplotter();
x := 1:100;
y := sqrt(x);
pgmask := array(T,100)
pgmask[20:30] := F
pgmask[50:55] := F
pg.plotxy(x,y,mask=pgmask);
```

Arguments

x	X values Allowed:non-complex numeric array
y	y values Allowed:non-complex numeric array
plotlines	Plot lines (T) or points (F) Allowed:boolean Default: T
newplot	Start a new plot (T) or overplot (F) Allowed:boolean Default: T
xtitle	x axis title (ignored if newplot is F) Allowed:string Default: none
ytitle	y axis title (ignored if newplot is F) Allowed:string Default: none
title	plot title (ignored if newplot is F) Allowed:string Default: none
linecolor	color index for lines or points Allowed:integer Default: 2
ptsymbol	symbol index (used if plotlines=F) Allowed:integer Default: 2
mask	mask for eliminating bad data points (T for good data, F for masked data) Allowed:boolean array Default: none

Returns
T or fail (illegal argument)

---

pgplotwidget.maskline - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Draw line segments based on input vectors and a mask

Synopsis

maskline(x, y, mask, decimate)

Description

`maskline` is analogous to `line` except that it takes a mask (an array of boolean elements) and draws a series of line segments with the masked data eliminated.
The argument `decimate` is used to speed up the search for masked pixels. It breaks the spectrum into `decimate` segments. It uses built in *Glish* functions to detect whether there are any masked pixels in each segment. If there are, then those masked pixels are found, one by one with the *Glish* indexing operator (slow). If you set `decimate==1` then each pixel of the spectrum is searched for masked pixels with the *Glish* indexing operator (very slow). The default decimation of 10 is safe for any length spectrum (i.e. it will realize if there aren't enough pixels for 10 segments).

Example

```
include 'pgplotter.g'
pg := pgplotter();
x := 1:100;
y := sqrt(x);
pgmask := array(T,100)
pgmask[20:30] := F
pgmask[50:55] := F
pg.env(0,100,0,10,0,0)
pg.maskline(x,y,pgmask)
```

Arguments

x	X values Allowed:non-complex numeric array
y	y values Allowed:non-complex numeric array
mask	mask for eliminating bad data points (T for good data, F for masked data) Allowed:boolean array Default: No mask
decimate	Number of segments to decimate into Allowed:Integer Default: 10

Returns
T or fail

---

pgplotwidget.displaylist - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Access the displaylist plot commands are kept in

Synopsis
displaylist()

Description
This function provides access to the displaylist that the recorded commands are kept in. A reference to the actual display list is returned, so you should be careful that you don't damage it.
You could use this function to, for example, go backwards through the plot command to find the last data plotted, extract it, fit it, and plot the fit to the data.

Returns
displaylist reference

---

pgplotwidget.shortnametofullname - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Get a longer description for pgplotwidget commands

Synopsis
shortnametofullname()

Description
This function takes a shortened Glish/PGPLOT function name and writes out a longer description of the function.

Example

```
- include 'pgplotter.g'
- f             :=frame();
- myplotwidget := pgplotwidget(f);
- field_names(myplotwidget);
shortnametofullname displaylist setcallback deactivatecallback
activatecallback record play postscript plotfile restore refresh done
plotxy clear message settings cursor lastchange curs size addredrawfunction
arro ask bbuf bin box circ conb conl cons cont ctab draw ebuf env eras errb
errx erry gray hi2d hist iden imag lab ldev len line move mtxt numb page
panl pap pixl pnts poly pt ptxt qah qcf qch qci qcir qcol qcr qcs qfs qhs
qid qinf qitf qls qlw qpos qtbg qtxt qvp qvsz qwin rect rnd rnge sah save
scf sch sci scir scr scrn sfs shls shs sitf sls slw stbg subp svp swin tbox
text unsa updt vect vsiz vstd wedg wnad
- myplotwidget.shortnametofullname('scf');
set character font
- myplotwidget.shortnametofullname('draw');
draw a line from the current pen position to a point
```

Returns
function description, if present, otherwise echoes argumentname

---

pgplotwidget.setcallback - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

set a function to be called for mouse or keyboard events

Synopsis
setcallback(name, callback)

Description
This function allows a function of yours to be called whenever a certain event happens. It is a simplification of the event-binding described in the Glish/PGPLOT chapter of the Glish manual.
The function returns an index which you can use to deactivate the callback.
At present, the events for which you may set a callback are:

motion Called whnenver the mouse moves in the plot area.

button Called whenever any button goes *down*.

button1 Called whenever button 1 goes *down*.

button2 Called whenever button 2 goes *down*.

button3 Called whenever button 3 goes *down*.

buttonup Called whenever a button goes *up*.

key Called whenever a key is pressed on the keyboard.

The callback functions take a single argument, which will be a record of the type described in the manual. Or of course you could just pass in the callback function:

```
callback := function(rec) {print rec;}
```

to discover the record structure. Note that to follow a drag you merely follow the motion after a button down and before a button up.

Example
For example, here is how you could implement a little "scribble" application.

```
include 'pgplotwidget.g'
f := frame();
pg := pgplotwidget(f);
pos := [dragging=F];

# Drag starts
downcallback := function(rec) {
    global pos;
    pos := [dragging=T, xlast=rec.world[1], ylast=rec.world[2]];
}

# Draw the new line segment if we are dragging
motioncallback := function(rec) {
    global pos;
    if (pos.dragging) {
        x := rec.world[1]; y := rec.world[2];
        pg.line([pos.xlast, x], [pos.ylast, y]);
        pos.xlast := x; pos.ylast := y;
    }
}

# Drag ends
```

```
upcallback := function(rec) {
    global pos;
    pos.dragging := F;
}

pg.env(0, 1, 0, 1, 0, -1);
pg.setcallback('button', downcallback);
pg.setcallback('motion', motioncallback);
pg.setcallback('buttonup', upcallback);
pg.message('Start scribbling! Drag the mouse with a button');
```

The amount of code could have been reduced somewhat by having only one callback for both button and buttonup, and using the "ButtonPress" field of the record to distinguish up from down events.

Arguments

name	events for which to call supply callback Allowed:motion,key,button,button[1-3],buttonup
callback	function to call on event Allowed:single argument function

Returns
integer index (for deactivatecallback)

---

pgplotwidget.deactivatecallback - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

deactivate a callback

Synopsis
deactivatecallback(which)

Description
This function gives you a way to deactivate a previously defined callback.

Arguments

which	callback index to remove Allowed:integer

Returns
T or fail (for an invalid index)

---

pgplotwidget.activatecallback - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

activate a callback

Synopsis
activatecallback(which)

Description
This function gives you a way to activate a previously defined callback.

Arguments

which	callback index to reactivate Allowed:integer

Returns
T or fail (for an invalid index)

---

pgplotwidget.getcallbacks - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Recover callbacks

Synopsis

getcallbacks()

Description
This function recovers a record with one entry per set callback. Each entry holds two fields, 'index' and 'active'. The 'index' is the index returned for that callback by function setcallback. Field 'active' says whether that callback is active or not.

Returns
A record

---

pgplotwidget.record - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Save the drawing commands

Synopsis
record(newstate)

Description
This function gives you control of whether or not the plotting commands are saved in the internal display list or not. You might want to do this for efficiency concerns (e.g. the commands are coming frequently and contain a lot of pixels), or so that you can "undraw" some temporary plotting with a refresh.
The return value is the *previous* state of the function to make it easy to get back to the previous state.

Example

```
- pg.record(F); # Turn off saving plot commands
- ... temporary ``decoration''
- pg.refresh()  # Make the temporary stuff go away.
- pg.record(T)  # Start recording again
```

Arguments

newstate	Record or not? Allowed:boolean

Returns
boolean - the former recording state

---

pgplotwidget.recording - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

return whether plot commands are being saved

Synopsis
recording()

Description
The pgplotmanager tool maintains a record of plot commands that have been issued in the form of a displaylist tool. This allows this tool to replay the commands later to, for example, refresh the screen or send a currently viewed plot to a PostScript file.
This function simply returns whether plot commands are currently being saved internally for later replay. A returned value of `T` indicates that recording is currently turned on and commands are being saved. The recording state can be changed via the record() function.

Returns
boolean – the current recording state

---

pgplotwidget.canplay - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

return whether a command can be executed

Synopsis
canplay()

Description
This function returns true if the given command is recognized as a supported plot command. To be recognized, the `command` parameter must be the name of a plotting function of this tool (either a "raw" plotting command supported by the Glish/PGPLOT agent or higher level command, like settings() or plotxy()) or a custom command that has been added via the addredrawfunction() function (usually as part of a plug-in).

Example
Consider this session in which no custom commands have been added:

```
- print pg.canplay('plotxy');
T
- print pg.canplay('gray');
T
- print pg.canplay('grayscale');
F
```

---

pgplotwidget.play - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Execute a series of plot commands

Synopsis
play(commands, record)

Description
This function is used to execute a series of plot commands in this plotter. It is used internally (e.g. by refresh), but it can also be of use to programmers. For example, you could keep external "decorations" in a list, and show or don't show them by calling `play`.
The format of the commands is that it is either a `displaylist` from a `pgplotwidget`, or it is a record, each field of which is a plot command of the form, e.g.:

```
[_function='line', xpts=1:10, ypts=1:10]
```

Arguments

commands	list of plot commands to execute Allowed:displaylist or record
record	save commands in our internal displaylist? Allowed:boolean

Returns
T or fail

---

pgplotwidget.postscript - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Save the plot as a PostScript file

Synopsis
postscript(file, color, landscape)

Description
This function saves the plot as a postscript file. Basically it changes the device to a PostScript device and then does a refresh, so any commands that have not been recorded will not be plotted. If the file already exists (and is writable), it will be overwritten without any request for confirmation.

By default it writes in landscape orientation and color PostScript, but these can be changed to portrait and BW respectively.

Arguments

file	file name Allowed:string Default: aipsplot.ps
color	write in color postscript? Allowed:boolean Default: T
landscape	write landscape? Allowed:boolean Default: T

Returns
T or fail (e.g., file not writable)

---

pgplotwidget.plotfile - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Save the plot in native format

Synopsis
plotfile(file)

Description
This function saves the plot as an AIPS++ Table. The plot can then be reopened and new commands added or old commands changed. Basically it saves the internal display list, so any commands that have not been recorded will not be saved. If the file already exists (and is writable), it will be overwritten without any request for confirmation.

Arguments

file	file name Allowed:string Default: aips++plot.plot

Returns
T or fail (e.g., file not writable)

---

pgplotwidget.restore - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Retrieve a plot

Synopsis
restore(file)

Description
This function retrieves a plot file that has been saved by `plotfile`. The current plot is replaced with the plot in the file. If you want to merge two plot files you can use `play` on the displaylist of another `pgplotwidget`.

Arguments

file	file name Allowed:string Default: aips++plot.plot

Returns
T or fail (e.g., file not readable)

---

pgplotwidget.refresh - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

replay the display list

Synopsis
refresh()

Description
This function redraws the plot. The most common use for it is to redraw the plot after the frame size has been changed.

Example

```
- f := frame()
- pg := pgplotwidget(f);
- whenever f->resize do pg.refresh();
```

Arguments

Returns
T or fail

---

pgplotwidget.resetplotnumber - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Reset internal plot number counter to 0

Synopsis
resetplotnumber()

Returns
T

Description

The pgplotwidget maintains an internal counter which is used to determine when to prompt the user (when function ask(T) has been called). This function resets the counter to 0, which means for the next plot, you won't be prompted.

---

pgplotwidget.size - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

size in pixels of the pgplotwidget

Synopsis
size()

Description
Returns the size in pixels of the pgplot frame.

Returns
integer[2]

---

pgplotwidget.clear - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

empty the display list

Synopsis
clear()

Description
This function empties the display list. This is unlike `eras`, which merely clears the current page. This function erases the screen and removes all accumulated plot commands.

---

pgplotwidget.message - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

write in the message line

Synopsis
message(message)

Description
This function sends a message to the message line above the plot area. It is intended for messages which instruct the user in driving the application the widget is embedded in. For example, use it to relay what the mouse or keyboard events are expected.
If you send a message that is more than one line long (i.e. if it has embedded newline characters), a button will appear to the right of the message. Pressing it will display the entire message in a popup window. You should try to arrange multi-line messages so that the first line contains a summary, and subsequent lines contain more detailed instructions. This way frequent users won't have to pop up a new window very often.
Clear the line by sending it an empty string.

Arguments

message	message to write Allowed:string

---

pgplotwidget.settings - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Set many PGPLOT attributes at once

Synopsis
settings()

Description
This function is used to set (possibly) many PGPLOT attributes at once. It is always inserted at the beginning of every plot's display list to make it easy to change settings in a global way. All values have their PGPLOT defaults.

```
ask                             : page prompting
nxsub, nysub                    : plot panelling
arrowfs, arrowangle, arrowvent  : arrow style
font                            : text font
ch                              : character height
ci                              : color index
fs                              : fill style
hsangle, hssepn, hsphase        : hatching style
```

```
ls                               : line style
lw                               : line width
tbci                             : text background color
```

---

pgplotwidget.cursor - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Set the cursor

Synopsis
cursor()

Description
This function is used to set the cursor's reference position, the cursor position and the cursor color. See the Glish/PGPLOT chapter for details on its use.

---

pgplotwidget.lastchange - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Version of the last change

Synopsis
lastchange()

Description
This function returns a vaule that changes every time the internal display list changes. It can be used, for example, to see if the plot has changed since the last time it was inspected.

Returns
integer - changes every time the displaylist changes

---

pgplotwidget.curs - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Get a key and position

Synopsis
curs(x, y)

Description
This function emulates the standard PGPLOT curs() event. It returns a record of the form:

```
[ok=bool, x=float, y=float, ch=string]
```

The "ch" field is the key that the user struck on the keyboard. Note that it *BLOCKS* until a key is pressed on the plot area. Setting a callback is usually more flexible than this routine since it tolerates inaction on the users part better.
The function may be passed a position where the programmer guesses that the user will want to position the cursor. It defaults to (0,0).

Arguments

x	estimated X position Allowed:float Default: 0
y	estimated Y position Allowed:float Default: 0

Returns
record

---

pgplotwidget.addredrawfunction - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

Add a function to redraw a command in the displaylist

Synopsis
addredrawfunction(name, redrawfunction)

Description
This function is only of interest if you want to add functions via plugins. Suppose you create via a plugin a high-level plot command that you want to show up in the displaylist as a high-level command (*i.e.*, not broken up into lower level PGPLOT commands). In this case, in your plugin you would enter into the displaylist a record with a field per argument, and a field `_function` that contains the name of the function.
To be able to redraw the function, you would use `addredrawfunction` to add a function which, when passed the record the plugin inserted in the display list, would ensure that the draw commands are executed.

Arguments

name	name of the draw function Allowed:string
redrawfunction	function to redraw a command from the displaylist Allowed:function

Returns
T or fail

See Also

1. Glish/PGPLOT
2. pgplotter
3. displaylist
4. plotter

pgplotwidget.done - Function

Package	**display**
Module	**plotter**
Tool	**pgplotwidget**

destroy the widget, release resources

Synopsis
done()

Description
This function destorys the widget and attempts to release all its resources. Note in particular that it destroys the parent frame the widget was created with.

1.2.3 pgplotter - Tool

Package	**display**
Module	**plotter**

stand alone pgplotter tool
include "pgplotter.g"

Constructors

pgplotter	Create a pgplotter tool in standalone GUI

Functions

addtool	Add a tool to the Tools menu
busy	Set or unset the busy symbol to the mouse cursor
commandgui	Create the "add commands" GUI
demo	Demo set for PGPLOT
done	Close the pgplotter tool
editgui	Create the "edit commands" GUI
gui	Call back the plotter window
plotxy1	Plot X and Y vectors with auto-scaling
plotxy2	Plot X and Y vectors with auto-scaling using the right and upper axes.
ploty	Plot Y vectors with auto-scaling
ploty2	Plot Y vectors with auto-scaling using the right Y axis
psprint	Print out all of the plots in the display list
psprinttofile	Print out all of the plots in the display list to the named file
screen	Dismiss the plotter window
setcolor	Set the color of subsequent plotting elements (e.g. lines, points, etc).
setplottitle	Set the title of the plot
setxaxisgrid	Turn on or off the display of the X-axis grid lines
setxaxislabel	Set the x-axis label

setxscale	Set the viewable X range of an already existing plot
sety2axisgrid	Turn on or off the display of the Y-axis grid lines for the right Y axis.
sety2scale	Set the viewable Y range of an already existing plot using the right Y axis
setyaxisgrid	Turn on or off the display of the Y-axis grid lines
setyaxislabel	Set the y-axis label
setyscale	Set the viewable Y range of an already existing plot
timey	Plot Y vectors versus time with auto-scaling
timey2	Plot Y vectors versus time with auto-scaling against the right and upper axes.
title	Set the pgplotter window title
tool	Show or hide a particular Tool
type	Return the type of object, in this case "pgplotter". This function is necessary for the toolmanager
userframe	Return an internal user Glish/Tk frame

Description

A "pgplotter" tool is used to manipulate line and raster plots. It is layered on top of both the Glish/PGPLOT commands and the pgplotwidget and so has a superset of those commands, in addition to its own functionality.
Plot commands can be entered both from the Glish command line, and also from the GUI. The plot can be saved to either a "plot" file or a PostScript file. The advantage of the former is that at a later time you can reopen the file and change or add to the plot. Once a PostScript file is written, it cannot be changed from this tool, it can only be overwritten.
The GUI consists of four graphical areas. The first is the menu bar, which has the following menus:

File From the file menu you can do the following:

- **Open pgplot file** This menu item will present you with a file chooser from which you can open a new plot file. If the file already exists the existing plot commands will be displayed on the screen.
- **Save pgplot file** This will save the plot file to disk. The name of the plot file is shown in the lower left corner of the GUI. It will overwrite any commands that are already in the plot file.
- **Save to postscript file** This will save the plot file to disk in postscript format.
- **Print** Send a file to the printer. It is possible to send the plot directly to ghistview for a preview, also.
- **Dismiss** Exit the pgplotter GUI, but leave the tool alive.
- **Done** Exit and destroy the pgplotter tool.

Tools This menu includes two items at present. Additional tools can be added by the user via the addtool function. That is, it is intended that users and sites might customize (*e.g.*, add an analysis facility) by adding an item to the tools menu via the plugins facility. The standard tools which are available are:

Cursor This tool can be used to change the cursor to crosshair style, which can facilitate reading data values from the plot.

Colormap If the plot has installed a colormap with the `ctab` function, you may manipulate the colormap with this tool. From the raised menu button you can choose the style of colormap you like, and then in the plot surface "drag" the cursor (left-button down, and move it without releasing it) until you like the colormap. Vertical movement of the cursor changes the "contrast" of the cursor (how tightly bunched the colors are), and horizontal movement changes the "brightness" (the pixel value where the colors are displayed). That is, in the `ctab` command, vertical motion controls the `contra` parameter, and horizontal motion controls the `bright` parameter. It is useful to have a color wedge to see the effects of the cursor motion. One may be inserted with the `wedg` command.

Edit The Edit menu is used to add new commands to the plot, or to change commands which have already been sent to the plotter. It has only one menu item:

Edit This command lets you change or add plot commands. For example, you might want to change a line thickness, or change the number of panels displayed per page, or change the axes to logarithmic.

If you want to change a plot command, click on the command you want to change in the Drawlist, change the parameters in the input box, and then press Return. The change should be reflected immediately in the plot.

To add a new pgplot drawing command, select the desired command from the functions list and then click on `Add to Drawlist`. The Clipboard is also available for adding several commands simultaneously.

If you want to remove a single command from the Drawlist, use `Delete`, and to clear the entire Drawlist use `Clear`.

Help Several options are available including directing your web browser to appropriate help sections, asking a question, and reporting a bug.

Below the menu bar is a one line "message" window which can be writtin to via the `message` function. It is intended primarily for things like informing the user which mouse buttons to use in the plot window. If the message contains more than one line, a button will appear beside the message. Pressing it will display the entire message in a popup window.
Next down is the plot frame, where all the plotting action happens. If there are enough available colors, it is possible to do color (raster and line) plots in

this window. If some other application has grabbed all the colours, then only black and white plots will be possible.
The bottom line has several items of interest. From left to right:

1. The name of the plot file. If the plot has unsaved changes, then the name of the plot file will be "raised", and you can press the button to save it.
2. The Clear button erases the screen and removes all commands from the draw history. It does not, however, empty the plot file on disk (the file name will raise after clear is pressed - if you then save the file to disk the plot will effectively be gone).
3. The central pair of numbers in a "groove" gives the current mouse position in plot coordinates. If there are several panels on the page the coordinates are relative to the last panel plotted, which may not be what you expect!
4. If the dismiss button is pressed, the GUI and its contents will go away. You will be given a chance to save unsaved changes first. This is equivalent to executing the `done` function.

The pgplotter class is enabled for plugins. This means that you can write your own "plugin" (.gp file) code and it will automatically be available to the plotter. For instance, you could set up a GUI to read data, fit a model to it, and plot the fit and residuals. this procedure will be documented at a later time.
In general, a pgplotter is just a shell around an underlying pgplotwidget. All functions of pgplotwidget are available in the pgplotter. The only functions documented here are those that are unique to pgplotter.

Example

```
- include 'pgplotter.g'
- pg := pgplotter()
- pg.env(0, 2*pi, -1, 1, 0, 0)
- x := 2*pi*[1:100]/100; y := sin(x);
- pg.line(x,y);
```

See Also

1. Glish/PGPLOT
2. pgplotwidget
3. displaylist
4. plotter

pgplotter - Constructor

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Create a pgplotter tool in standalone GUI

Synopsis
pgplotter(plotfile, size, foreground, background, mincolors, maxcolors)

Description
This command creates a pgplotter tool with a GUI, and attaches it to a plot file. If the plot file is filled, the plot commands are drawn to the screen. If there are any plugins (at present there is one plugin for displaying PGPLOT demonstrations), they are attached before the tool is returned to the user.
If it is not possible to create the tool (for example, no DISPLAY environment variable is set), a fail is returned.
If the requested number of colors `mincolors` cannot be allocated with a shared colormap, a private colormap will be installed. This will cause colormap flashing on some hardware.

Arguments

plotfile	Name of the plot file to attach to Allowed:string Default: aips++plot.plot
size	Size in pixels for the drawing area Allowed:integer[2] Default: [600,450]
foreground	Foreground colour Allowed:string (valid colour name Default: white
background	Background colour Allowed:string (valid colour name) Default: black
mincolors	Minimum number of colors we require Allowed:integer Default: 2
maxcolors	Maximum number of colors we will take Allowed:integer Default: 100

Returns
pgplotter tool or fail

---

pgplotter.commandgui - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Create the "add commands" GUI

Synopsis
commandgui()

Description
This command starts up the "Add commands" GUI. It is unlikely you will call this function directly. Instead you are apt to invoke it from the "Edit" menu of the pgplotter tool.

---

pgplotter.editgui - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Create the "edit commands" GUI

Synopsis
editgui()

Description
This command starts up the "Edit commands" GUI. It is unlikely you will call this function directly. Instead you are apt to invoke it from the "Edit" menu of the pgplotter tool.

---

pgplotter.addtool - Function

Package	**display**

Module plotter
Tool pgplotter

Add a tool to the Tools menu

Synopsis
addtool(name, start, suspend)

Description
This command is only of interest if you want to customize the `Tools` menu.
This command creates a "Tool" - a custom GUI that the user can interact with to perform some functionality. The tool might be used purely to manipulate the plot (for example the standard zoom tool), or it might be used to perform some data analysis.
Typically the tool will operate by finding some plot commands in the displaylist, and then changing or adding to them appropriately.
The tool is defined by supplying a name and two functions, one of which is called when the tool is invoked and the other of which is called when the tool is dismissed.
The arguments to these `start` nad `suspend` functions are:

frame A Glish/Tk frame in which the Tool may put whatever buttons or other widgets it needs.

public The `pgplotter` tool which is calling the Tool. Typically the Tool will use this to go through the displaylist and add additional plot commands.

state This is a record which the Tool can use as a "scratchpad" to retain information between invocations. If the length of the state record is zero, then the Tool has not been initialized. Typically the Tool will put its GUI widgets (e.g. buttons in the state record) as well as state that needs to be remembered from invocation to invocation.

Typically a tool will be defined in a plugin.

Arguments

name	name of the tool Allowed:string
start	function to call when the Tool GUI becomes visible Allowed:function
suspend	function to call with the Tool GUI is dismissed Allowed:function

Returns
T or fail (if the tool has already been defined)

---

pgplotter.tool - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Show or hide a particular Tool

Synopsis
tool(name, show)

Description
This command is only of interest if you want to start or hide a Tool from the Glish command line.
The tool must have already been defined with `addtool`.

Arguments

name	name of the tool Allowed:string
show	show or hide the tool Allowed:boolean Default: T

Returns
T or fail (if the tool does not exist)

---

pgplotter.busy - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Set or unset the busy symbol to the mouse cursor

Synopsis
busy(isbusy)

Description
This changes the mouse cursor to an X 'watch' cursor to signify that the plotter is busy. Remember to switch it back to not busy when finished.

Arguments

isbusy	Is the plotter busy? Allowed:boolean Default: T

Returns
Boolean

---

pgplotter.demo - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Demo set for PGPLOT

Synopsis
demo()

Description
This brings up a Glish/Tk widget holding many buttons for demonstrating different types of plots in PGPLOT

---

pgplotter.done - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Close the pgplotter tool

Synopsis
done()

Description
The `done` command closes the pgplotter tool and removes it from memory. It will *not* ask you if you want to store any unsaved changes first.

---

pgplotter.gui - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Call back the plotter window

Synopsis
gui()

Description
If the pgplotter is dismissed with the "Dismiss" button. It may be recalled with the gui function which remaps the frame onto the screen. It is the reverse function of "screen" which unmaps the pgplotter window (the CLI version of the Dismiss button).

---

pgplotter.screen - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Dismiss the plotter window

Synopsis
screen()

Description
This dismisses the plotter window. It may be recalled with the "gui" command.

---

pgplotter.title - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Set the pgplotter window title

Synopsis
title(msg)

Arguments

msg	Text of the new window title Allowed:string	

Returns
boolean

---

pgplotter.userframe - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Return an internal user Glish/Tk frame

Synopsis
userframe()

Description

This function returns one of the Pgplotter's internal frame into which you can embed your own GUI elements. These elements will appear directly below the plotting area and above the buttons at the bottom.

Returns
Glish/Tk agent

The following commands are included through the plugin file, pgplotter_gplot1d.gp. See also the Getting Results in AIPS++ section on "Line Plots with Pgplotter in AIPS++".

pgplotter.ploty - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Plot Y vectors with auto-scaling

Synopsis
ploty(y, xlab, ylab, tlab, plotlines, ptsymbol)

Arguments

y	A collection of Y vectors Allowed:float
xlab	A label for the X axis Allowed:string
ylab	A label for the Y axis Allowed:string
tlab	A label for the plot title Allowed:string
plotlines	Toggle for plotting lines or points Allowed:Boolean Default: T (lines)
ptsymbol	Symbol ID Allowed:integer Default: 2

Returns
boolean

Description
This `gplot1d`-emulator plots a collection of Y vectors with auto-scaling. If there is more than one Y vector, they should be arranged in a 2-D array with the first dimension being the number of vectors to plot and the second dimension being the length of each vector. The X axis will default to an index number starting at 1. The color of each Y vector on the plot will be different for the first 16 plots, after that the colors will cycle through the same 16 colors.

Example

```
include 'pgplotter.g'
pg := pgplotter();
x := 1:100;
y := sqrt(x);
pg.ploty(y,'X','Y','TITLE');
pg.ploty(y*2,,'Y2');
```

---

pgplotter.plotxy1 - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Plot X and Y vectors with auto-scaling

Synopsis
plotxy1(x, y, xlab, ylab, tlab, plotlines, ptsymbol)

Arguments

x	An X vector Allowed:float
y	A collection of Y vectors Allowed:float
xlab	A label for the X axis Allowed:string
ylab	A label for the Y axis Allowed:string
tlab	A label for the plot title Allowed:string
plotlines	Toggle for plotting lines or points Allowed:Boolean Default: T (lines)
ptsymbol	Symbol ID Allowed:integer Default: 2

Returns
boolean

Description
This gplot1d-emulator plots a collection of Y vectors with auto-scaling against a single X vector. If there is more than one Y vector, they should be arranged in a 2-D array with the first dimension being the number of vectors to plot and the second dimension being the length of each vector. The length of the X axis vector must be the same as the length of each Y vector. The color of each Y vector on the plot will be different for the first 16 plots, after that the colors will cycle through the same 16 colors. The ylabflag determimes whether the y axis label is printed on the plot itself, in addition to the y axis. For a single plot, this may sensibly be set to false. However, if plotxy1 is called repeatedly without clearing the plot, the successive calls will overplot the new data on the original, and having this flag set to true is useful for determining which color plot corresponds to which data. This function is called plotxy1 to distinguish it from the standard pgplotter function, plotxy, which has some subtle differences.

Example

```
include 'pgplotter.g'
pg := pgplotter();
x := 1:100;
y1a := sin(x/10);
y1b := cos(x/10);
y2 := sqrt(x);
pg.plotxy1(x,y1a,'X Index','Sine Function','Title');
pg.plotxy1(x,y1b,,'Cosine Function');
pg.plotxy2(x,y2,,'SQRT Function');
pg.ptxt(105,5.5,270,0.5,'SQRT (X)');
```

---

pgplotter.ploty2 - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Plot Y vectors with auto-scaling using the right Y axis

Synopsis
ploty2(y, xlab, ylab, tlab, plotlines, ptsymbol)

Arguments

y	A collection of Y vectors Allowed:float
xlab	A label for the X axis Allowed:string
ylab	A label for the Y axis Allowed:string
tlab	A label for the plot title Allowed:string
plotlines	Toggle for plotting lines or points Allowed:Boolean Default: T (lines)
ptsymbol	Symbol ID Allowed:integer Default: 2

Returns
boolean

Description
This `gplot1d`-emulator plots a collection of Y vectors with auto-scaling using the right Y axis (which will have separate labels from the left Y axis). If there is more than one Y vector, they should be arranged in a 2-D array with the first dimension being the number of vectors to plot and the second dimension being the length of each vector. The X axis will default to an index number starting at 1. The color of each Y vector on the plot will be different for the first 16 plots, after that the colors will cycle through the same 16 colors. The order of these colors is the reverse order of that used in the ploty and plotxy1 function.

Example

```
include 'pgplotter.g'
pg := pgplotter();
x := 1:100;
pg.ploty(sqrt(x),'X','Y','TITLE');
pg.ploty2(sin(x/3),,'Y2');
```

---

pgplotter.psprint - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Print out all of the plots in the display list

Synopsis
psprint()

Description
This function will execute your display list and print out all plots to the printer. You should use the edit tool to examine your displaylist before printing.

---

pgplotter.psprinttofile - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Print out all of the plots in the display list to the named file

Synopsis
psprinttofile(filename)

Description
This function will execute your display list and print out all plots to a file. You should use the edit tool to examine your displaylist before printing.

Arguments

filename	filename on disk to hold the plot session Allowed:string

---

pgplotter.plotxy2 - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Plot X and Y vectors with auto-scaling using the right and upper axes.

Synopsis
plotxy2(x, y, xlab, ylab, tlab, plotlines, ptsymbol)

Arguments

x	An X vector Allowed:float
y	A collection of Y vectors Allowed:float
xlab	A label for the X axis Allowed:string
ylab	A label for the Y axis Allowed:string
tlab	A label for the plot title Allowed:string
plotlines	Toggle for plotting lines or points Allowed:Boolean Default: T (lines)
ptsymbol	Symbol ID Allowed:integer Default: 2

Returns
boolean

Description
This `gplot1d`-emulator plots a collection of Y vectors with auto-scaling against a single X vector using the right and upper axes. These axes will have separate labelling from the lower and left axes. If there is more than one Y vector, they should be arranged in a 2-D array with the first dimension being the number of vectors to plot and the second dimension being the length of each vector. The length of the X axis vector must be the same as the length of each Y vector. The color of each Y vector on the plot will be different for the first 16 plots, after that the colors will cycle through the same 16 colors. The order of these colors is the reverse order of that used in the ploty and plotxy1 functions.

Example

```
include 'pgplotter.g'
pg := pgplotter();
```

```
x := 1:100;
y1 := sin(x/10);
y2 := sqrt(x);
pg.plotxy1(x,y1,'X Index','Sine Function','Title');
pg.plotxy2(x,y2,,'SQRT Function');
pg.ptxt(105,5.5,270,0.5,'SQRT (X)');
```

---

pgplotter.timey - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Plot Y vectors versus time with auto-scaling

Synopsis
timey(x, y, xlab, ylab, tlab, plotlines, ptsymbol)

Arguments

x	A vector of times Allowed:float
y	A collection of Y vectors Allowed:float
xlab	A label for the X axis Allowed:string
ylab	A label for the Y axis Allowed:string
tlab	A label for the plot title Allowed:string
plotlines	Toggle for plotting lines or points Allowed:Boolean Default: T (lines)
ptsymbol	Symbol ID Allowed:integer Default: 2

Returns
boolean

Description
This `gplot1d`-emulator plots a collection of Y vectors against a single X vector. The X vector is assumed to be time in seconds. If there is more than one Y vector, they should be arranged in a 2-D array with the first dimension being the number of vectors to plot and the second dimension being the length of each vector. The length of the X axis vector must be the same as the length of each Y vector. The color of each Y vector on the plot will be different for the first 16 plots, after that the colors will cycle through the same 16 colors.

---

pgplotter.timey2 - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Plot Y vectors versus time with auto-scaling against the right and upper axes.

Synopsis
timey2(x, y, xlab, ylab, tlab, plotlines, ptsymbol)

Arguments

x	An vector of times Allowed:float
y	A collection of Y vectors Allowed:float
xlab	A label for the X axis Allowed:string
ylab	A label for the Y axis Allowed:string
tlab	A label for the plot title Allowed:string
plotlines	Toggle for plotting lines or points Allowed:Boolean Default: T (lines)
ptsymbol	Symbol ID Allowed:integer Default: 2

Returns
boolean

Description
This `gplot1d`-emulator plots a collection of Y vectors against a single X vector using the right and upper axes (which will have separate labels from the other axes). The X vector is assumed to be time in seconds. If there is more than one Y vector, they should be arranged in a 2-D array with the first dimension being the number of vectors to plot and the second dimension being the length of each vector. The length of the X axis vector must be the same as the length of each Y vector. The color of each Y vector on the plot will be different for the first 16 plots, after that the colors will cycle through the same 16 colors. The order of these colors is the reverse order of that used in the ploty and plotxy1 functions.

---

pgplotter.type - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Return the type of object, in this case "pgplotter". This function is necessary for the toolmanager

Synopsis
type()

---

pgplotter.setxscale - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Set the viewable X range of an already existing plot

Synopsis
setxscale(xmin, xmax)

Arguments

xmin	The minimum X value to show Allowed:float
xmax	The maximum X value to show Allowed:float

Returns
boolean

Description
This `gplot1d`-emulator function must be used after a ploty, plotxy1, or timey has already been called. It resets the viewable X range and refreshes the display. This alters the displaylist in the pgplotter.

---

pgplotter.setyscale - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Set the viewable Y range of an already existing plot

Synopsis
setyscale(Ymin, Ymax)

Arguments

Ymin	The minimum Y value to show Allowed:float
Ymax	The maximum Y value to show Allowed:float

Returns
boolean

Description
This `gplot1d`-emulator function must be used after ploty, plotxy1, or timey has already been called. It resets the viewable Y range and refreshes the display. This alters the displaylist in the pgplotter.

---

pgplotter.sety2scale - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Set the viewable Y range of an already existing plot using the right Y axis

Synopsis
sety2scale(ymin, ymax)

Arguments

ymin	The minimum Y value to show Allowed:float
ymax	The maximum Y value to show Allowed:float

Returns
boolean

Description
This `gplot1d`-emulator function must be used after ploty2, plotxy2, or timey2 has already been called. It resets the viewable Y range and refreshes the display. This alters the displaylist in the pgplotter.

---

pgplotter.setxaxisgrid - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Turn on or off the display of the X-axis grid lines

Synopsis
setxaxisgrid(on)

Arguments

on	Turn the grid on? Allowed:boolean Default: T

Returns
boolean

Description
This `gplot1d`-emulator function must be used after ploty, plotxy, or timey has already been called. It turns on or off the display of the X-axis grid lines.

---

pgplotter.setyaxisgrid - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Turn on or off the display of the Y-axis grid lines

Synopsis
setyaxisgrid(on)

Arguments

on	Turn the grid on? Allowed:boolean Default: T

Returns
boolean

Description
This `gplot1d`-emulator function must be used after ploty, plotxy, or timey has already been called. It turns on or off the display of the Y-axis grid lines.

---

pgplotter.sety2axisgrid - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Turn on or off the display of the Y-axis grid lines for the right Y axis.

Synopsis
sety2axisgrid(on)

Arguments

on	Turn the grid on? Allowed:boolean Default: T

Returns
boolean

Description
This `gplot1d`-emulator function must be used after ploty2, plotxy2, or timey2 has already been called. It turns on or off the display of the Y-axis grid lines for the right Y-axis.

---

pgplotter.setcolor - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Set the color of subsequent plotting elements (e.g. lines, points, etc).

Synopsis
setcolor(color)

Arguments

color	The PGPLOT color number to draw. Allowed:integer

Description
1=white, 2=red, 3=green, 4=blue, 5=cyan, 6=purple, 7=yellow, 8=orange, 9=light green, 10=blue-green, 11=light blue, 12=dark purple, 13=strawberry, 14+=shades of grey

Returns
boolean

---

pgplotter.setplottitle - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Set the title of the plot

Synopsis
setplottitle(title)

Arguments

title	A title to display Allowed:string

Returns
boolean

---

pgplotter.setxaxislabel - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Set the x-axis label

Synopsis
setxaxislabel(xlabel)

Arguments

xlabel	An x-axis label to display Allowed:string

Returns
boolean

---

pgplotter.setyaxislabel - Function

Package	**display**
Module	**plotter**
Tool	**pgplotter**

Set the y-axis label

Synopsis
setyaxislabel(ylabel)

Arguments

ylabel	An y-axis label to display Allowed:string

Returns
boolean

1.2.4 pgplottertest - Function

Package	**display**
Module	**plotter**

Test pgplotter by drawing the examples

Synopsis
pgplottertest(autodestruct)

Description
This function creates a pgplotter and runs all 13 PGPLOT example plots non-interactively.

Arguments

autodestruct	Destroy the pgplotter immediately after the test Allowed:Boolean Default: T

Returns
Boolean