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**