AIPS++ User Reference Manual - Display Package

AIPS++ Group, eds

October 15, 2006

Copyright ©1995-1999 AIPS++ Consortium

AIPS++ User Reference Manual - Display Package.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the AIPS++ Consortium.

The AIPS++ consortium may be reached by email at aips2-request@nrao.edu. The postal address is: AIPS++ Consortium, c/o NRAO, 520 Edgemont Rd., Charlottesville, Va. 22903-2475 USA.

Contents

1 Display **7**

1.1 guiutils - Module . . . 7
1.1.1 choice - Function . . . 8
1.1.2 donechoice - Function . . . 9
1.1.3 exitchoice - Function . . . 10
1.1.4 filechooser - Function . . . 11
1.1.5 datachooser - Function . . . 13
1.1.6 tablechooser - Function . . . 14
1.1.7 chooser - Function . . . 15
1.1.8 guiframework - Tool . . . 16
1.1.9 gui - Tool . . . 30
1.1.10 status_line - Tool . . . 33
1.1.11 messagebox - Tool . . . 36
1.1.12 single_entry_dialog_box - Tool . . . 38
1.1.13 evalwindow - Tool . . . 40
1.1.14 tracewindow - Tool . . . 43
1.1.15 progress - Tool . . . 48
1.1.16 displaylist - Tool . . . 51
1.1.17 displaylisttest - Function . . . 64
1.1.18 popuphelp - Function . . . 65
1.1.19 popupremove - Function . . . 66
1.1.20 popupmenu - Function . . . 68
1.1.21 addpopuphelp - Function . . . 69
1.1.22 barchart - Tool . . . 70
1.1.23 choicewindow - Function . . . 76
1.1.24 infowindow - Function . . . 77
1.2 plotter - Module . . . 78
1.2.1 pgplotmanager - Tool . . . 79
1.2.2 pgplotwidget - Tool . . . 106
1.2.3 pgplotter - Tool . . . 134
1.2.4 pgplottertest - Function . . . 160
1.3 viewer - Module . . . 162
1.3.1 viewer - Tool . . . 168
1.3.2 viewercanvasmanager - Tool . . . 186

1.3.3 viewercanvasprintmanager - Tool . . . 194
1.3.4 viewercolormapmanager - Tool . . . 201
1.3.5 viewerdisplaydata - Tool . . . 213
1.3.6 viewerdisplaypanel - Tool . . . 247
1.3.7 viewerslicedp - Tool . . . 288
1.3.8 viewerstandardgui - Tool . . . 292
1.3.9 viewerdatamanager - Tool . . . 297
1.3.10 viewerannotations - Tool . . . 298
1.3.11 skycatalog - Tool . . . 311
1.3.12 is_viewer - Function . . . 319
1.4 widgets - Module . . . 321
1.4.1 actionoptionmenu - Tool . . . 322
1.4.2 autocli - Tool . . . 325
1.4.3 autogui - Tool . . . 328
1.4.4 autoguitest - Function . . . 340
1.4.5 blinklabel - Tool . . . 341
1.4.6 checkmenu - Tool . . . 345
1.4.7 combobox - Tool . . . 365
1.4.8 deleteregions - Tool . . . 393
1.4.9 dialogbox - Tool . . . 400
1.4.10 extendoptionmenu - Tool . . . 401
1.4.11 fontchooserwidget - Tool . . . 419
1.4.12 gchooser - Tool . . . 421
1.4.13 gchoosertest - Function . . . 425
1.4.14 gopher - Tool . . . 425
1.4.15 guientry - Tool . . . 429
1.4.16 guientrytest - Function . . . 467
1.4.17 helpmenu - Tool . . . 468
1.4.18 histogramgui - Tool . . . 472
1.4.19 listboxcontainer - Tool . . . 479
1.4.20 multiautogui - Tool . . . 491
1.4.21 multiscale - Tool . . . 497
1.4.22 menuframes - Tool . . . 505
1.4.23 messageline - Tool . . . 514
1.4.24 optionmenu - Tool . . . 518
1.4.25 pixelrange - Tool . . . 544
1.4.26 popupselectmenu - Tool . . . 550
1.4.27 profilewidthentry - Tool . . . 554
1.4.28 radiobuttons - Tool . . . 558
1.4.29 recordbrowserwidget - Tool . . . 565
1.4.30 restoreregions - Tool . . . 567
1.4.31 rollup - Tool . . . 573
1.4.32 saveregions - Tool . . . 583
1.4.33 scrolllistbox - Tool . . . 589
1.4.34 scrolltext - Tool . . . 595
1.4.35 selectablelist - Tool . . . 600

1.4.36 synclistboxes - Tool 608
1.4.37 tabdialog - Tool 628
1.4.38 tabdialogtest - Function 640
1.4.39 tapedeck - Tool 641
1.4.40 taqlwidget - Tool 647
1.4.41 twodactionoptionmenu - Tool 651
1.4.42 widgetserver - Tool 654
1.4.43 is_widgetserver - Function 695

Chapter 1

Display

package for image and data display

Description
The display package contains modules that support the display of AIPS++ Images, *Glish* arrays and other AIPS++ data.

1.1 guiutils - Module

GUI Related Utilities

Description
guiutils contain several graphic user interface (GUI) releated functions that allow a programmer/user to layer a GUI on top of an application. Please look at the "Writing a GUI for Your Application" in Note 197: How to Write an AIPS++Application.

Tools

barchart	Record values and display as a barchart
displaylist	Maintain an editable list of items
evalwindow	A simple GUI text editor from which Glish code may be evaluated.
gui	Standard Input Forms generated from Glish records
guiframework	Standard framework for Glish based AIPS++Graphical User Interfaces (GUI's)
messagebox	A simple popup text window for displaying short to medium-size messages.
progress	Display visually the progress of some task
single_entry_dialog_box	A popup dialog box for entering one item of text.
status_line	A one-line text widget for displaying status information.
tracewindow	A text display widget for tracing Glish function execution

Functions

addpopuphelp	add popup help to a GUI
choice	Have a user select one of a number of choices
choicewindow	Display a question in a window with multiple choice answers
chooser	AIPS++ chooser subsequence
datachooser	Data Chooser
displaylisttest	Test/demonstration of the displaylist class
donechoice	A choice dialog appropriate for done()
exitchoice	A choice dialog appropriate for exit
filechooser	File Chooser
infowindow	Display an informational message in a window
popuphelp	add popup help to a GUI
popupmenu	add a popup type selection help menu to a GUI
popupremove	remove popup help from a GUI
tablechooser	Table Chooser

1.1.1 choice - Function
Package display
Module guiutils

Have a user select one of a number of choices

Synopsis
choice(description, choices, types, default, interactive, timeout, widgetset)
include "choice.g"

Description

Choice is used to have the user select one of several strings. It puts up a gui box with some descriptive text, and a button for each choice. A default choice can be indicated. When left unset, the the first choice is the default. The default is used if it is not possible to put up a GUI or a timeout occurs.
This function is used by the tasking system, but might be of use to people writing their own scripts.
The types, when specified, should be chosen from those known to the widgetserver. These include "plain", "action", "halt", and "dismiss". When types is unset (its default value) all buttons will be "plain".
When a default is specified, it indicates which element of choices is returned when no GUI can be displayed or a timeout occurs.

Arguments

description	Descriptive text for the choices Allowed:string
choices	One of these choices will be returned Allowed:string array
types	Button types for choices Allowed:string array Default: unset
default	Which choice is the default one Allowed:integer Default: 1
interactive	If interactive, use a GUI, otherwise return the first choice. Allowed:boolean
timeout	How long to wait for the user to respond Allowed:integer
widgetset	Which widgetserver to use Allowed:widgetserver Default: dws

Returns
string

Example

```
# all plain buttons, default choice is no
result := choice('Do you want to quit?', ['no', 'yes'])
if (result == 'yes') {print 'Bye!'; exit; };
```

Example

```
# halt and dismiss button types, choice is yes
result := choice('Do you want to quit?', ['no', 'yes'],
                 ['dismiss', 'halt'], 2)
if (result == 'yes') {print 'Bye!'; exit;};
```

1.1.2 donechoice - Function

Package display
Module guiutils

A choice dialog appropriate for done()

Synopsis
donechoice(description, widgetset)
include "choice.g"

Description
This is a choice appropriate for use with a tool done function. Done which may be invoked by pressing a "Done" button on the tool GUI. Since done is used to destroy the tool, it is often desirable to ensure that the user really meant to invoke that function. This function exists so that the choice dialog boxes used with done all have a similar appearance. Only the descriptive text will vary. This function returns a boolean T if the user has pressed the "yes" button. In all other cases, a F will be returned.

Arguments

description	Descriptive text for the choices Allowed:string
widgetset	Which widgetserver to use Allowed:widgetserver Default: dws

Returns
boolean

Example

```
amidone := donechoice();
if (amidone) {print 'This tool has been destroyed';};
```

1.1.3 exitchoice - Function
Package display
Module guiutils

A choice dialog appropriate for exit

Synopsis
exitchoice(widgetset)
include "choice.g"

Description
This is a choice appropriate for use with a GUI "Exit" button. According to the GUI Guidelines, "Exit" is used to allow the user to exit from glish. It is usually desirable to ensure that the user really meant to exit glish. This function exists so that the choice dialog boxes used with "Exit" all have the same appearance.
This function returns a boolean T if the user has pressed the "yes" button. In all other cases, a F will be returned.

Arguments

widgetset	Which widgetserver to use Allowed: widgetserver Default: dws

Returns
boolean

Example

```
reallyexit := exitchoice();
if (reallyexit) {print 'Bye!'; exit;};
```

1.1.4 filechooser - Function
Package display
Module guiutils

File Chooser

Synopsis
filechooser(dirName, restrictions, title, wait, writeOK, multi, access)

Description

Provides AIPS++ with a standard file chooser. Typical use would be in one of three ways (there are lots of permuation).

filechooser() blocks, waits around for the user to make a choice and returns the result in a glish record, guiReturns='filename'. This will likely change so that it returns only the file name.

filechooser(wait=F) non-blocking, returns an agent. The agent signal a "returns" event when the OK button is pressed.

filechooser(multi=T) blocks, allows multiple choices. Returns the choices in a string vector.

You may use the restrictions argument to fine tune how the filechooser presents the directories and files.

restrictions $=[data = F, tables = F]$ file chooser default. Allows you to pick just files.

restrictions $=[data = T, tables = T]$ data chooser default. Allows you to pick tables and files and allows the user to choose what kinds of data to be able to pick.

restrictions $=[data = F, tables = T]$ table chooser default. Allows only tables to be choosen. No refinements of which tables to allowed.

restrictions $=[data = T, tables = F]$ not useful.

You may use the access argument to select only those files with certain access permissions. The access argument can contain any combination of the values "r", "w", and"x".

access ="r" the default value for all choosers. Only files with 'read' permission are displayed.

access ="w". Only files with 'write' permission are displayed.

access ="x". Only files with 'execute' permission are displayed.

access ="rw". Files which are both readable and writable are displayed. Other combinations are possible.

Arguments

dirName	Directory name Allowed:Any directory name Default: Current directory
restrictions	Whether to show tables or files Allowed:Two element boolean record Default: [T,T]
title	Title to put on Window Allowed:Any String Default: AIPS++ chooser
wait	Flag for whether to wait for result or not Allowed:T or F Default: T
writeOK	Flag for allowing writing of files Allowed:T or F Default: T
multi	Flag for allowing multiple choices Allowed:T or F Default: F
access	String specifying required access permissions of files Allowed:any combination of r,w, and x Default: r

Returns
An agent or record containing the file name

1.1.5 datachooser - Function
Package display
Module guiutils

Data Chooser

Synopsis
datachooser(dirName, title, wait, writeOK, multi, access)

Description
The data chooser allows the user to select aips++ tables and files. There are options available for limiting the scope of the data to be selected. You may choose to not block by setting wait=F.

Arguments

dirName	Directory name Allowed:Any directory name Default: Current directory
title	Title to put on Window Allowed:Any String Default: AIPS++ chooser
wait	Flag for whether to wait for result or not Allowed:T or F Default: T
writeOK	Flag for allowing writing of files Allowed:T or F Default: T
multi	Flag for allowing multiple choices Allowed:T or F Default: F
access	String specifying required access permissions of files Allowed:any combination of r,w, and x Default: r

Returns
An agent or record containing the table/file name

1.1.6 tablechooser - Function

Package display
Module guiutils

Table Chooser

Synopsis
tablechooser(dirName, title, wait, writeOK, multi, access)

Description
The table chooser allows the user to select only aips++ tables. You may choose to not block by setting wait=F.

Arguments

dirName	Directory name Allowed:Any directory name Default: Current directory
title	Title to put on Window Allowed:Any String Default: AIPS++ chooser
wait	Flag for whether to wait for result or not Allowed:T or F Default: T
writeOK	Flag for allowing writing of files Allowed:T or F Default: T
multi	Flag for allowing multiple choices Allowed:T or F Default: F
access	String specifying required access permissions of files Allowed:any combination of r,w, and x Default: r

Returns
An agent or record containing the table name

1.1.7 chooser - Function
Package display
Module guiutils

AIPS++ chooser subsequence

Synopsis
chooser(dirName, restrictions, title, wait, writeOK, multi, access)

Description
The chooser subsequence provides the actual gui for the file, data, and table choosers. You may call it directly but the preference is for glish programmers to use the file, data, or table chooser.

Arguments

dirName	Directory name Allowed:Any directory name Default: Current directory
restrictions	Whether to show tables or files Allowed:Two element boolean record Default: [T,T]
title	Title to put on Window Allowed:Any String Default: AIPS++ chooser
wait	Flag for whether to wait for result or not Allowed:T or F Default: T
writeOK	Flag for allowing writing of files Allowed:T or F Default: T
multi	Flag for allowing multiple choices Allowed:T or F Default: F
access	String specifying required access permissions of files Allowed:any combination of r,w, and x Default: r

Returns
An agent, sends a returns event when OK button is pressed

1.1.8 guiframework - Tool

Package display
Module guiutils

Standard framework for Glish based AIPS++Graphical User Interfaces (GUI's) include "guiframework.g"

Description
The guiframework (gf) provides a framework for constructing consistent graphical user interfaces. It handles the overhead of the standard set of widgets that most GUI's should have, menubars, command buttons, status lines. Customization of the menu bar is fairly straight forward with the use of Glish records.
A menu record consists of

```
menu_name::text                #! Text appearing on the Menu Button
menu_name.menu_item
```

```
menu_name.menu_item.text      #! Text appearing on the Menu Item
menu_name.menu_item.relief
menu_name.menu_item.action
menu_name.menu_item.type
menu_name.menu_item.state
```

An action record consists of

```
action_name.text
action_name.action
```

The functions, addactions and addmenus, expect the action and menu records to live off a root menu or action tree (i.e. menu_root.menu_name.menu_item). Any valid glishTk button attribute maybe set for each menu or action item. The following member functions illustrate how to construct the menu/action records.

default_filemenu make a simple menu,

default_options make a menus with cascading menus

defaultmenus put together a menus record

defaulthelp put together a help record

default_actions create an actions record

```
self.default_filemenu := function()
{  wider public;
   file := [=];
   file::text := 'File';
   file.open := [=];
   file.open.text := 'Open...';
   file.open.relief := 'flat';
   file.open.action := function(){
               return filechooser(title='AIPS++ Filechooser -- Open File');}
   file.save := [=];
   file.save.text := 'Save...'
   file.save.relief := 'flat';
   file.save.action := function(){
               return filechooser(title='AIPS++ Filechooser -- Save File');}
   file.blank := [=];
   file.blank.text := '';
   file.blank.relief :='flat';
   file.blank.disable :=T;

   file.bug := [=];
   file.bug.text :='Report Bug...';
```

```
   file.bug.relief :='flat';
   file.bug.action := bug;

   file.blank1 := [=];
   file.blank1.text := '';
   file.blank1.relief :='flat';
   file.blank1.disable :=T;

   file.close := [=];
   file.close.text := 'Close';
   file.close.relief := 'flat';
   file.close.action := public.dismiss;

   return ref file ;
}

self.default_options := function()
{
   options := [=];
   options::text := 'Options'
   options.browser := [=];
   options.browser::text := 'Browser'
   options.browser.text := 'Browser'
   options.browser.type := 'menu'
   options.browser.relief := 'flat'
#
   options.browser.menu := [=];
   options.browser.menu.netscape := [=];
   options.browser.menu.netscape.text := 'Netscape';
   options.browser.menu.netscape.type := 'radio';
   options.browser.menu.netscape.relief := 'flat';
   options.browser.menu.netscape.action := function(){ global helpsystem;
                                           helpsystem::browser := 'netscape';}
#
   options.browser.menu.mosaic := [=];
   options.browser.menu.mosaic.text := 'Mosaic';
   options.browser.menu.mosaic.type := 'radio';
   options.browser.menu.mosaic.relief := 'flat';
   options.browser.menu.mosaic.action := function(){global helpsystem;
                                           helpsystem::browser := 'mosaic';}
#
   options.browser.menu.other := [=];
   options.browser.menu.other.text := 'Other';
   options.browser.menu.other.type := 'radio';
   options.browser.menu.other.relief := 'flat';
   options.browser.menu.other.action := function(){ global helpsystem;
```

```
                                        helpsystem::browser := 'other';}
   return ref options;
}

   # Returns a reference to the default menus, file and options

public.defaultmenus := function()
{  wider self;
   menus := [=];
   menus.file := self.default_filemenu();
   menus.options := self.default_options();
   return ref menus;
}

   # Returns a reference to the default help menu

public.defaulthelp := function()
{
   hmenu := [=];
   hmenu::text := 'Help';
   hmenu.about := [=];
   hmenu.about.text := 'About';
   hmenu.about.relief := 'flat';
   hmenu.about.action := about;
#
   hmenu.refman := [=];
   hmenu.refman.text := 'Reference Manual';
   hmenu.refman.relief := 'flat';
   hmenu.refman.action := function()
                         { help('Refman:'); }
#
   hmenu.gstart := [=];
   hmenu.gstart.text := 'Getting Started';
   hmenu.gstart.relief := 'flat';
   hmenu.gstart.action := function()
                          { help('gettingstarted:'); }

   return ref hmenu;
}

   # Returns a reference to the default actions

public.defaultactions := function()
{  wider public;
   action := [=];
   action.apply := [=];
```

```
   action.apply.text := 'Apply';
   action.apply.action := function(){print 'Apply Pressed'};
   action.reset := [=];
   action.reset.text := 'Reset';
   action.reset.action := function(){print 'Reset Pressed'};
   action.cancel := [=];
   action.cancel.text := 'Cancel';
   action.cancel.action := public.dismiss;
   return ref action;
}
```

Here are a couple of simple examples that illustrate the use of guiframework.
First we construct a simple message window with a dismiss button.

```
# Info dialog

infowindow := function( some_text='Your slogan here.',
                        title='AIPS++ Infomation Window')
{
       #  Create the default action button, Dismiss

    action := [=];
    action.dismiss.text := 'Dismiss';

       #  Create the guiframework, title sets the title of the frame,
       #   the first F, tells the constructor not to use the default menus
       #   the second F, tells the constructor not to use the default help menu
       #   action, says create the action buttons (there's only one)

    a := guiframework(title, F, F, action);

       # We need set the dismiss handler after we create the tool otherwise
       # it wouldn't know how to dismiss itself.

    a.addactionhandler('dismiss', a.dismiss);

       # Now get the frame that will be the parent of all our application
       # gui components

    wf := a.getworkframe();

       # Attach a message to the work frame

    m := message(wf, some_text, width=600);
```

```
       # Return a reference to the info window.
    return ref a;
}
```

Here's a more sophisticated example. It implements a choices window.

```
choicewin := function(question, choices, interactive=have_gui(), timeout=150)
{
    if (!is_string(question) || !is_string(choices)) {
        fail '::choice(description,choices) : description and choices must be strings'
    }

    if (!interactive || !have_gui()) {
        return choices[1]
    }

       #  Setup up the buttons, the default action procedure for each button
       #  is to return its value.

    action := [=];
    for(choice in choices){
       action[choice] := [=];
       action[choice].text := choice;
    }
       # Create the guiframework, no menus, but as many actions as buttons

    a := guiframework("AIPS++ Please make a choice", F, F, action);
    wf := a.getworkframe();

       # Display the query in the workframe

    m := message(wf, question, width=600);

       # Now we setup the timing loop and do a blocking wait for a response

    done := F;
    timer := client("timer", 0.2);
    choiceIs := choices[1];

       # Put the amount of time left to choose in the status loop

    a.updatestatus(spaste('Time left to choose: ', timeout/5, ' s'));

       #  Wait for an answer or until we timeout
    while(!done){
          # Check if a handler has returned
       if(a.handle::returned){
```

```
            # Yup set the choice a break the loop
           choiceIs := a.handle::value;
           break;
        }
           # Nope, wait for the timer
        await timer->ready
        if ($name == "ready") {
  timeout -:= 1;
  done := timeout <= 0;
           timestring := '     ';
           if(!(timeout%5)){
                 # Update the status line with the timeleft, count down seconds
              timestring := spaste('Time left to chose: ', timeout/5, ' s');
     a.updatestatus( timestring);
           }
         }
    }

      # All done, cleanup timer and window and return the choice

    send timer->terminate();
    a.dismiss();
    return choiceIs;
}
```

Constructors

guiframework	Constructs the GUI framework, menus, work area, etc...

Functions

addactionhandler	Sets the handler for a menu or command buttion
addactions	Add buttons to command area.
addhelp	Add help menus to the menubar.
addmenus	Add menus to the menubar.
busy	Turns the icon into a watch or pointer. Disables the guiframework frame.
cleanup	removes agents generated by guiframework
defaultactions	Returns the default actions for command buttons.
defaulthelp	Returns the default help menu.
defaultmenus	Returns the default menus.
dismiss	Dismisses the main window.
getworkframe	Returns a reference to the frame where most GUI components are found.
updatestatus	Updates the status line on the window.
updatetitle	Updates the title line on the frame/window.

guiframework - Constructor

Package **display**
Module **guiutils**
Tool **guiframework**

Constructs the GUI framework, menus, work area, etc...

Synopsis
guiframework(title, menus, helpmenu, actions, guiframenote, parent)

Arguments

title	Window title Allowed:Any string Default: AIPS++Window
menus	menu records Allowed:Glish menus records Default: Default file and options menus
helpmenu	help record Allowed:Glish help menu record Default: Default help menus
actions	Action record Allowed:Glish action record Default: Default action buttons
guiframenote	note function Allowed:any note function Default: note
parent	Parent frame Allowed:Frame to attach guiframework Default: F

Description

The guiframework provides a consistent look and feel GUI for gui applications that use it. With it you can specify standard menus, help and actions.
If the parent frame is `F` guiframe work will generate the top-level frame.
Often we need to change the state of button, grey it, change a button label or the like. Rather than provide lots of guiframework functions to do these things, there is a backdoor using the app record. The app datamember of guiframework contains all the GUI elements defined by a guiframework. The scheme
The record `app.mb.btns` contain all the buttons in the menubar, a button is identified by the menu name and the menu item name. For example the 'file'

menu is identified `app.mb.btns.file`, the open option on file is `app.mb.btns.fileopen`, menu items concatinated the menu name and menu item name to identify themselves.
Action buttons belong to the `app.cmd.b` record. A go action would be `app.cmd.b.go`.

Returns
A standard AIPS++GUI framework window

---

guiframework.busy - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Turns the icon into a watch or pointer. Disables the guiframework frame.

Synopsis
busy(busyFlag)

Arguments

busyFlag	Busy Flag Allowed:T or F, T for busy, F for not busy Default: T

---

guiframework.dismiss - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Dismisses the main window.

Synopsis
dismiss()

Returns

Removes the window from the display

---

guiframework.defaultmenus - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Returns the default menus.

Synopsis
defaultmenus()

Returns
Returns a record of the default menus, file & options

---

guiframework.defaulthelp - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Returns the default help menu.

Synopsis
defaulthelp()

Returns
Returns a record of the default help menu.

---

guiframework.defaultactions - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Returns the default actions for command buttons.

Synopsis
defaultactions()

Returns
Returns a record of the default action buttions, Apply, Reset, and Cancel.

---

guiframework.addmenus - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Add menus to the menubar.

Synopsis
addmenus(menus)

Arguments

menus	Record, containing menus Allowed:Any Glish menu record

Returns
Nothing

---

guiframework.addhelp - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Add help menus to the menubar.

Synopsis
addhelp(helpmenu)

Arguments

helpmenu	Help menu record Allowed:Glish menu record

Returns
Nothing

---

guiframework.addactions - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Add buttons to command area.

Synopsis
addactions(actions)

Arguments

actions	Action buttons Allowed:Glish action record

Returns
Nothing

---

guiframework.getworkframe - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Returns a reference to the frame where most GUI components are found.

Synopsis
getworkframe()

Returns

A reference to frame which you use for the parent of GUI components.

---

guiframework.updatestatus - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Updates the status line on the window.

Synopsis
updatestatus(status)

Arguments

status	Status line Allowed: Any string

Returns
Nothing

---

guiframework.updatetitle - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Updates the title line on the frame/window.

Synopsis
updatetitle(status)

Arguments

status	Status line Allowed: Any string

Returns

Nothing

---

guiframework.addactionhandler - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

Sets the handler for a menu or command buttion

Synopsis
addactionhandler(button, action)

Arguments

button	button name Allowed:String Default: Valid button name
action	action functions Allowed:Glish function Default: Valid function

Returns
Nothing

Description
Sometimes it's not possible to prescribe a button/menu action before construction of the framework. Addactionhandler allows you do this after the tool is created. A typical example is a menu exit or an action dismiss button. The following code excerpt illustrates adding a command action.

```
action := [=];
action.dismiss := [=];
action.dismiss.text = 'Dismiss';

a := guiframework('My window', F, F, action);
a.addactionhandler('dismiss', a.dismiss());
```

This code excerpt illustrates adding a menu action. Note that for menus we concatenate the menu button names to produce the proper button name. So, the dismiss option of the file menu has a name of filedismiss. If you have cascading menus, you just keep concatenating names, i.e. options, browser, netscape becomes optionsbrowsernetscape.

```
menus := [=];
menus.file := [=];
menus.file::text := 'File';
.
.
.
menus.file.dismiss := [=];
menus.file.dismiss.text := 'Dismiss'
menus.options := [=];
.
.
.
a := guiframework('My app', menus, T, T)
a.addactionhandler('filedismiss', a.dismiss);
```

---

guiframework.cleanup - Function

Package	**display**
Module	**guiutils**
Tool	**guiframework**

removes agents generated by guiframework

Synopsis
cleanup()

Returns
Nothing

1.1.9 gui - Tool

Package	**display**
Module	**guiutils**

Standard Input Forms generated from Glish records
include "inputframe.g"

Description
The GUI tool provies a way for generating standard input forms. Input forms are supported in two ways:

1. Stand-alone input form,

2. Input form imbedded in a glishTk frame,

Additionally the input forms may be selected either via tabs or a scrolling list. The inputform and tabform methods use a *screen input record* or a collection of screen input records (passed as a record). The *screen input record* has the following members

title	text in the window title
label	text in the window
actions	(a.label a.function)record of functions
layout	ignored for now
data	record of datums
progress	T or F (Shows a progress bar, not implemented)

Each member of a *data record* is a datum and has the following members

label	Description of datum field
type	one of the following: string, vector_string, float, double long, integer, file, table, time, date, or text - string with more than oneline of text
required	boolean, T if required for processing
hint	check, radio, list, menu
enums	vector of allowed values
multiple	boolean for allowing multiple choices of enums
range.min	minimum value
range.max	maximum value
default	default value
verify	function to verify input
mask	display mask (not implemented)
help.url	a URL to drive a web browser (not implemented)
help.text	text for help of this field (not implemented)

If a user keys in a value starting with a $, the Glish variable with that same name (minus the $) will be used for that input value.

Constructors

Functions

inputform	Creates a input form, either stand-alone or in a frame
tabform	Creates a tab-styled input form, either stand-alone or in a frame

gui.inputform - Function

Package **display**
Module **guiutils**
Tool **gui**

Creates a input form, either stand-alone or in a frame

Synopsis
inputform(scrnInputs, title, dismiss, parent)

Arguments

scrnInputs	Screen input record Allowed:any valid screen input record
title	Label or window title of input form Allowed:Any String Default: AIPS++ Input Form
dismiss	Dismiss flag Allowed:Flag to dismiss the form after returning from one of the action events Default: T
parent	The parent frame Allowed:Either F or the frame to put the input form Default: F

---

gui.tabform - Function

Package **display**
Module **guiutils**
Tool **gui**

Creates a tab-styled input form, either stand-alone or in a frame

Synopsis
tabform(methods, title, dismiss, parent, tabcount, side)

Arguments

methods	Collection of screen input records used by inputform Allowed:any valid record of screen input records
title	Label or window title of input form Allowed:Any String Default: AIPS++ Tab Input Form
dismiss	Dismiss flag Allowed:Flag to dismiss the form after returning from one of the action events Default: T
parent	The parent frame Allowed:Either F or the frame to put the tab-input form Default: F
tabcount	Number of functions that cause a list box to be used Allowed:any integer but should be less than 7 Default: 5
side	Which side to pack the list box on Allowed:top, bottom, left, right Default: left

Description
The function's argument may have a two attributes, disallow and show. The disallow attributes tells the tabframe not to allow the array of functions to display or execute that function. The show option will gray out a function in the tab list rather than hide it.

Several utility functions are available via the guiutils module. include "guicomponents.g"

Description

1.1.10 status_line - Tool

Package display
Module guiutils

A one-line text widget for displaying status information.
include "guicomponents.g"

Description
Many GUI programs will be easier to use if status information is displayed at

appropriate times for the user to read. This very simple widget provides a convenient and standard way to do this.

Constructors

status_line	Construct a status line widget.

Functions

append	Add text to the end of the current text.
clear	Remove all text from the widget.
delete	Destroy the status_line widget
show	Clear the line, and display this message

status_line - Constructor

Package	**display**
Module	**guiutils**
Tool	**status_line**

Construct a status line widget.

Synopsis
status_line(parentFrame)

Arguments

parentFrame	a glishtk frame widget

Returns
a status_line tool

---

status_line.show - Function

Package	**display**
Module	**guiutils**
Tool	**status_line**

Clear the line, and display this message

Synopsis
show(message)

Arguments

message	a new status message Allowed: any string

Returns
T

Description

---

status_line.clear - Function

Package	**display**
Module	**guiutils**
Tool	**status_line**

Remove all text from the widget.

Synopsis
clear()

Returns
T

---

status_line.append - Function

Package	**display**
Module	**guiutils**
Tool	**status_line**

Add text to the end of the current text.

Synopsis
append(message)

Arguments

message Allowed:any string

Returns
T

Description
A blank space is prepended to [message], and the result is written to the end of the current text displayed in the widget.

---

status_line.delete - Function

Package **display**
Module **guiutils**
Tool **status_line**

Destroy the status_line widget

Synopsis
delete()

Returns
T

1.1.11 messagebox - Tool

Package **display**
Module **guiutils**

A simple popup text window for displaying short to medium-size messages. include "guicomponents.g"

Description
This widget creates its own toplevel frame, displays the supplied message, and

deletes itself when the delete member function is called, or the user presses the "Dismiss" button.

Constructors

messagebox	Create a message box.

Functions

delete	Destroy the messagebox window.
text	Erase any existing text, and display the new message.

messagebox - Constructor

Package	**display**
Module	**guiutils**
Tool	**messagebox**

Create a message box.

Synopsis
messagebox(message, background)

Arguments

message	The text to display Allowed:any string
background	background color Allowed:any color available on your system. Default: white

Returns
A messagebox tool

Description
Newlines can be embedded in the message. The window has quite a wide default width, about two-thirds the width of the average display screen. If the longest line of your message is shorter than that, the messagebox widget shrinks in width to be just the size of the longest line. No scrollbars are provided to accomodate wider or longer messages.

---

messagebox.text - Function

Package **display**
Module **guiutils**
Tool **messagebox**

Erase any existing text, and display the new message.

Synopsis
text(message)

Arguments

message	New text to display Allowed:any String, newlines included

Returns
T

---

messagebox.delete - Function

Package **display**
Module **guiutils**
Tool **messagebox**

Destroy the messagebox window.

Synopsis
delete()

Returns
T

1.1.12 single_entry_dialog_box - Tool

Package **display**
Module **guiutils**

A popup dialog box for entering one item of text.
include "guicomponents.g"

Description
This popup widget provides a way to get input from the user via a transient window. The programmer specifies a prompt, and an optional one-argument callback function – which will be called with the entered text as its sole argument. When the user types [return] in the entry widget, or presses the 'Okay' button, the popup window is destroyed, and the callback function is called. If the user presses the 'Dismiss' button, then the popup window is destroyed, and the callback function is not called.

Constructors

single_entry_dialog_box	Create the popup.

Functions

delete	Destroy the single_entry_dialog_box.

single_entry_dialog_box - Constructor

Package	**display**
Module	**guiutils**
Tool	**single_entry_dialog_box**

Create the popup.

Synopsis
single_entry_dialog_box(prompt, callbackFunction)

Arguments

prompt	Text to display, as a label, to the left of the text entry widget Allowed: any string Default: "
callbackFunction	Called at normal termination Allowed: Any one-argument Glish function. Default: F

Returns
A single_entry_dialog_box

Description
There are occasions in any GUI program where you need the user to enter a small amount of text. Sometimes the best way to get that text is with a popup window containing a text entry widget, and some appropriate termination buttons. A good way to make this operation modeless (i.e., run asynchronously) is to provide a function which will be called when the dialog box terminates; this widget returns the user's text as the first argument to the callback function.

Example

```
callback := function (stringValue) {
   print 'the user is: ', stringValue;
   }
nameDialog := single_entry_dialog_box ('Your name', callback);
```

---

single_entry_dialog_box.delete - Function

Package **display**
Module **guiutils**
Tool **single_entry_dialog_box**

Destroy the single_entry_dialog_box.

Synopsis
delete()

Returns
T

1.1.13 evalwindow - Tool

Package **display**
Module **guiutils**

A simple GUI text editor from which Glish code may be evaluated.

include "guicomponents.g"

Description
This top level window provides a text editor specialized for Glish. It is specialized in the sense that any text you type can be sent to the Glish interpreter for evaluation. This makes it a good environment for developing (at least) small amounts of Glish code. As of now (December 1996) there are many standard editor features which are not yet available: saving text to a file, reading from a file, inserting text, pasting text, searching text. But all of this is possible in principle because the underlying tk 'Text' widget is quite powerful. Once these capabilities are developed, and sensibly presented in this composite widget, the 'evalwindow', it may be a good environment for the development of even large amounts of Glish code.

See Also
AIPS++Note 194: The SDCalc User Environment

Constructors

evalwindow	Create a Glish-specific editor

Functions

clear	Deletes all the text in the window.
contents	Get all of the text in the window
delete	Destroy the evalwindow

evalwindow - Constructor

Package	**display**
Module	**guiutils**
Tool	**evalwindow**

Create a Glish-specific editor

Synopsis
evalwindow(title, columns, rows)

Arguments

title	A label for the window Allowed:any string Default: ”
columns	Width in characters Allowed:any integer Default: 60
rows	Height in lines Allowed:any integer Default: 20

Returns
An evalwindow tool

---

evalwindow.contents - Function

Package	**display**
Module	**guiutils**
Tool	**evalwindow**

Get all of the text in the window

Synopsis
contents()

Returns
A string, including newlines

---

evalwindow.clear - Function

Package	**display**
Module	**guiutils**
Tool	**evalwindow**

Deletes all the text in the window.

Synopsis
clear()

Returns

T

---

evalwindow.delete - Function

Package	**display**
Module	**guiutils**
Tool	**evalwindow**

Destroy the evalwindow

Synopsis
delete()

Returns
T

1.1.14 tracewindow - Tool

Package	**display**
Module	**guiutils**

A text display widget for tracing Glish function execution
include "guicomponents.g"

Description
The tracewindow is a work in progress. As of this writing (December 1996) the best use of the tracewindow is for the simple display of messages from a running Glish application. (See sdavg.g for the first example of this.) As any application performs its work, some of its operations will be performed behind the scenes, hidden from the user's view; this is especially true of a GUI application. Some users will want to know what is going on behind the scenes, and the well-written program will display those operations as actual Glish function calls, by sending appropriate text to the tracewindow.
The next level of tracing is only hinted at right now (December 1996). The goal – partly accomplished with some global functions located at the bottom of guicomponents.g – is to provide a tool for the user to select any function that appears in the tracewindow, and to view the Glish code that makes up that function. This approach, if properly supported, would allow an interested user to navigate at will through Glish code, starting with the function calls the

application prints to the tracewindow, and going just as deep, or as far afield, as she wishes.
This window is primarily useful when used as part of a properly constructed Glish GUI appplication. Such an application will, at the user's request, print the text of crucial function calls and variables to the tracewindow, and will ensure that important lower level functions have text representations for display here in the tracewindow as well. (Please see view_function_test () and view_function_test::text in guicomponents.g.)

See Also
AIPS++Note 194: The SDCalc User Environment

Constructors

tracewindow	Create a window for navigating Glish code.

Functions

append	Display newString on a new line, at the bottom of existing text
clear	Delete all of the text in the window
delete	Destroy the tracewindow
view_named_function	Display the text of the function

tracewindow - Constructor

Package	**display**
Module	**guiutils**
Tool	**tracewindow**

Create a window for navigating Glish code.

Synopsis
tracewindow(title, columns, rows)

Arguments

title	A label for the window Allowed:any string Default: ”
columns	Width in characters Allowed:any integer Default: 50
rows	Height in lines Allowed:any integer Default: 12

Returns
a tracewindow tool

---

tracewindow.append - Function

Package	**display**
Module	**guiutils**
Tool	**tracewindow**

Display newString on a new line, at the bottom of existing text

Synopsis
append(newString)

Arguments

newString Allowed:any string

Returns
T

---

tracewindow.clear - Function

Package	**display**
Module	**guiutils**
Tool	**tracewindow**

Delete all of the text in the window

Synopsis
clear()

Returns
T

---

tracewindow.delete - Function

Package	**display**
Module	**guiutils**
Tool	**tracewindow**

Destroy the tracewindow

Synopsis
delete()

Returns
T

---

tracewindow.view_named_function - Function

Package	**display**
Module	**guiutils**
Tool	**tracewindow**

Display the text of the function

Synopsis
view_named_function(functionName)

Arguments

functionName

Returns
T

1.1.15 progress - Tool

Package display
Module guiutils

Display visually the progress of some task
include "progress.g"

Description

A progress bar is used to track the progress of some task. It has a "meter" with a sliding line that advances across the window as the task executes. The progress bar will normally disappear when the task has reached its maximum value. The progress bar can optionally estimate the time remaining in the task by assuming that the remaining portion will complete at the same rate as the portion completed to date.
If X windows is not available (`have_gui() == F`) then this tool just does no-op's.

Example

```
- include "progress.g"
- bar := progress(-100, 100, 'main title', 'subtitle')
- bar.update(-75)
- bar.update(-50)
- ...
- bar.update(100)      # GUI window is removed
```

progress - Constructor

Package display
Module guiutils
Tool progress

Create a progress meter

Synopsis

progress(min, max, title, subtitle, minlabel, maxlabel, estimate_time, barwidth, barheight, startopen)

Description

Create a progress meter. If `have_gui()` is false (for example, no DISPLAY environment variable is set) this results in a no-op tool. You generally do not need to set `minlabel` and `maxlabel`, since the defaults are generally what you want.

Note that `min` may in fact be greater than `max`. If so, the progress meter will "count down" correctly.

The `startopen` parameter is a little unusual. If it is a boolean (tt T or `F`) then, if true, the progress meter starts with the GUI "on". If a number, the progress meter's GUI turns on after the supplied number of seconds has elapsed. The reason this has been implemented is that it is distracting to see very short-lived progress meters popping up and disappearing very quickly.

Arguments

min	minimum value Allowed:Numeric
max	maximum value Allowed:Numeric
title	window title Allowed:String Default: ""
subtitle	bar title Allowed:String Default: ""
minlabel	left label Allowed:String Default: "" (minimum value)
maxlabel	right label Allowed:String Default: "" (maximum value)
estimate_time	estimate time remaining? Allowed:Boolean Default: T
barwidth	width of bar in pixels Allowed:Integer Default: 200
barheight	height of bar in pixels Allowed:Integer Default: 20
startopen	start with the window displayed? Or, how long to wait before displaying? Allowed:Boolean or Integer Default: 5

Returns
progress tool

---

progress.update - Function

Package	**display**
Module	**guiutils**
Tool	**progress**

Update a progress meter

Synopsis
update(value, autodisable)

Description
Updates the position of the bar on the progress meter. If `autodisable` is true, it will remove the window if it reaches the maximum value.

Arguments

value	New value of the progress meter Allowed:Numeric
autodisable	Remove the window when "max" is reached? Allowed:Boolean Default: T

Returns
Real - the fraction of the bar that is complete, 0.0-1.0.

---

progress.activate - Function

Package	**display**
Module	**guiutils**
Tool	**progress**

Remove the window from the screen

Synopsis

activate()

Description
Remove the progress meter window from the screen. If the window is already inactive, the function is a no-op.

---

progress.deactivate - Function

Package	**display**
Module	**guiutils**
Tool	**progress**

Return the window to the screen

Synopsis
deactivate()

Description
If the window is not currently active on the screen, this function returns it, even if the window has previously hit its maximum. If the window is already active, this function is a no-op.

1.1.16 displaylist - Tool

Package	**display**
Module	**guiutils**

Maintain an editable list of items
include "displaylist.g"

Description

A display list is an editable list of items. An item may either be in the "draw" list, or it may be in the "clipboard". An item moves back and forth via `cut`, `copy`, and `paste` operations.
This class is not intended to be used by general users, rather it is intended to be used in the constructions of GUI's.

Example
See global function, `displaylisttest()`

displaylist - Constructor

Package	**display**
Module	**guiutils**
Tool	**displaylist**

Create a displaylist tool

Synopsis
displaylist(parentframe, formatfunction)

Description
Create a list tool ready to start adding values to. Unless a frame is given to the constructor, the default is to not have a GUI representation of the displaylist. Generally you should provide a function to turn inserted values into strings to display in the list. By default, `as_string()` will be used, which is rarely what you will want.

Arguments

parentframe	frame to use for gui Allowed:frame or F Default: F (no frame)
formatfunction	turn list values into string for display Allowed:function

Returns
displaylist tool

---

displaylist.add - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

add a value to the display list

Synopsis
add(value)

Description
This function adds a new value to the draw list. Note that you may add any type of value to the display list, although generally you will add record types

Arguments

value	item to add to the draw list Allowed:any

Returns
number of entries in the draw list

---

displaylist.get - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

get an item from the displaylist

Synopsis
get(num, fromdrawlist)

Description
Use an index (1 relative) to get an item from either the draw list or the clipboard (by default the draw list). You can determine how many items there are by calling `ndrawlist()` or `nclipboard()`.

Arguments

num	index Allowed:integer
fromdrawlist	get from drawlist? Allowed:T

Returns
the item or fail if index invalid

---

displaylist.set - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

change or add an item in displaylist

Synopsis
set(num, value)

Description
Change or an existing item in the draw list (only) or add a new one. An existing item is changed if the index is in the range 1–`ndrawlist()`, otherwise a new item is added. (The index should only be one larger than the current length of the drawlist).

Arguments

num	index Allowed:integer
value	item to change or add in the draw list Allowed:any

Returns
the new length of the draw list or fail if index invalid

---

displaylist.clear - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

clear the draw list and clipboard

Synopsis
clear()

Description
Empties all items in the draw list and clipboard.

---

displaylist.cleardrawlist - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

clear the draw list

Synopsis
cleardrawlist()

Description
Empties all items in the draw list.

---

displaylist.clearclipboard - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

clear the clipboard

Synopsis
clearclipboard()

Description
Empties all items in the clipboard.

---

displaylist.ndrawlist - Function

Package	**display**

Module	**guiutils**
Tool	**displaylist**

how many items are in the draw list

Synopsis
ndrawlist()

Description
Returns the number of items currently in the draw list.

Returns
integer

---

displaylist.nclipboard - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

how many items are in the clipboard

Synopsis
nclipboard()

Description
Returns the number of items currently in the clipboard.

Returns
integer

---

displaylist.cut - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

move items from the draw list to the clipboard

Synopsis
cut(drawnums)

Description
Cut is used to move items (desiganted as an array of integers) from the draw list to the clipboard.

Arguments

drawnums	index array Allowed:integer

Returns
the length of the drawlist or fail

---

displaylist.copy - Function

Package **display**
Module **guiutils**
Tool **displaylist**

copy items from the draw list to the clipboard

Synopsis
copy(drawnums)

Description
Copy is used to to copy items (desiganted as an array of integers) from the draw list to the clipboard.

Arguments

drawnums	index array Allowed:integer

Returns
the length of the drawlist or fail

---

displaylist.paste - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

copy items from clipboard to the drawlist

Synopsis
paste()

Description
Paste is used to copy items from the clipboard to the drawlist. At present all items from the clipboard are copied to the end of the drawlist. We should probably at least allow pasting into the middle of the draw list.

Returns
the length of the drawlist or fail

---

displaylist.compact - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

restore wasted space in tool

Synopsis
compact()

Description
After many editing operations there can be wasted space inside the tool. Compact reclaims this space. No visibile change occurs in the state of the tool.

---

displaylist.apply - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

call a function on items

Synopsis
apply(fun, dodraw)

Description
Apply applies a function to every item in the draw list (default) or clipboard. The function must take a single argument (or all the other arguments must default), *i.e.*, the item it is being passed.

Arguments

fun	the function we are applying to items Allowed:function
dodraw	apply to items in draw list or clipboard Allowed:boolean Default: T (draw)

Returns
the length of the drawlist or or clipboard or fail

---

displaylist.lastchange - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

return a number that specifies the state of the displaylist

Synopsis
lastchange()

Description

It is sometimes useful to know whether or not the displaylist has changed since the last time you have looked at it (given that the displaylist will often be under user control through some GUI). If the number returned by this function has changed since the last time you called it, it has. The number returned is always non-negative, so you can initialize your cache with a negative number.

Returns
integer

---

displaylist.gui - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

show the displaylist in a GUI

Synopsis
gui(parentframe)

Description
Create a GUI for the displaylist using a supplied frame. The clipboard appears in a listbox on the left, the draw list in a listbox on the right, and various buttons between them to control cutting, pasting, *etc.* Items are selected by clicking and dragging in the listboxes.

Arguments

parentframe	parent frame to write into Allowed:frame

Returns
T or fail

---

displaylist.nogui - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

turn off GUI view of displaylist

Synopsis
nogui()

Description
Turns off the GUI representation of this display list, but does not clear the actual items in the draw list and clipboard. Note that this function sets the parentframe of `gui()` to F.

---

displaylist.save - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

save the displaylist into a file

Synopsis
save(filename)

Description
This function saves the displaylist (including the clipboard), into a file. At present, the file format is just a Glish value, but this is subject to change. If the file already exists, it is overwritten. A fail is returned if the file cannot be written to (probably a read-only or non-existent directory).

Arguments

filename	file to write into Allowed:string

Returns
T or fail

---

displaylist.restore - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

restore the displaylist from a file

Synopsis
restore(filename)

Description
This function restores the displaylist from a file. The current contents are removed first. At present there is no way to merge from a file, but this could be added if desired.

Arguments

filename	file to read from Allowed:string

Returns
T or fail on read failure

---

displaylist.setselectcallback - Function

Package **display**
Module **guiutils**
Tool **displaylist**

call a function when an item is selected

Synopsis
setselectcallback(callback)

Description
This sets a callback function to be called whenever an item is selected in the draw list of the GUI. The function must take one argument - the selected item.

Arguments

callback	function to call on item Allowed:function

Returns
T or fail

---

displaylist.setchangecallback - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

call a function when the list changes

Synopsis
setchangecallback(callback)

Description
This sets a callback function to be called when the list is first changed, at which point the callback is unset. The idea is that the user somehow "saves" the drawlist, at which point he wants to be notified when the list has changed. When he has saved it again, he resets the callback.

Arguments

callback	function to call on item Allowed:function

Returns
T or fail

---

displaylist.setrefreshcallback - Function

Package	**display**
Module	**guiutils**
Tool	**displaylist**

call function on cut and paste

Synopsis
setrefreshcallback(callback)

Description
Set a function to be called whenever there is a cut or paste. Arguably, the function should be called at other times (e.g., copy), or at least there should be additional callback functions. This function must take no arguments.

Arguments

callback	function to call on item Allowed:function

Returns
T or fail

---

displaylist.done - Function

Package **display**
Module **guiutils**
Tool **displaylist**

release/destory this tool

Synopsis
done()

Description
Destroy all resources (GUI and the item lists) allocated by this tool.

1.1.17 displaylisttest - Function

Package **display**
Module **guiutils**

Test/demonstration of the displaylist class

Synopsis
displaylisttest()

Description
Combination test/demonstration of displaylist.

1.1.18 popuphelp - Function

Package display
Module guiutils

add popup help to a GUI

Synopsis
popuphelp(fr, txt, hlp, combi, width)
include "popuphelp.g"

Description
`popuphelp` Is a function which adds the code necessary to add "popup" help to an agent. In particular, when your cursor "lingers" or "hovers" over that agent (e.g. a button) for more than about half a second, the help will popup, and go away when the cursor leaves the agent or when a button is pushed, etc. Help text comes in two flavours: short and long. If only one of them is available, they will both be identical. Short help is displayed on one line if it is not longer than the specified width (60 by default), else on several lines. If no explicit long help is present, the shorthelp is truncated at a new line or at a period, and will never be longer than the specified width. If the longhelp differs from the shorthelp (either by truncation or otherwise), the shorthelp is extended with ".." at the end of the displayed short text.
Short help is displayed by default when the cursors hovers above the agent. Long help is tied to the right mouse button by default.
Popup help can, dynamically, be tied by setting the 'system.help.type' to 0 (no popup), 1(short and long), 2(long only with MB3), 3(long only in hover).
A "popupmenu" function is also available to add a "?" button to your GUI which let the user choose the context popup type action. If a popupmenu is generated, the default can be set with the aipsrc variable "help.popup.type" (possible values: 'none', 'both', 'mb3long' and 'hoverlong'), or by the programmer.
For most popup help items you merely specify a text string that is the help text. For a listbox, you can supply a callback function that takes two arguments and returns a string. The two supplied arguments are:

1. the text of the listbox item the cursor is near; and

2. the Y-index (0 relative!) of the cursor position in the listbox.

It should be noted that timing errors could crash the system (or leave popuphelp floating around) if an agent that supports popuphelp is just zeroed (e.g. by frameagent :=F). A function 'popupremove(agent)' exists to do a proper zeroing.

Arguments

fr	agent to which popup text should be added Allowed:record or agent
txt	popup text, or, for listbox, possible callback routine Allowed:string or function as described above Default: F
hlp	optional separate short help text, or,for listbox, possible callback routine Allowed:string or function as described above Default: F
combi	specifies if short text should precede long text Allowed:bool Default: F
width	specifies maximum length of one shorthelp line Allowed:integer Default: 60

Returns
T (or fail)

Example

```
include 'popuphelp.g';
private := [=];
private.f := frame(height=100);
private.b := button(private.f);
popupmenu(private.f);  # to add a ? menu button (if wanted)
popuphelp(private.b, 'Press the button, and make this a long text');
private.l := listbox(private.f);
private.l->insert('AAA'); private.l->insert('BBB');
private.l.helpfct := function(name, index) {
    # Only need name here.
    return spaste('Your cursor is near ', name);
}
popuphelp(private.l, private.l.helpfct);
```

1.1.19 popupremove - Function

Package display
Module guiutils

remove popup help from a GUI

Synopsis
popupremove(ag, mxlevels)
include "popuphelp.g"

Description
`popupremove` Is a function which deletes any popuphelp (if present) from the specified agent, and sets the agent to False after having done it. Using it is necessary if, e.g., you want to zero the agent in a whenever: timing problems could crash the system if popuphelp is present and lingering somewhere.
The function works recursively, if fields in the argument are records.

Arguments

ag	agent which should be removed (possibly recursively) Allowed:agent Default: F
mxlevels	max. recursion level Allowed:Int Default: 8

Returns
T

Example

```
include 'popuphelp.g';
private := [=];
private.f := frame(height=100);
private.f.a := frame(private.f);
private.f.a.b := button(private.f.a);
popupmenu(private.f);  # to add a ? menu button (if wanted)
popuphelp(private.f.a.b, 'Press the button, and make this a long text');
private.l := listbox(private.f);
```

```
private.l->insert('AAA'); private.l->insert('BBB');
private.l.helpfct := function(name, index) {
    # Only need name here.
    return spaste('Your cursor is near ', name);
}
popuphelp(private.l, private.l.helpfct);
whenever private.f.a.b->press do {
    popupremove(private.f.a);
}
```

1.1.20 popupmenu - Function

Package display
Module guiutils

add a popup type selection help menu to a GUI

Synopsis
popupmenu(agent, deflt, relief)
include "popuphelp.g"

Description
`popupmenu` Is a function which adds to an agent a "?" button with popup selection options of "none", "both", "mb3" or "hover". By specifying a deflt argument of 0, 1, 2 or 3 you can select one of them as the default (default both assumed).

Arguments

agent	agent to which menu button should be added Allowed:agent
deflt	default popup useage type Allowed:integer Default: F
relief	button relief Allowed:relief type string Default: flat

Returns
T (or fail)

Example
See “popuphelp”.

1.1.21 addpopuphelp - Function

Package display
Module guiutils

add popup help to a GUI

Synopsis
addpopuphelp(agents, maxlevels)
include ”popuphelp.g”

Description
`addpopuphelp` Is a function which recurses through a data structure (typically this will be the “private” variable of your closure tool, and for every agent that it finds that has a field named “shorthelp”, it will add the code necessary to add “popup” help to that agent.
It is advised to use the “popuphelp” function directly in stead.
Sometimes record/agent structures are set up where a subrecord is in fact a reference to a parent record. Since this might cause infinite recurses, a `maxlevels` parameter is used to prevent such infinite loops. It is set to 4 by default, which should generally suffice, but very deep structures may require this number to be raised.

Arguments

agents	structure that holds the agents Allowed:record or agent
maxlevels	maximum number of levels to recurse Allowed:4

Returns
T (or fail)

Example

```
include 'popuphelp.g';
private := [=];
private.f := frame(height=100);
private.b := button(private.f);
private.b.shorthelp := 'Press the button';
private.l := listbox(private.f);
private.l->insert('AAA'); private.l->insert('BBB');
private.l.shorthelp := function(name, index) {
    # Only need name here.
    return spaste('Your cursor is near ', name);
}
addpopuphelp(private);
```

1.1.22 barchart - Tool

Package display
Module guiutils

Record values and display as a barchart
include "barchart.g"

Description

`barchart` tools are used to record name and value pairs, and to optionally display them in barchart form in a GUI.
If enough colors are available, the GUI display will use green bars if the size is below a specifiable "large" value, and red if the value is above the large value. If there are not enough colors, the bars are always drawn as outlines in black and white.

Example

```
- bc := barchart('My Title', large=30);
- bc.chart('small', 10)
- bc.chart('large', 50)   # Red, because it is >30
- bc.chart('small', 20)   # Update with new value
- bc.large(15)            # Turn small to Red by changing threshold
```

See Also

1. Glish/PGPLOT
2. pgplotter
3. pgplotwidget

barchart - Constructor

Package	**display**
Module	**guiutils**
Tool	**barchart**

Create a barchart tool

Synopsis
barchart(title, width, height, large, show)

Description
Note that after construction of the barchart, even if `show` is `T`, the barchart will not be visible on the screen until at least one value is charted.

Arguments

title	title for the barchart frame Allowed:string Default: AIPS++ Bar Chart
width	width of the barchart in pixels Allowed:integer Default: 400
height	height of the barchart in pixels Allowed:integer Default: 200
large	charted values larger than this are displayed in red if enough colors are available Allowed:real Default: 1.0e+38
show	show charted values? Allowed:boolean Default: T

Returns
barchart tool or fail

barchart.chart - Function

Package	**display**
Module	**guiutils**
Tool	**barchart**

Chart a new name,value pair

Synopsis
chart(name, value)

Description
This function adds a new name,value pair to the barchart. It does not matter if the name already exists or not, and value can increase or decrease.
If `name` and `value` are arrays, you can update many values at once (of course the arrays must be the same length).

Arguments

name	name of the barchart item Allowed:string
value	new value for "name" Allowed:real

Returns
T on success or F on error

---

barchart.remove - Function

Package	**display**
Module	**guiutils**
Tool	**barchart**

Remove an item from the barchart

Synopsis
remove(name)

Description

This function removes a named item from the barchart.

Arguments

name	name of the barchart item Allowed:string

Returns
T on success or F on error

---

barchart.values - Function

Package	**display**
Module	**guiutils**
Tool	**barchart**

Return all values from the barchart as a record

Synopsis
values()

Description
It is occasionally useful to retrieve all the values from a barchart. This function allows you to do that.

Returns
record of name=value pairs

---

barchart.nogui - Function

Package	**display**
Module	**guiutils**
Tool	**barchart**

Do not display the barchart any more (until gui() is called)

Synopsis

nogui()

Description
This function removes the barchart GUI from the screen if it is being displayed, and prevents it from being displayed until `gui()` is called.

---

barchart.gui - Function

Package	**display**
Module	**guiutils**
Tool	**barchart**

Display the barchart if possible

Synopsis
gui()

Description
This function causes the barchart to be displayed in a GUI if possible (that is, if the glish builtin `have_gui()` is `T`. If enough colors are available (4) a color display is used, otherwise white on black.

---

barchart.large - Function

Package	**display**
Module	**guiutils**
Tool	**barchart**

Display and optionally modify the "large" value

Synopsis
large(newval)

Description

If called with no arguments, this function merely returns the current value of the "large" value (the value at which bars will be drawn in red if enough colors are available).

If called with a value, the large value will be changed as well as returned. If the GUI is being displayed the color of displayed bars will be changed to red if necessary.

Arguments

newval	new large value Allowed:real Default: unset

Returns
real (large value)

---

barchart.done - Function

Package display
Module guiutils
Tool barchart

Delete the barchart tool from memory

Synopsis
done()

Description
This function removes the barchart GUI from memory. After calling `done`, the tool is destroyed and cannot be accessed.

1.1.23 choicewindow - Function

Package display
Module guiutils

Display a question in a window with multiple choice answers

Synopsis
choicewindow(question, choices, interactive, timeout, ws)

include "choicewindow.g"

Description
The choicewindow is a simple dialog window that lets one pose a question to a user with a limited number of responses. The window will dismiss itself after the timeout period specified returning the first choice in the choices lists. If the user make a selection before the timeout it will return the user's choice.

Arguments

question	The question you want answered Allowed:string
choices	Vector of choices Allowed:string
interactive	Is the session iteractive Allowed:Boolean Default: has_gui()
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 'choicewindow.g';
cw := choicewindow('Do you want to continue?', "Yes No");
```

1.1.24 infowindow - Function
Package display
Module guiutils

Display an informational message in a window

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

1.3 viewer - Module

Viewing Images and *Glish* arrays
include "viewer.g"

Description

The `viewer` module provides display services. It can produce false color image, contour and vector map representations of AIPS++ images and *Glish* arrays, rendered either on-screen or to PostScript files. Extensive control over the display of data is available from both graphical user and programming (command-line) interfaces.
The most important `tool` in this module is the Viewer `tool`, which combines a number of the other `tools` in the module to produce a cohesive application, operable from the command-line or from multiple windows in a graphical user interface. Most display services can be accessed directly from the Viewer `tool`.
An example on how to operate the Viewer `tool` from the command-line can be found here.
In general, there should be little need to read documentation on the other `tools` in this module. However, for reference purposes and if you want to integrate the Viewer `tool` into your own application/script take a closer look at the description of the `tools` which are part of this module.
Most of the `tools` in this module can be driven by graphical user interfaces. This documentation starts with a general description of how to use these GUIs to interact with the Viewer `tool`. You really only need to know about the GUIs which the Viewer `tool` provides access to:

1. The GUI which controls the import of data, the Viewerdatamanager `tool`
2. The GUI which controls your display

A list of aipsrc setting to customize the Viewer can be found in the Resources section.
Ok - here a little sitemap for this module:

- **Toolkit approach** go here
- **GUI functionality** look here for the Viewerdisplaypanel `tool` and the Viewerdatamanager `tool`.
- **Customization** look here
- **Viewer - behind the scenes** look here
- **CLI example** look here.

Fundamentals

Displaydatas

The main task of the Viewer `tool` is to produce visual representations of AIPS++ data. Presently, the data itself may be an AIPS++ Image stored on disk, or a *Glish* array, while the visual representation can be a false color image (hereafter a "raster"), or a contour map (hereafter a "contour"). One of the two fundamental `tools` that the Viewer `tool` is built upon is the Viewerdisplaydata `tool`. This `tool` combines a data source and a representation method into a single unit which can be passed around the various `tools` which are managed by a single Viewer `tool`.
A Viewerdisplaydata `tool` is completely self-contained, and has "state". Its state consists of various attributes, some of which will not change during the lifetime of the Viewerdisplaydata `tool`, and many others which probably will. Examples of unchangeable state include the dimensionality of the Image or *Glish* array, and the type of visual representation requested: a raster or a contour. Examples of changeable state include the many attributes which describe how the data should be translated into the requested represetation, such as the "colormap" (see below) to use for a raster, or the contour levels to use for a contour Viewerdisplaydata `tool`. The changeable state of a Viewerdisplaydata `tool` can be modified from the command-line or in a graphical user interface. **Many Viewerdisplaydata `tools` can be created and managed by a single Viewer `tool`.**

Displaypanels

The Viewerdisplaypanel `tool` provides one ore more "canvas(es)" on which one or more Viewerdisplaydata `tools` can draw themselves, and is the second fundamental `tool` upon which the Viewer `tool` is built. This `tool` provides facilities for Viewerdisplaydata `tools` to register and unregister themselves, that is, to join and leave the list of Viewerdisplaydata `tools` that will get asked to draw themselves whenever the canvas is redrawn. Furthermore, it provides a "controlbox" of operations which can be applied to the canvas, for example, zooming (magnifying) the display. Finally, each Viewerdisplaypanel `tool` has two "managers" associated with it: a Viewercanvasmanager `tool` which provides the facility to modify some of the canvas-specific settings (for example, background color, margin widths, and the number of panels), and a Viewercanvasprintmanager `tool` which can be used to generate PostScript and X11 Pixmap output from the contents of the Viewerdisplaypanel `tool`. **Many Viewerdisplaypanel `tools` can be created and managed by a single Viewer `tool`.**

Options One of the most important concepts of the Viewer `tool` is the way

states and attributes are set. `tools` like Viewerdisplaydata `tool` and Viewerdisplaypanel `tool` have functions *getoptions* and *setoptions* to control these "options". For example, the state of a Viewerdisplaydata `tool` is held in

an option record. The documentation of each `tool` summarizes it's options which can be controlled by the user.
Let's say you have made a Viewerdisplaydata `tool` as a raster and you want to know the options on this displaydata, you would to something like this:

```
dd := dv.loaddata('myfantasticimage','raster');
opts := dd.getoptions();
print field_names(opts);
```

These options are usually context dependent, i.e. an option record for a "raster" Viewerdisplaydata `tool` is different from a "contour" Viewerdisplaydata `tool` option record.

Colormaps

The display of rasters requires that the pixel values in a data source be mapped to colors selected from a pre-defined set of colors, or a "colormap." The Viewer `tool` provides colormaps via the Viewercolormapmanager `tool`, and there is one and only one Viewercolormapmanager `tool` per Viewer `tool`. Sixteen unique colormaps are available. Whilst generation of new colormaps by the user or *Glish* programmer is not available yet, the existing colormaps can be modified in several ways by directly using the Viewercolormapmanager `tool`, or its graphical user interface. The usual "slope-shift" colormap fiddling is provided, together with "contrast-brightness" fiddling. Both of these are also available from the controlbox of Viewerdisplaypanel `tools`. Further capabilities (available via only the Viewercolormapmanager `tool`) include the independent inversion of the red, green and blue components of the colors in the colormap.
Viewer `tool` colormaps are entities which can be used by on many Viewerdisplaypanel `tools` concurrently, and which share color resources when possible. They automatically resize themselves to use as many colors as they can, based both on the number of colors that the various Viewerdisplaypanel `tools` are set up to use, and on the number (and weightings) of other Viewer `tool` colormaps that are in use. For this reason, when multiple Viewerdisplaypanel `tools` are in use, registering or unregistering a raster-type Viewerdisplaydata `tool` on one Viewerdisplaypanel `tool` may well cause other Viewerdisplaypanel `tools` to refresh, as more or less colors become available for other colormaps in use. The programmer can reduce the resulting "flickering" by "holding" and "releasing" the Viewer `tools` that are in use. The user can control the number of colors that the various Viewerdisplaypanel `tools` are setup to use, and can modify the Viewer `tool` colormaps themselves, by fiddling and inverting components.

> Advanced note: The Viewer `tool` inserts a software layer above the hardware colormap of the screen in use. This is so that the user can choose to have as many colors as possible for the Viewerdisplaypanel `tools`, knowing they will get color flashing, or instead to use the system colormap, which invariably will limit the number of colors that are used

by the Viewer `tool`, but will prevent color flashing. Indeed, for a single Viewer `tool`, multiple Viewerdisplaypanel `tools` can be used: some of these may use "private" colormaps, and others may use the system colormap.

Once a Viewerdisplaypanel `tool` is on the screen, you cannot change whether it is using the system colormap or a private colormap. You *can*, however, change the number of colors it is actually making use of in its colormap. This can be useful in two ways: firstly, sometimes rasters look better if you use fewer colors and sometimes look better with more colors: so with this feature, you can dynamically reduce or increase the "smoothness" (in colorspace) of raster images. Secondly, let's say we made a Viewerdisplaypanel `tool` and asked it to use the system colormap: that is, we were happy to have fewer colors so that we could avoid color flashing. Sometime later, we start up a (non-AIPS++) image viewing program on the same screen, and because the system colormap is nearly over-subscribed, partly by the Viewerdisplaypanel `tool`, this external program decides to use a private colormap. This might not be satisfactory, so you can actually close that program, reduce the number of colors being used by your Viewerdisplaypanel `tool` (via a graphical user interface perhaps), and then re-start the external program. *You did not have to close the Viewerdisplaypanel* `tool` *to accomplish this!*

While a single Viewer `tool` can have many Viewerdisplaypanel `tools` operating, it only ever has one Viewercolormapmanager `tool`, whose colormaps are shared by all the Viewerdisplaypanel `tools`. Viewerdisplaypanel `tools` which are using the system X colormap (ie. that aren't causing color flashing) actually all share one contiguous section of the system X colormap. Whatever Viewer `tool` colormaps are needed are placed in this block of color cells, usually uniformly distributed. For example, suppose I have two Viewerdisplaypanel `tools` on screen, both using the same 80 cells of the system X colormap. If they both have raster Viewerdisplaydata `tools` registered for display, each of which has its own (different) Viewer `tool` colormap (say one is using "Rainbow 4" and the other "Hot Metal 2"), then each of the Viewer `tool` colormaps will get 40 cells each. The "Rainbow 4" colormap can be fiddled and inverted to your hearts content without ever affecting the raster drawn with the "Hot Metal 2" colormap. If I close one of the Viewerdisplaypanel `tools`, the other one will notice this, and stretch the colormap it is using back out to fill the 80 cells. Conversely, if I open another Viewerdisplaypanel `tool` using the system X colormap, and display yet another raster using yet another Viewer `tool` colormap, the three required colormaps will be packed into the 80 cells (having lengths 27, 27 and 26).

Let's say, however, that with the two system X colormap Viewerdisplaypanel `tools` still open, I make a third Viewerdisplaypanel `tool`, but ask that it use a *private* colormap. I'm going to get color flashing, but I'm also going to get way more than 80 colors: I'll probably get around 250 on an 8-bit system, and around 1000 on a 10-bit system. If I go and register a third raster Viewerdisplaydata `tool` in this Viewerdisplaypanel `tool`, and ask it to use a different Viewer `tool` colormap, eg. "Greyscale 1", then the two existing Viewerdisplaypanel

`tools` are *not* affected, and the "Greyscale 1" colormap will fill the 250 (or more) colors available in the third Viewerdisplaypanel `tool`. But this "Greyscale 1" colormap can be mapped into more than one X colormap! So if I choose this colormap to be used for one of the first two Viewerdisplaydata `tools`, then it will replace, say, "Rainbow 4", in the 80 cells of the system X colormap we are using, but it will remain occupying 250 cells in the private colormap Viewerdisplaypanel `tool`. Yet they are the same Viewer `tool` colormap, so modifying the "Greyscale 1" colormap in any way, either via the Viewercolormapmanager `tool`, its graphical user interface, or a Viewerdisplaypanel `tool` controlbox, will cause equivalent changes in the display of both rasters that are being drawn with "Greyscale 1".

Resources

The Viewer `tool` looks for and recognizes a number of variables in the system and user `.aipsrc` files. They are listed in aipsrcdata.

`Tools` in this module

- The Vieweranimator `tool`, providing services to (manually or automatically) step through multi-dimensional images.
- The Viewercanvasmanager `tool`, which provides services to control the appearance of the canvases owned by Viewerdisplaypanel `tools`.
- The Viewercanvasprintmanager `tool`, which provides printing services to for the canvases of Viewerdisplaypanel `tools`.
- The Viewercolormapmanager `tool`, which is used to generate and modify colormaps for use by other `tools` in this module.
- The Viewerdatamanager `tool`, which provides a simple graphical user interface for constructing and destroying Viewerdisplaydata `tools`from files and `tools`.
- The Viewerdisplaydata `tool`, which is used to render false color images or contour maps from various data sources (AIPS++ Images or *Glish* arrays). To become visible to the user, Viewerdisplaydata `tools` must be registered with one or more Viewerdisplaypanel `tools`, but they can be created, manipulated and destroyed without ever being registered. More generally, the Viewerdisplaydata `tool` embodies some data source, and some method for drawing that data.
- The Viewerdisplaypanel `tool`, whose fundamental purpose is to provide a "canvas" on which Viewerdisplaydata `tools` can draw themselves. This `tool` is where users should expect to see their data displayed.
- The Viewerannotations tool which can be used to annotate display panels.

- The Viewerslicepanel `tool`, which provides 3D slicing capabilities. Apart from this the interface is very similar to the Viewerslicepanel `tool`.

Example

Here is a trivial example to give you the flavour of using a Viewer `tool`. Suppose we have an AIPS++ image on disk and wish to view it on the screen then generate a "quick" PostScript file of the image, with axis labels:

```
- include 'viewer.g';                                                    # 1
- mdd := dv.loaddata('myaips++image', 'raster');                         # 2
- mdd.setoptions([axislabelswitch=T]);                                   # 3
- mdp := dv.newdisplaypanel();                                           # 4
- mdp.register(mdd);                                                     # 5
- mdp.canvasprintmanager().writeps('mydiskimage.ps', dpi=300);           # 6
- mdd.done();                                                            # 7
- mdp.done();                                                            # 8
```

1. This gives us access to the Viewer `tool` code, and in particular, the **defaultviewer** `tool`.
2. Create a Viewerdisplaydata `tool` which is a raster representation of the AIPS++ Image with the given name.
3. Turn axis labelling on.
4. Create a Viewerdisplaypanel `tool` for rendering.
5. Register the **mdd** Viewerdisplaydata `tool` on the **mdp** Viewerdisplaypanel `tool`. At this point, the raster should now be visible somewhere on your screen.
6. Print a copy of the raster on the screen to the named PostScript file, at a nice high resolution of 300 dots per inch.
7. Destroy the Viewerdisplaydata `tool`.
8. Destroy the Viewerdisplaypanel `tool`.

Tools

skycatalog	Tool to create and manipulate Viewer `tool` readable skycatalog tables
viewer	Tool for viewing AIPS++ images and arrays
viewerannotations	Tool for drawing annotations onto a displaypanel.
viewercanvasmanager	Tool for managing the canvases of Viewerdisplaypanel `tools`
viewercanvasprintmanager	Tool for managing the printing of canvases of Viewerdisplaypanel `tools`
viewercolormapmanager	Tool for managing colormaps used by the Viewer tool
viewerdatamanager	Tool for importing data into the Viewer

viewerdisplaydata	Tool for managing the drawing of AIPS++ data
viewerdisplaypanel	Tool for providing drawing windows (canvases)
viewerslicedp	Tool for providing 3D slice canvases
viewerstandardgui	Interface for operating Viewer `tool` graphical user interfaces

Functions

is_viewer	Is the argument a viewer tool?

1.3.1 viewer - Tool
Package display
Module viewer

Tool for viewing AIPS++ images and arrays
include "viewer.g"

Constructors

viewer	Create a Viewer `tool`

Functions

alldisplaydatas	Return a list of all Viewerdisplaydata `tools` known to this Viewer `tool`
alldisplaypanels	Return a list of all Viewerdisplaypanel `tools` known to this Viewer `tool`
colormapmanager	Return the Viewercolormapmanager `tool` for this Viewer `tool`
deleteall	Delete all Viewerdisplaydata `tools`
deletedata	Delete a Viewerdisplaydata `tool`
disable	Disable the GUIs of all Viewerdisplaypanel `tools` and Viewerdatamanager
done	Close down (destroy) this Viewer `tool`
emittoolkitlist	Emit an event describing the toolkitlist for this Viewer `tool`
enable	Enable the GUIs of all Viewerdisplaypanel `tools` and Viewerdatamanager
gui	Create and show a new Viewerdatamanager `tool` and/or Viewerdisplaypar
hold	Prevent refreshes occuring on all Viewerdisplaypanel `tools`
loaddata	Create a Viewerdisplaydata `tool` for future use
newcolormapmanagergui	Create and optionally display a new GUI to control this Viewer `tool`'s Vie
newdatamanager	Create and show a new Viewerdatamanager `tool` on the screen
newdisplaypanel	Create and show a new Viewerdisplaypanel `tool` on the screen
release	Allow refreshes to occur on all Viewerdisplaypanel `tools`
title	Return the (construction) title of this `tool`
toolkit	Returns the toolkit used by this `tool`
type	Return the type of this `tool`
updatedatasets	Update the list of global images and arrays kept by the Viewer `tool`
widgetset	Return the Widgetserver `tool` used by this `tool`

Description

The Viewer `tool` is an AIPS++ tool which provides data display services. It can produce false color image and contour map representations of AIPS++ images or *Glish* arrays, rendered either on-screen or to PostScript files. Extensive control over the display of data is available from both graphical and programming interfaces. The Viewer `tool` is the primary interface to the viewer module of the Display package.

viewer - Constructor

Package	**display**
Module	**viewer**
Tool	**viewer**

Create a Viewer `tool`

Synopsis
viewer(title, deleteatexit, widgetset)

Arguments

title	Title of this Viewer `tool` which appears in windows Allowed:String Default: 'viewer'
deleteatexit	Delete this Viewer `tool` when *Glish* exits? Allowed:Boolean Default: T
widgetset	Widgetserver `tool` to use for the GUIs Allowed:Widgetserver `tool` Default: dws

Returns
Viewer `tool` or fail

Description

This is the only constructor for Viewer `tools`. It constructs (creates) a Viewer `tool` which can subsequently be used to construct and manage the various other component `tools` which might make up a data viewing application or the display services for an existing application.

---

viewer.alldisplaydatas - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Return a list of all Viewerdisplaydata `tools` known to this Viewer `tool`

Synopsis
alldisplaydatas()

Returns
list of Viewerdisplaydata `tools`

Description

This function simply returns a record containing all of the Viewerdisplaydata `tools` known to (managed by) this Viewer `tool`. The record contains as many top-level fields as there are Viewerdisplaydata `tools`, and each field is a Viewerdisplaydata `tool`. The fields are named by the identifying name of the corresponding Viewerdisplaydata `tool`, ie. the return value of the viewer:viewerdisplaydata.name function.

---

viewer.alldisplaypanels - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Return a list of all Viewerdisplaypanel `tools` known to this Viewer `tool`

Synopsis
alldisplaypanels()

Returns
list of Viewerdisplaypanel `tools`

Description

This function simply returns a record containing all of the Viewerdisplaypanel `tools` known to (managed by) this Viewer `tool`. The record contains as many top-level fields as there are Viewerdisplaypanel `tools`, and each field is a Viewerdisplaypanel `tool`. The fields are named by increasing integers.

---

viewer.colormapmanager - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Return the Viewercolormapmanager `tool` for this Viewer `tool`

Synopsis
colormapmanager()

Returns
Viewercolormapmanager `tool`

Description

Every Viewer `tool` has one and only one Viewercolormapmanager `tool` associated with it. It is constructed at construction time of the Viewer `tool`. This function returns the Viewercolormapmanager `tool` for this Viewer `tool`. More details on the functionality of a Viewercolormapmanager `tool` can be found in the documentation for Viewercolormapmanager `tool`.

---

viewer.deleteall - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Delete all Viewerdisplaydata `tools`

Synopsis
deleteall(doneit, quiet)

Arguments

doneit	truly delete the Viewerdisplaydata `tools`? Allowed:Boolean Default: T
quiet	prevent emission of updated list of Viewerdisplaydata `tools`? Allowed:Boolean Default: F

Returns
None

Description

This function simply calls the `deletedata` function for every Viewerdisplaydata `tool` managed by (known to) this Viewer `tool`. It is normally called when the Viewer `tool` is dying.

---

viewer.deletedata - Function

Package **display**
Module **viewer**
Tool **viewer**

Delete a Viewerdisplaydata `tool`

Synopsis
deletedata(displaydata, doneit, quiet)

Arguments

displaydata	Viewerdisplaydata `tool` to delete Allowed:Viewerdisplaydata `tool`
doneit	truly delete the Viewerdisplaydata `tool`? Allowed:Boolean Default: T
quiet	prevent emission of updated list of Viewerdisplaydata `tools`? Allowed:Boolean Default: F

Returns
None

Description

This function should be used to delete Viewerdisplaydata `tools` that were created by this Viewer `tool`. It will ensure that the Viewerdisplaydata `tool` is unregistered from all Viewerdisplaypanel `tools`, and then if `doneit=T` (the default), will actually free the resources used by the Viewerdisplaydata `tool` by calling its `done` function. The `quiet` parameter should only be `T` when the entire Viewer `tool` is closing done via its own `done` function.

---

viewer.disable - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Disable the GUIs of all Viewerdisplaypanel `tools` and Viewerdatamanager `tools`

Synopsis
disable()

Returns
none

Description

This function is used to disable the GUIs of all the Viewerdisplaypanel `tools` and all the Viewerdatamanager `tools` that are managed by this Viewer `tool`. This differs from the `hold` function in that refreshes of the displays will still be allowed, but the user will not be able to interact with the GUIs themselves. The GUIs will be re-enabled when an equal number of calls are made to the `enable` function. Actually, the GUIs can only effectively be disabled (greyed out) when they created their own top-level frames, so for applications where Viewerdisplaypanel `tools` for example are embedded in externally "owned" frames, this function does nothing.

---

viewer.done - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Close down (destroy) this Viewer `tool`

Synopsis
done()

Returns
T

Description

Calling this function frees up the resources used by this Viewer `tool`, and renders it useless. This function should be called when you are finished with a particular Viewer `tool`, and is also called when *Glish* exits, if the `deleteatexit` argument to the Viewer `tool` constructor was T. All GUIs (including Viewerdisplaypanel `tools`) created by this Viewer `tool` will be removed from the display, and all Viewerdisplaydata `tools` managed by this Viewer `tool` will be destroyed. Any Images used to construct those Viewerdisplaydata `tools` will of course remain on disk. Depending on how imaginatively you have used the Viewer `tool`, this function can take a few seconds to complete.

---

viewer.emittoolkitlist - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Emit an event describing the toolkitlist for this Viewer `tool`

Synopsis
emittoolkitlist()

Returns
None

Description

This function simply makes the Viewer `tool` emit an event with the name 'toolkit', with the event value being the return value of a call to the `toolkitlist` function. Typically this is used internally to synchronise all GUIs which display a list of the available tools and their mappings.
This function is normally used internally only.

---

viewer.enable - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Enable the GUIs of all Viewerdisplaypanel `tools` and Viewerdatamanager `tools`

Synopsis
enable()

Returns
none

Description

This function is used to enable the GUIs of all the Viewerdisplaypanel `tools` and all the Viewerdatamanager `tools` that are managed by this Viewer `tool`. This differs from the `release` function in that it does not affect refreshes of the display, but does affect whether users can interact with the GUIs themselves. The GUIs will only be re-enabled if the number of calls made to this function is equal to that made to the `disable` function.

---

viewer.hold - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Prevent refreshes occuring on all Viewerdisplaypanel `tools`

Synopsis
hold()

Returns
none

Description

This function is used to prevent refreshing occuring on all Viewerdisplaypanel `tools` that are managed by this Viewer `tool`. Refreshes will be re-enabled when an equal number of calls have been made to the `release` function. Typically, this is used internally when Viewerdisplaydata `tools` are being deleted or registered or unregistered from Viewerdisplaypanel `tools`. Programmers should make use of the `hold` and `release` functions when performing multiple operations before a refresh is desired.

---

viewer.gui - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Create and show a new Viewerdatamanager `tool` and/or Viewerdisplaypanel `tool` on the screen

Synopsis
gui(datamanager, displaypanel)

Arguments

datamanager	Show a new Viewerdatamanager `tool`? Allowed:Boolean Default: T
displaypanel	Show a new Viewerdisplaypanel `tool`? Allowed:Boolean Default: T

Returns
Viewerdatamanager `tool`

Description

This function is a simple front-end to the newdatamanager and newdisplaypanel functions. If the `datamanager` argument is T, then a new Viewerdatamanager `tool` graphical user interface will be placed on the screen in its own frame. Likewise, if the `displaypanel` argument is T, then a new Viewerdisplaypanel `tool` will be put on screen in a frame of its own.
This function is provided partly for convenience, but mostly to suit the requirements of the Toolmanager `tool`. See the individual functions newdatamanager and newdisplaypanel for further information.

---

viewer.loaddata - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Create a Viewerdisplaydata `tool` for future use

Synopsis
loaddata(data, drawtype, autoregister)

Arguments

data	data to load Allowed:array or image object, string filename of image or measurement set
drawtype	type of Viewerdisplaydata `tool` to create Allowed:'raster', 'contour', 'vector', 'marker', or 'sky-catalog'
autoregister	auto-register this on any auto-register Viewerdisplay-panel `tools`? Allowed:Boolean Default: F

Returns
Viewerdisplaydata `tool`

Description

This function provides the main route to creating Viewerdisplaydata `tools`. Any Viewerdisplaydata `tool` created using this function will be managed by this Viewer `tool`. Supported file types besides image tools and arrays are:

- AIPS++ images (for maximum performance use this)
- FITS images
- Miriad images
- Gipsy images (via FITS conversion;you need to have GIPSY available to be able to convert it to FITS)
- AIPS++ measurement sets (only the 'raster' drawtype is allowed at present)

The arguments are straightforward, so for more information on what you can do with your Viewerdisplaydata `tool` once you've made it, see the documentation on Viewerdisplaydata `tools`. **Internal use:**This function uses the `createdata` function.

---

viewer.newcolormapmanagergui - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Create and optionally display a new GUI to control this Viewer `tool`'s Viewercolormapmanager `tool`

Synopsis
newcolormapmanagergui(parent, show, hasdismiss, hasdone, widgetset)

Arguments

parent	parent frame in which to place the Viewerdatamanager `tool` Allowed:agent or F Default: F
show	show the Viewerdatamanager `tool`? Allowed:Boolean Default: T
hasdismiss	add a "Dismiss" button to the GUI? Allowed:Boolean Default: F
hasdone	add a "Done" button to the GUI? Allowed:Boolean Default: F
widgetset	Widgetserver `tool` to use to build the GUI Allowed:Widgetserver `tool` or unset Default: unset

Returns
`tool` having Viewerstandardgui interface

Description

This function can be used to place on the screen a new GUI for controlling this Viewer `tool`'s Viewercolormapmanager `tool`. Careful use of the `hasdismiss, hasdone` and `parent` arguments can provide quite flexible control over how the user builds their GUI.
See the Viewercolormapmanager `tool` documentation for more information.

---

viewer.newdatamanager - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Create and show a new Viewerdatamanager `tool` on the screen

Synopsis
newdatamanager(parent, show, showload, showedit, hasdismiss, hasdone, widgetset)

Arguments

parent	parent frame in which to place the Viewerdatamanager `tool` Allowed: agent or F Default: F
show	show the Viewerdatamanager `tool`? Allowed: Boolean Default: T
showload	show the "Load data" rollup? Allowed: Boolean Default: F
showedit	show the "Adjust data" rollup? Allowed: Boolean Default: F
hasdismiss	add a "Dismiss" button to the GUI? Allowed: Boolean Default: F
hasdone	add a "Done" button to the GUI? Allowed: Boolean Default: F
widgetset	Widgetserver `tool` to use to build the GUI Allowed: Widgetserver `tool` or unset Default: unset

Returns
Viewerdatamanager `tool`

Description

This function is generally used to place a new Viewerdatamanager `tool` on the screen. By judicious use of the `parent, hasdismiss` and `hasdone` arguments the appearance of the Viewerdatamanager `tool` can be controlled quite carefully. The Viewerdatamanager `tool` is a relatively simple graphical user interface which simplifies the construction and editing of Viewerdisplaydata `tools`.

---

viewer.newdisplaypanel - Function

Package **display**
Module **viewer**
Tool **viewer**

Create and show a new Viewerdisplaypanel `tool` on the screen

Synopsis

newdisplaypanel(parent, width, height, nx, nx, maptype, newcmap, mincolors, maxcolors, autoregister, holdsdata, show, hasgui, guihasmenubar, guihascontrolbox, guihasanimator, guihasbuttons, hasdismiss, hasdone, isolationmode, widgetset, slicepanel)

Arguments

parent	parent frame in which to place the Viewerdisplaypanel `tool` Allowed: agent or F Default: F
width	width of the drawing area on screen Allowed: Integer Default: 400
height	height of the drawing area on screen Allowed: Integer Default: 300
nx	number of panels in x Allowed: Integer Default: 1
nx	number of panels in y Allowed: Integer Default: 1
maptype	how colors will be addressed on this Viewerdisplaypanel `tool` Allowed: 'index' 'rgb' 'hsv' Default: 'index'
newcmap	install a private colormap? Allowed: Boolean or unset Default: unset
mincolors	minimum number of colors to allocate Allowed: Integer or unset Default: unset
maxcolors	maximum number of colors to allocate Allowed: Integer or unset Default: unset
autoregister	accept auto-register data on this Viewerdisplaypanel `tool`? Allowed: Boolean Default: F
holdsdata	is this Viewerdisplaypanel `tool` going to hold data? Allowed: Boolean Default: T
show	show the Viewerdisplaypanel `tool`? Allowed: Boolean Default: T
hasgui	adorn the Viewerdisplaypanel `tool` with elements of a graphical user interface? Allowed: unset or Boolean Default: unset
guihasmenubar	add the menubar if a GUI is added? Allowed: Boolean Default: T
guihascontrolbox	add the controlbox if a GUI is added? Allowed: Boolean Default: T
guihasanimator	add the animator tapedeck if a GUI is added? Allowed: Boolean Default: T
guihasbuttons	add the button bar if a GUI is added? Allowed: Boolean Default: T

Returns
Viewerdisplaypanel `tool`

Description

This function is generally used to place a new Viewerdisplaypanel `tool` on the screen. By judicious use of the `parent, hasdismiss` and `hasdone` arguments the appearance of the Viewerdisplaypanel `tool` can be contolled quite carefully. The `guihas*` arguments control the adornment of the Viewerdisplaypanel `tool` with various graphical user interface components. The `hasdismiss` and `hasdone` arguments can be used to specify what type of "Dismiss" button is placed on the Viewerdisplaypanel `tool`, if any. For further information, see the documentation on the Viewerdisplaypanel `tool`constructor, and its addgui function.
The `newcmap` argument is particular to this function, and parsed before it is passed on to the Viewerdisplaypanel `tool` constructor. If it is `unset`, and a parent frame is *not* given, (ie. `parent=F`), then this function will first attempt to construct a Viewerdisplaypanel `tool` using ("sharing") the system colormap, and if that fails, it will try again with a private colormap. If the `newcmap` argument is a boolean, then that is passed directly to the Viewerdisplaypanel `tool` constructor. The last argument `slicepanel` controls if the new tool should be a slicepanel to enable 3D slicing.

---

viewer.release - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Allow refreshes to occur on all Viewerdisplaypanel `tools`

Synopsis
release()

Returns
none

Description

This function is used to re-establish refreshing occuring on all Viewerdisplaypanel `tools` that are managed by this Viewer `tool`. Refreshes

will be re-enabled when an equal number of calls have been made to this function as have been made to the `hold` function. Typically, this is used internally when Viewerdisplaydata `tools` are being deleted or registered or unregistered from Viewerdisplaypanel `tools`. Programmers should make use of the `hold` and `release` functions when performing multiple operations before a refresh is desired.

---

viewer.title - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Return the (construction) title of this `tool`

Synopsis
title()

Returns
string

Description

This function returns the title that was specified at construction of this Viewer `tool`. It is ordinarily used only to generate title text for graphical user interfaces for this Viewer `tool`.

---

viewer.toolkit - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Returns the toolkit used by this `tool`

Synopsis
toolkit()

Returns

Viewer toolkit

Description

This function returns the Viewer toolkit of the current Viewer `tool`, which defines which tools (eg. Zoomer tool) are assigned to which mouse buttons and which tools appear in the controlbox. *This function is normally used internally only.*

---

viewer.type - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Return the type of this `tool`

Synopsis
type()

Returns
'viewer'

Description

This function simply returns the string `'viewer'`, which can be used to identify this `tool` as a Viewer `tool`.

---

viewer.updatedatasets - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Update the list of global images and arrays kept by the Viewer `tool`

Synopsis
updatedatasets()

Returns
none

Description

This function is quite specific in its use. Its purpose is to search for global *Glish* symbols which are either *Glish* arrays or Image `tools`, and to store them in an internal list. This list is used by other Viewer `tool` functions. After creating the updated list, it emits an event called 'datasets_updated' so that any interested parties are informed that new data might be available for their own lists. For example, the Viewerdatamanager `tool` listens for this event so that its list of data remains current. This function might also be used by a programmer after a global Image `tool` is explicitly constructed in their code.

---

viewer.widgetset - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Return the Widgetserver `tool` used by this `tool`

Synopsis
widgetset()

Returns
Widgetserver `tool`

Description

This function simply returns the Widgetserver `tool` that is used by this Viewer `tool` to build GUIs. It is generally reserved for internal use.

1.3.2 viewercanvasmanager - Tool

Package	**display**
Module	**viewer**

Tool for managing the canvases of Viewerdisplaypanel `tools`
include "viewercanvasmanager.g"

Constructors

viewercanvasmanager	Create a Viewercanvasmanager `tool`

Functions

done	Close down (destroy) this Viewercanvasmanager `tool`
getoptions	Get the current canvas settings
gui	Create a new graphical user interface for this `tool`
restoreoptions	Restore a previously saved state of this `tool`
saveoptions	Save the state of this `tool`
setoptions	Install new canvas settings
type	Return the type of this `tool`
viewer	Return the Viewer `tool` which manages this `tool`

Description

This `tool` provides services to manage and manipulate the drawing canvases of Viewerdisplaypanel `tools`. At present, this means adjusting margin spacing (to make room for axis labels, for example), selecting the number of panels for channel maps, selecting screen or paper colors (that is, white-on-black or black-on-white), and adjusting the number of colors used by colormaps on the Viewerdisplaypanel `tools`.
The graphical user interface provided by the Viewercanvasmanager `tool` is shown in Figure 1.1. A Viewercanvasmanager `tool` GUI for a Viewerdisplaypanel `tool` can be placed on the screen by selecting Canvas manager from the File menu of the Viewerdisplaypanel `tool`, or by invoking the newcanvasmanagergui function of the Viewerdisplaypanel `tool`. A single Viewercanvasmanager `tool` can have multiple GUIs on the screen at any one time, but this can become confusing to the user. The Viewercanvasmanager `tool` GUI makes use of an autogui to allow modification of the various canvas parameters.

Graphical User Interface

The graphical user interface consists of three rollups:

- Geometry This rollup controls the margins around the image. These are reserved for axis labels. If you don't use labelling you can set them to 0.
- Layout This rollup is only of interest to you if you want to create channel maps, i.e. draw frames of consecutive channels of a data cube on one display. These will be arranged from top-left to right-bottom, starting at

the frame selected in the animator. The sliders control the number of displays in x and y as well as the space between them. Note that this space is on top of the margins in Geometry. To use these appropriately set all margins to zero. This rollup won't show up if a viewercanvasmanager tool is associated with a viewerslicedp tool.

- Basic settings Basic settings control the numer of colors (index mode) and the background color of the display. The background color by default is black, but the options `Use paper colors` can be set to true so that it would look like a PS print out. The orientation of Color Wedges can also be selected here.

Canvas settings can be saved to and restored from the AIPS++ "inputs table" using the Save and Restore buttons.
The viewercanvasmanager tool is context sensitive to some degree. A Viewerdisplaypanel in RGB mode for example has a slightly different interface and on a True Color display in RGB mode you won't get an option to modify the number of colors.

viewercanvasmanager - Constructor

Package	**display**
Module	**viewer**
Tool	**viewercanvasmanager**

Create a Viewercanvasmanager `tool`

Synopsis
viewercanvasmanager(displaypanel)

Arguments

displaypanel	The Viewerdisplaypanel `tool` which will use this Viewercanvasmanager `tool` Allowed:Viewerdisplaypanel `tool`

Returns
Viewercanvasmanager `tool` or fail

Description

This is the only constructor for Viewercanvasmanager `tools`. It constructs (creates) a Viewercanvasmanager `tool` which can subsequently be used to provide canvas management to a Viewer `tool`.
This constructor should not be called by users: it is called during construction of Viewerdisplaypanel `tools`. The Viewercanvasmanager `tool` for a given Viewerdisplaypanel `tool` can be retrieved by calling its canvasmanager function.

---

viewercanvasmanager.type - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasmanager**

Return the type of this `tool`

Synopsis
type()

Returns
'viewercanvasmanager'

Description

This function simply returns the string `'viewercanvasmanager'`, which can be used to identify this `tool` as a Viewercanvasmanager `tool`. It is generally reserved for internal use.

---

viewercanvasmanager.viewer - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasmanager**

Return the Viewer `tool` which manages this `tool`

Synopsis
viewer()

Returns
Viewer `tool`

Description

This function simply returns the Viewer `tool` which owns the Viewerdisplaypanel `tool` that constructed and uses this Viewercanvasmanager `tool`. It is generally reserved for internal use.

---

viewercanvasmanager.done - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasmanager**

Close down (destroy) this Viewercanvasmanager `tool`

Synopsis
done()

Returns
T

Description

If you no longer need to use this Viewercanvasmanager `tool`, calling this function will free up its resources. Any graphical user interfacesassociated with this `tool` will be removed from the screen.
It is unwise to close down a Viewercanvasmanager `tool` by an explicit call to this function. It is generally reserved for internal use by the Viewerdisplaypanel `tool` which uses this Viewercanvasmanager `tool`.

---

viewercanvasmanager.gui - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasmanager**

Create a new graphical user interface for this `tool`

Synopsis
gui(parent, show, hasdismiss, hasdone, widgetset)

Arguments

parent	parent frame for GUI Allowed:agent or F Default: F
show	map the GUI to the screen immediately? Allowed:Boolean Default: T
hasdismiss	place a 'Dismiss' button in the GUI? Allowed:Boolean Default: F
hasdone	place a 'Done' button in the GUI? Allowed:Boolean Default: F
widgetset	Widgetserver `tool` to use for the GUI Allowed:Widgetserver `tool` Default: dws

Returns
`tool` having Viewerstandardgui interface

Description

Calling this function will create and return a new GUI which can be used to view and edit the canvas settings that are managed by this Viewercanvasmanager `tool`. By judicious use of the function arguments, the graphical user interface can optionally be placed within existing frames, and left off-screen for later use. The interface of the return value conforms to the Viewerstandardgui interface.

---

viewercanvasmanager.getoptions - Function

Package **display**
Module **viewer**
Tool **viewercanvasmanager**

Get the current canvas settings

Synopsis
getoptions()

Returns
autogui full-form parameter record

Description

This function can be used to determine the current canvas settings according to this Viewercanvasmanager `tool`. The returned record is simply a full-form autogui parameter record (see the autogui documentation).
The parameters that make up the returned record are described in the documentation for the getoptions function of the Viewerdisplaypanel `tool`.

---

viewercanvasmanager.setoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasmanager**

Install new canvas settings

Synopsis
setoptions(options)

Arguments

options	autogui short-form parameter record describing the requested state Allowed:record

Returns
T

Description

This function should be used to set the state of the canvas managed by this `tool`, according to the field names of the record returned by the getoptions `tool function`. Installing new options on a Viewercanvasmanager `tool` will

generally cause the refresh (that is, a total re-drawing) of the Viewerdisplaypanel `tool` which it manages.

---

viewercanvasmanager.saveoptions - Function

Package **display**
Module **viewer**
Tool **viewercanvasmanager**

Save the state of this `tool`

Synopsis
saveoptions(setname)

Arguments

setname	name under which to store this state Allowed:string

Returns
T (success) or F (failure)

Description

This function saves the current state (options) of this Viewercanvasmanager `tool` in the AIPS++ `Inputs Table`. The options are stored as inputs for the Viewercanvasmanager `tool`, with the "method name" set to the `setname` argument given in the call to this function.

Example
An example should show how this can be used:

```
- mv := viewer();
- mdp := mv.newdisplaypanel();
- mcm := mdp.canvasmanager();
- mcm.setoptions([leftmarginspacepg=0]);
- mcm.saveoptions('my canvas options');
```

---

viewercanvasmanager.restoreoptions - Function

Package **display**
Module **viewer**
Tool **viewercanvasmanager**

Restore a previously saved state of this `tool`

Synopsis
restoreoptions(setname)

Arguments

setname	name of state to retrieve Allowed:string

Returns
T (success) or F (failure)

Description

This function tries to retrieve a previously saved state (options) of the given name, and if successful, applies them to this Viewercanvasmanager `tool`. The `setname` argument must correspond to a previously saved set of options, otherwise this function will fail.

1.3.3 viewercanvasprintmanager - Tool

Package **display**
Module **viewer**

Tool for managing the printing of canvases of Viewerdisplaypanel `tools` include "viewer.g"

Constructors

viewercanvasprintmanager	Create a Viewercanvasprintmanager `tool`

Functions

done	Close down (destroy) this Viewercanvasprintmanager `tool`
gui	Create a new graphical user interface for this `tool`
type	Return the type of this `tool`
viewer	Return the Viewer `tool` which owns this `tool`
writeps	Write a PostScript file
writexpm	Write an X11 Pixmap file

Description

This `tool` provides services to produce copies of the drawing canvases of Viewerdisplaypanel `tools`. PostScript and bitmap output is available at present.

Graphical User Interface This tool controls two forms of hardcopy output:

PostScript for printing, and X-Windows pixmap format which can be easily converted to GIF, JPEG or other bitmap formats. To initiate printing of what is visible on the Viewerdisplaypanel `tool` simply press the Print... button at the bottom of the window, or select Print manager... from the File menu. This will bring up the GUI for this tool which allows you to set some basic output parameters, most of which are only important for PostScript output (as indicated by the *[PS]* prefix:

- Output file

  This can be left `<unset>`, and the Viewer will automatically create a unique filename for your output. Otherwise, just type in a specific name for your output file.

- [PS] Output media

  You may select to print to A4 or US Letter paper.

- [PS] Orientation

  Selection from portrait or landscape to orient the output on the selected paper.

- [PS] Resolution

  You may choose the resolution of your output here. Bear in mind that the size of the output file for raster images will generally grow as the square of this number, and likewise the speed with which the hardcopy will be generated and printed. For test output, generally leave the resolution quite low, and only increase it to your printer resolution for final hardcopy output. This setting enables you to tune the output to the exact resolution of your hardcopy device.

- [PS] Magnification

  You can scale your output by any factor between 0 and 1.

- [PS] Write EPS format?

  Set this option to True if you want encapsulated PostScript output.

Once you have filled in the desired options, you need to hit one of three buttons at the bottom of the Print manager GUI:

- Save XPM

  This button will write an X-Windows Pixmap copy of the Viewerdisplaypanel `tool` contents to a file on disk. The output will be an *exact* copy of what is displayed on-screen, and then a utility like 'xv' can be used to convert the output file to a GIF file, or a JPEG file, or whatever takes your fancy.

- Save PS

  This button will write a PostScript file which aims to duplicate what is seen on-screen in a higher-resolution mode suitable for printing. Every attempt is made to preserve the aspect ratio of the on-screen display.

- Print

  This button will first write a PostScript file as if the Save PS button had been pressed, and will then bring up a simple window which allows you to direct the output to a printer, to a previewer, or to cancel the operation.

Press Dismiss to cancel printing.

viewercanvasprintmanager - Constructor

Package	**display**
Module	**viewer**
Tool	**viewercanvasprintmanager**

Create a Viewercanvasprintmanager `tool`

Synopsis
viewercanvasprintmanager(displaypanel)

Arguments

displaypanel	The Viewerdisplaypanel `tool` which will use this Viewercanvasprintmanager `tool` Allowed:Viewerdisplaypanel `tool`

Returns
Viewercanvasprintmanager `tool` or fail

Description

This is the only constructor for Viewercanvasprintmanager `tools`. It constructs (creates) a Viewercanvasprintmanager `tool` which can subsequently be used to provide canvas printing facilities to a Viewerdisplaypanel `tool`. **This constructor should not be called by users:** it is called during construction of Viewerdisplaypanel `tools`. The Viewercanvasprintmanager `tool` for a given Viewerdisplaypanel `tool` can be retrieved by calling its canvasprintmanager function.

---

viewercanvasprintmanager.type - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasprintmanager**

Return the type of this `tool`

Synopsis
type()

Returns
'viewercanvasprintmanager'

Description

This function simply returns the string `'viewercanvasprintmanager'`, which can be used to identify this `tool` as a Viewercanvasprintmanager `tool`. It is generally reserved for internal use.

---

viewercanvasprintmanager.viewer - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasprintmanager**

Return the Viewer `tool` which owns this `tool`

Synopsis
viewer()

Returns
Viewer `tool`

Description

This function simply returns the Viewer `tool` which owns the
Viewerdisplaypanel `tool` that constructed and uses this
Viewercanvasprintmanager `tool`. It is generally reserved for internal use.

---

viewercanvasprintmanager.done - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasprintmanager**

Close down (destroy) this Viewercanvasprintmanager `tool`

Synopsis
done()

Returns
T

Description

If you no longer need to use this Viewercanvasprintmanager `tool`, calling this
function will free up its resources. Any graphical user interfacesassociated with
this `tool` will be removed from the screen.
It is unwise to close down a Viewercanvasprintmanager `tool` by an explicit call
to this function. It is generally reserved for internal use by the
Viewerdisplaypanel `tool` which uses this Viewercanvasprintmanager `tool`.

---

viewercanvasprintmanager.gui - Function

Package **display**
Module **viewer**
Tool **viewercanvasprintmanager**

Create a new graphical user interface for this `tool`

Synopsis
gui(parent, show, hasdismiss, hasdone, widgetset)

Arguments

parent	parent frame for GUI Allowed:agent or F Default: F
show	map the GUI to the screen immediately? Allowed:Boolean Default: T
hasdismiss	place a 'Dismiss' button in the GUI? Allowed:Boolean Default: F
hasdone	place a 'Done' button in the GUI? Allowed:Boolean Default: F
widgetset	Widgetserver `tool` to use for the GUI Allowed:Widgetserver `tool` Default: unset

Returns
`tool` having Viewerstandardgui interface

Description

Calling this function will create and return a new GUI which can be used to view and edit the printing settings of this Viewercanvasprintmanager `tool`. By judicious use of the function arguments, the graphical user interface can optionally be placed within existing frames, and left off-screen for later use. The interface of the return value conforms to the Viewerstandardgui interface.

---

viewercanvasprintmanager.writeps - Function

Package **display**
Module **viewer**
Tool **viewercanvasprintmanager**

Write a PostScript file

Synopsis
writeps(filename, media, landscape, dpi, zoom, eps)

Arguments

filename	name of file to write Allowed:string or unset Default: unset
media	paper type Allowed:'A4' 'LETTER' Default: 'A4'
landscape	print in landscape mode? Allowed:boolean Default: F
dpi	resolution of output in dots per inch Allowed:integer Default: 100
zoom	scaling of output Allowed:float in (0,1] Default: 1.0
eps	generate encapsulated PostScript output Allowed:boolean Default: F

Returns
fail or name of file written

Description

This function writes a PostScript version of the current view on display in the Viewerdisplaypanel `tool` for which this `tool` provides canvas print management facilities. The PostScript output will have the same aspect ratio as that of the Viewerdisplaypanel `tool` displayed on screen (provided the screen pixel aspect ratio is close to 1.0), but may differ slightly in color use

and resampling resolution for raster images. If the `filename` argument is left `unset`, then a sensible filename is generated by the Viewer `tool`. The `dpi` argument allows the caller to specify the resolution of the output.

---

viewercanvasprintmanager.writexpm - Function

Package **display**
Module **viewer**
Tool **viewercanvasprintmanager**

Write an X11 Pixmap file

Synopsis
writexpm(filename)

Arguments

filename	name of file to write Allowed:string or unset Default: unset

Returns
name of file written

Description

This function writes an X11 Pixmap version of the current view on display in the Viewerdisplaypanel `tool` for which this `tool` provides canvas print management facilities. The resultant file is an *exact* copy of the screen display: it is actually written directly from the internal copy of the Viewerdisplaypanel `tool` contents. If the `filename` argument is left `unset`, then a sensible filename is generated by the Viewer `tool`.

1.3.4 viewercolormapmanager - Tool

Package **display**
Module **viewer**

Tool for managing colormaps used by the Viewer tool
include "viewercolormapmanager.g"

Constructors

viewercolormapmanager	Create a Viewercolormapmanager `tool`

Functions

colormap	Return a handle to a named colormap
colormapnames	Return the names of the available colormaps
done	Close down (destroy) this Viewercolormapmanager `tool`
getbrightness	Return the brightness setting of a named colormap
getcontrast	Return the contrast setting of a named colormap
getinvertflags	Get the current red, green and blue invert state
gui	Create a new graphical user interface for this `tool`
nmaps	Return the number of colormaps available
reset	reset the named colormap to its default state
resetall	reset all colormaps to initial state
setbrightness	Set the brightness of a named colormap
setcontrast	Set the contrast of a named colormap
setinvertflags	Set the red, green and blue invert state of a colormap
type	Return the type of this `tool`
viewer	Return the Viewer `tool` which manages this `tool`

Description

This `tool` provides services to manage and manipulate colormaps for use by the Viewer `tool`. A colormap, or color look-up table, provides a set of colors which are normally used to represent varying intensity values in false color images (rasters). The Viewercolormapmanager `tool` provides sixteen basic colormaps, amongst them two greyscale-type colormaps and four rainbow-type colormaps. Each Viewer `tool` has one and only one Viewercolormapmanager `tool`, from which it obtains the names of the available colormaps, and also the actual colormaps when they are needed by Viewerdisplaydata `tools`.

Colormaps can be modified in a number of simple ways to achieve quite significant effects. Foremost, colormaps can be "fiddled:" a colormap can be stretched and shifted so that only a fraction of the colormap is actually used for drawing. Secondly, the contrast and brightness of the colors in the map can be modified. Thirdly, the individual red, green and blue components of the map can be independently inverted, leading to completely different colormaps.

Graphical User Interface

The graphical user interface provided by the Viewercolormapmanager `tool` is shown in Figure 1.2.
The left of Viewercolormapmanager `tool` GUI displays a listbox showing primary names of the colormaps known to the Viewer `tool` and therefore to the Viewercolormapmanager `tool`. Once a colormap is selected the sliders on the right-hand side can be used to modify various "fiddle" parameters of the selected colormap. The next three fields underneath can be used to invert the red, green and/or blue components in the selected colormap.
Finally, the `Dismiss` button is provided to remove this GUI from the screen.

viewercolormapmanager - Constructor

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Create a Viewercolormapmanager `tool`

Synopsis
viewercolormapmanager(viewer)

Arguments

viewer	The Viewer `tool` which will use this Viewercolormapmanager `tool` Allowed: Viewer `tool`

Returns
Viewercolormapmanager `tool` or fail

Description

This is the only constructor for Viewercolormapmanager `tools`. It constructs (creates) a Viewercolormapmanager `tool` which can subsequently be used to provide colormaps and colormap editing services to a Viewer `tool`.
This constructor should not be called by users: it is called during construction of the Viewer `tool`. The Viewercolormapmanager `tool` for a given Viewer `tool` can be retrieved by calling its colormapmanager function.

---

viewercolormapmanager.type - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Return the type of this `tool`

Synopsis
type()

Returns
'viewercolormapmanager'

Description

This function simply returns the string `'viewercolormapmanager'`, which can be used to identify this `tool` as a Viewercolormapmanager `tool`. It is generally reserved for internal use.

---

viewercolormapmanager.viewer - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Return the Viewer `tool` which manages this `tool`

Synopsis
viewer()

Returns
Viewer `tool`

Description

This function simply returns the Viewer `tool` which uses this Viewercolormapmanager `tool`, and probably constructed it. It is generally reserved for internal use.

---

viewercolormapmanager.done - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Close down (destroy) this Viewercolormapmanager `tool`

Synopsis
done()

Returns
T

Description

If you no longer need to use this Viewercolormapmanager `tool`, calling this function will free up its resources. Any graphical user interfacesassociated with this `tool` will be removed from the screen.
It is unwise to close down a Viewercolormapmanager `tool` by an explicit call to this function. It is generally reserved for internal use by the Viewer `tool` which uses this Viewercolormapmanager `tool`.

---

viewercolormapmanager.gui - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Create a new graphical user interface for this `tool`

Synopsis
gui(parent, show, hasdismiss, hasdone, widgetset)

Arguments

parent	parent frame for GUI Allowed:agent or F Default: F
show	map the GUI to the screen immediately? Allowed:Boolean Default: T
hasdismiss	place a 'Dismiss' button in the GUI? Allowed:Boolean Default: F
hasdone	place a 'Done' button in the GUI? Allowed:Boolean Default: F
widgetset	Widgetserver `tool` to use for the GUI Allowed:Widgetserver `tool` Default: dws

Returns
`tool` having Viewerstandardgui interface

Description

Calling this function will create and return a new GUI which can be used to view and edit the colormaps that are provided by this Viewercolormapmanager `tool`. By judicious use of the function arguments, the graphical user interface can optionally be placed within existing frames, and left off-screen for later use. The interface of the return value conforms to the Viewerstandardgui interface.

---

viewercolormapmanager.colormapnames - Function

Package **display**
Module **viewer**
Tool **viewercolormapmanager**

Return the names of the available colormaps

Synopsis
colormapnames()

Returns
vector of strings

Description

This function should be used to obtain a list of the colormaps that are available for use. The names can then be used to actually get hold of the colormaps themselves.

---

viewercolormapmanager.colormap - Function

Package **display**
Module **viewer**
Tool **viewercolormapmanager**

Return a handle to a named colormap

Synopsis
colormap(name)

Arguments

name	name of requested colormap Allowed:string

Returns
colormap agent or fail

Description

This function can be used to get hold of a specific colormap agent.
Once again, this is generally only used internally.

---

viewercolormapmanager.nmaps - Function

Package **display**
Module **viewer**
Tool **viewercolormapmanager**

Return the number of colormaps available

Synopsis
nmaps()

Returns
integer

Description

This function simply returns the number of colormaps that it is presently managing.

---

viewercolormapmanager.getbrightness - Function

Package **display**
Module **viewer**
Tool **viewercolormapmanager**

Return the brightness setting of a named colormap

Synopsis
getbrightness(name)

Arguments

name	name of colormap to query Allowed:string

Returns
float or fail

Description

This function returns the current brightness setting (in the range 0 to 1) of the named colormap.

---

viewercolormapmanager.setbrightness - Function

Package **display**
Module **viewer**
Tool **viewercolormapmanager**

Set the brightness of a named colormap

Synopsis
setbrightness(name, value)

Arguments

name	name of colormap to modify Allowed:string
value	new brightness value Allowed:float in range 0 to 1

Returns
T or fail

Description

This function sets the brightness of the named colormap to the specified value.

---

viewercolormapmanager.getcontrast - Function

Package **display**
Module **viewer**
Tool **viewercolormapmanager**

Return the contrast setting of a named colormap

Synopsis
getcontrast(name)

Arguments

name	name of colormap to query Allowed:string

Returns
float or fail

Description

This function returns the current contrast setting (in the range 0 to 1) of the named colormap.

---

viewercolormapmanager.setcontrast - Function

Package **display**
Module **viewer**
Tool **viewercolormapmanager**

Set the contrast of a named colormap

Synopsis
setcontrast(name, value)

Arguments

name	name of colormap to modify Allowed:string
value	new contrast value Allowed:float in range 0 to 1

Returns
T or fail

Description

This function sets the contrast of the named colormap to the specified value.

---

viewercolormapmanager.getinvertflags - Function

Package **display**
Module **viewer**
Tool **viewercolormapmanager**

Get the current red, green and blue invert state

Synopsis
getinvertflags(name)

Arguments

name	name of colormap to query Allowed:string

Returns
vector of booleans or fail

Description

This function returns a length 3 vector describing whether the red, green and blue components of the named colormap are inverted (T) or not (F).

---

viewercolormapmanager.setinvertflags - Function

Package **display**
Module **viewer**
Tool **viewercolormapmanager**

Set the red, green and blue invert state of a colormap

Synopsis
setinvertflags(name, value)

Arguments

name	name of colormap to modify Allowed:string
value	length 3 vector describing invert state Allowed:vector of booleans

Returns
T or fail

Description

This function sets the red, green and blue component invert states for the named colormap to those specified in the given length 3 vector.

---

viewercolormapmanager.reset - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

reset the named colormap to its default state

Synopsis
reset(name)

Arguments

name	name of colormap to restore to initial state Allowed:string

Returns
T or fail

Description

This function returns the contrast and brightness settings of the named colormap to 0.5, and sets each of the invert "flags" to F.

---

viewercolormapmanager.resetall - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

reset all colormaps to initial state

Synopsis
resetall()

Returns
T

Description

This function simply calls the `reset` function for each colormap known to this Viewercolormapmanager `tool`.

1.3.5 viewerdisplaydata - Tool

Package display
Module viewer

Tool for managing the drawing of AIPS++ data
include "viewerdisplaydata.g"

Constructors

viewerdisplaydata	Create a Viewerdisplaydata `tool`

Functions

addtouserchoicelist	Add a new possibility to an option which is of type 'userchoice'
classtype	Return the class type of this Viewerdisplaydata `tool`
ddd	Return the drawingdisplaydata agent
ddproxy	Return the true displaydata agent
displaytype	Return the displaytype
done	Close down (destroy) this Viewerdisplaydata `tool`
filename	Return a handle to the underlying image/array
getoptions	Return a record describing the state of this `tool`
hasbeam	has this Viewerdisplaydata `tool`a restoring beam defined
name	Return the (unique) name of this `tool`
newdisplaydatagui	Create a new graphical user interface for this `tool`
pixeltype	Return the data type of the pixels
restoreoptions	Restore a previously saved state of this `tool`
saveoptions	Save the state of this `tool`
setoptions	Set the state of this `tool`
type	Return the type of this `tool`
viewer	Return the Viewer `tool` which manages this `tool`
wedgedd	Return the Wedge displaydata agent
zlength	Return the length of the movie axes

Description

This `tool` is a fundamental component of the Viewer `tool`. It bundles up some piece of AIPS++ data together with functionality to draw that data, into a self-contained object which can be shuffled around by a Viewer `tool`, and manipulated by the user and/or programmer. For example, a specific Viewerdisplaydata `tool` may be a `tool` that knows how to draw a contour map (hereafter a 'contour') of a *Glish* array, or it may be a `tool` that can draw a false color image (hereafter a 'raster') of an AIPS++ Image.
The Viewerdisplaydata `tool` carries around instructions on how to draw itself, but it doesn't carry around a "canvas" on which to draw itself. Thus, to become visible to the user, a Viewerdisplaydata `tool` must be registered with one or more Viewerdisplaypanel `tools`. However, a Viewerdisplaydata `tool` may be created, manipulated and destroyed without ever being registered (and therefore displayed on-screen); why exactly anyone would like to do this is beyond me though! A Viewerdisplaydata `tool` is constructed from some AIPS++ data source, and a string defining the type of drawing, that is, 'raster' or 'contour'.
A Viewerdisplaydata `tool` is totally self-contained, and has "state". Its state consists of various attributes, some of which will not change during the lifetime of the `tool`, and many others which probably will. Examples of unchangeable state include the dimensionality of the AIPS++ Image or *Glish* array, and the type of drawing requested—a raster or a contour. Examples of changeable state include the many attributes which describe how the data should be translated into the requested drawing, such as the colormap to use for a raster, or the contour levels to use for a contour Viewerdisplaydata `tool`. The state of a Viewerdisplaydata `tool` can be modified from the command-line, or from a graphical user interface.
Normally, Viewerdisplaydata `tools` should be created with an appropriate call to a Viewer `tool`; many Viewerdisplaydata `tools` can be created and managed by a single Viewer `tool`.

Events

viewerdisplaydata - Constructor

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Create a Viewerdisplaydata `tool`

Synopsis

viewerdisplaydata(viewer, name, displaytype, data)

Arguments

viewer	the Viewer `tool` which will manage this `tool` Allowed:Viewer `tool`
name	a unique name for this Viewerdisplaydata `tool` Allowed:string
displaytype	a string describing the type of drawing Allowed:'raster', 'contour', 'simpleaxes' or 'skycatalog'
data	The data source to draw, either an AIPS++ Image name, or a *Glish* array Allowed:string or array

Returns
Viewerdisplaydata `tool` or fail

Description

This is the only constructor for Viewerdisplaydata `tools`. It constructs (creates) a Viewerdisplaydata `tool` which can subsequently be registered on (and hence displayed on) Viewerdisplaypanel `tools`.
The `displaytype` argument can be:

- `'raster'`, for which `data` can be a (float or double) *Glish* array having two or more dimensions, or a string giving the name of an AIPS++ Image on disk.
- `'contour'`, for which `data` can be a (float or double) *Glish* array having two or more dimensions, or a string giving the name of an AIPS++ Image on disk.
- `'simpleaxes'`, for which `data` must be tt '¡none¿'.
- `'skycatalog'`, for which `data` must be a string giving the name of an AIPS++ Table on disk.

This constructor should not be called by users: it is called by the loaddata function of the Viewer `tool`, which returns a Viewerdisplaydata `tool`.

---

viewerdisplaydata.addtouserchoicelist - Function

Package **display**
Module **viewer**
Tool **viewerdisplaydata**

Add a new possibility to an option which is of type 'userchoice'

Synopsis
addtouserchoicelist(vl)

Arguments

vl	record with fields 'param' which is the string identifier of the option, and 'newitem' which is the new option to add to the list. Allowed:record

Returns
T or fail

Description

This function is used to add a possibility to a choice list for display options which are of type 'userchoice'.

viewerdisplaydata.classtype - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the class type of this Viewerdisplaydata `tool`

Synopsis
classtype()

Returns
integer

Description

This function returns an integer indicating the class type of this Viewerdisplaydata `tool`. The possible return values are: `Display::Raster`,

`Display::Vector`, `Display::Annotation`, and `Display::CanvasAnnotation`. The return value is used by the Viewer `tool` to decide whether position tracks should be shown for this Viewerdisplaydata `tool`.

---

viewerdisplaydata.ddd - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the drawingdisplaydata agent

Synopsis
ddd()

Returns
agent

Description
This function retruns a DrawingDisplayData agent. This for example is used to draw the beam if it is defined.

viewerdisplaydata.ddproxy - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the true displaydata agent

Synopsis
ddproxy()

Returns
agent

Description

This function returns a handle to the true displaydata agent underlying this Viewerdisplaydata `tool`. **This function is for gurus only.**

viewerdisplaydata.displaytype - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the displaytype

Synopsis
displaytype()

Returns
'raster', 'contour', 'vector', 'marker', or 'skycatalog'

Description

This function returns the displaytype chosen to construct this Viewerdisplaydata `tool`.

viewerdisplaydata.done - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Close down (destroy) this Viewerdisplaydata `tool`

Synopsis
done()

Returns
T

Description

If you no longer need to use this Viewerdisplaydata `tool`, calling this function will free up its resources. Any Viewerdisplaypanel `tools` on which this Viewerdisplaydata `tool` is registered will automatically unregister this `tool` and refresh their displays. Any graphical user interfacesassociated with this `tool` will be removed from the screen.

---

viewerdisplaydata.filename - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return a handle to the underlying image/array

Synopsis
filename()

Returns
string or array

Description

This function returns the hadle to the underlying data structure which can be the filename of the underlying image/table or an array. Viewerdisplaydata `tool`.

viewerdisplaydata.getoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return a record describing the state of this `tool`

Synopsis
getoptions()

Returns
record

Description

This function returns a record which describes the internal state of this Viewerdisplaydata `tool`. The record has a format that is compatible with the format used by the Autogui `tool`, and indeed is used to build an Autogui `tool` in the `newdisplaydatagui` function of this `tool`. The record is also described in a bit more detail and from an Autogui `tool` point of view here.
These options on the type of the Viewerdisplaydata `tool`. The record for examplee is slightly different for 'raster' and 'contour' Viewerdisplaydata `tool`. A brief description of the fields which *may* exist in the returned record follows:

- `xaxis`: the axis name in the data to map to the X (horizontal) direction.
- `yaxis`: the axis name in the data to map to the Y (vertical) direction.
- `zaxis`: the axis name in the data to map to the Z (in-to-the-screen) direction. This axis is the "movie" axis.
- `haxisN`: the position along the Nth axis in arrays or Images having more than three axes.
- `aspect`: rule on how to control the aspect ratio of data pixels when they are mapped to display device pixels.
- `pixeltreatment`: whether pixels should be aligned with the edge of the canvas at their centres or edges.
- `axislabelswitch`: global switch to turn axis labelling on or off for this Viewerdisplaydata `tool`. Note that axis labels will only ever be drawn for the *first* registered Viewerdisplaydata `tool` for any given Viewerdisplaypanel `tool`.
- `titletext`: title text if axis labels are drawn.
- `titletextcolor`: color of the title text.
- `xaxistext`: label to appear along X axis.
- `xaxistextcolor`: color of X axis label text.
- `yaxistext`: label to appear along Y axis.
- `yaxistextcolor`: color of Y axis label text.
- `xgridtype`: rule for how to draw tickmarks or gridlines along the X axis.
- `xgridcolor`: color of X axis tickmarks and gridlines.
- `ygridtype`: rule for how to draw tickmarks or gridlines along the Y axis.
- `ygridcolor`: color of Y axis tickmarks and gridlines.

- `ticklength`: length of tickmarks (in mm?).
- `plotoutline`: draw a border around the plot?
- `plotoutlinecolor`: color of plot border.
- `labelcharsize`: size of characters for axis labelling.
- `labelposition`: the position of the world axes labels.
- `labelcharfont`: font for axis labelling.
- `labellinewidth`: line width for axis labelling.
- `axislabelpixelworld`: use pixel or world axis labels
- `axislabelabsrel`: relative or absolute coordinate labels
- `axislabeldirectionsystem`: the direction reference frame (e.g. J2000)
- `axislabeldirectionunit`: the unit to label the direction coordinate axes with
- `axislabelspectralunit`: the unit to label the spectral axis with
- `axislabelvelocitytype`: the velocity refernce frame (e.g. radio)
- `resample`: how to resample data pixels to screen pixels.
- `region`: the region (subimage) to display.
- `datamin`: the minimum data value to map to the colormap.
- `datamax`: the maximum data value to map to the colormap.
- `power`: scaling power for mapping data values to colors.
- `colormap`: which colormap to use.
- `wedge:` display a color wedge.
- `wedgeyaxistext`: the wedge axis label text, unset uses brighntes unit.
- `wedgeyaxistextcolor`: the wedge axis label text color.
- `wedgelabelcharfont`: font for wedge axis labelling.
- `wedgelabelcharsize`: size of characters for wedge axis labelling.
- `beam:` display the restoring beam.
- `beamcolor:` set the color of the beam.
- `beampos:` position the beam

- `beamlinewidth:` the linewidth for the beam
- `beamoutline:` draw the beam as an outline or filled
- `levels`: contour levels.
- `scale`: scale factor for the contour levels.
- `type`: whether the contour levels are absolute (in data units) or fractional (in range 0.0 to 1.0).
- `line`: linewidth of contours.
- `dashpos`: whether to dash negative-valued contours.
- `dashneg`: whether to dash positive-valued contours.
- `color`: contour color.
- `complexmode`: how to produce real data for display from a complex source.

Example
The best way to see the structure of the options record is to get hold of one and view it with the `recordbrowserwidget`. Alternatively, using the CLI only the misc module provides a function listfields

```
- include 'viewer.g';
- include 'recordbrowserwidget.g';
- mv := viewer();
- ar := array(1:137, 13, 17);
- mdd := mv.loaddata(ar, 'raster');
- opts := mdd.getoptions();
- mrb := recordbrowserwidget(therecord=opts);
# or
- dm.listnames(opts,F)
```

---

viewerdisplaydata.hasbeam - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

has this Viewerdisplaydata `toola` restoring beam defined

Synopsis
hasbeam()

Returns
T or F

Description

This function indicates if the underlying image has a restoring beam.

---

viewerdisplaydata.name - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the (unique) name of this `tool`

Synopsis
name()

Returns
string

Description

This function simply returns the (unique) name of this `tool` that was a construction argument.

---

viewerdisplaydata.newdisplaydatagui - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Create a new graphical user interface for this `tool`

Synopsis
newdisplaydatagui(parent, uneditable, show, hasdismiss, hasdone, widgetset)

Arguments

parent	parent frame for GUI Allowed: agent or F Default: F
uneditable	parameters which should not be editable Allowed: vector of strings Default: “”
show	map the GUI to the screen immediately? Allowed: Boolean Default: T
hasdismiss	place a 'Dismiss' button in the GUI? Allowed: Boolean Default: F
hasdone	place a 'Done' button in the GUI? Allowed: Boolean Default: F
widgetset	Widgetserver `tool` to use for the GUI Allowed: Widgetserver `tool` Default: dws

Returns
`tool` having Viewerstandardgui interface

Description

Calling this function will create and return a new GUI which can be used to adjust the state of this Viewerdisplaydata `tool`. By judicious use of the function arguments, the graphical user interface can optionally be placed within existing frames, and left off-screen for later use. The interface of the return value conforms to the Viewerstandardgui interface.
The `uneditable` argument should be a vector of strings listing the internal names of the parameters which should not be editable by the user in the graphical user interface. eg. `uneditable=''xaxis yaxis colormap''` would prevent the user from modifying the displayed X and Y axes and the colormap of this Viewerdisplaydata `tool` in the GUI: they could still modify these parameters from the command-line via the setoptions function.

---

viewerdisplaydata.pixeltype - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the data type of the pixels

Synopsis
pixeltype()

Returns
float or complex

Description

This function simply returns the data type of the pixels in the data. At the moment this can be complex or float.

---

viewerdisplaydata.restoreoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Restore a previously saved state of this `tool`

Synopsis
restoreoptions(setname)

Arguments

setname	name of state to retrieve Allowed:string

Returns
T (success) or F (failure)

Description

This function tries to retrieve a previously saved state (options) of the given name, and if successful, applies them to this Viewerdisplaydata `tool`. It is perfectly acceptable to save options from a contour `tool` and sooner or later apply them to a raster `tool`: only the options relevant to a raster will be used. The `setname` argument must correspond to a previously saved set of options.

---

viewerdisplaydata.saveoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Save the state of this `tool`

Synopsis
saveoptions(setname)

Arguments

setname	name under which to store this state Allowed:string

Returns
T (success) or F (failure)

Description

This function saves the current state (options) of this Viewerdisplaydata `tool` in the AIPS++ `Inputs Table`. The options are stored as inputs for the Viewerdisplaydata `tool`, with the "method name" set to the `setname` argument given in the call to this function.

Example
An example should show how this can be used:

```
- mv := viewer();
- mdd := mv.loaddata('myimage', 'raster');
- mdd.setoptions([power=1.6, aspect='flexible']);
- mdd.saveoptions('trial options');
```

viewerdisplaydata.setoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Set the state of this `tool`

Synopsis
setoptions(newopts, emit, id)

Arguments

newopts	options record describing requested state Allowed:record
emit	emit the options that have been installed Allowed:Boolean Default: T
id	option request identification tag Allowed:Numeric Default: -1

Returns
T

Description

This function should be used to set the internal state of a Viewerdisplaydata `tool`. The format of the `newopts` record is a short form of the record returned by the `getoptions` function, and simply consists of one or more top-level fields, whose values are the values that should be set for each option. The `emit` argument should rarely be anything other than `T`, since this is how other interested parties can find out when the Viewerdisplaydata `tool` has been adjusted. The `id` argument is reserved for internal use, and should *never* be explicitly listed in a call to this function, other than from calls from within the Viewer `tool` which manages this `tool`.
Installing new options in a Viewerdisplaydata `tool` will generally cause the refresh (that is, a total re-drawing) of any Viewerdisplaypanel `tools` on which this `tool` is registered. This is because modifying the options will modify the appearance of the drawing of this data. Furthermore, there are some options

(for example, `region` and `complexmode`) which, when adjusted, will cause *internal* changes to other options, normally `datamin` and `datamax` for raster `tools`.

Example
A simple example will help clarify things:

```
- mv := viewer();
- mdd := mv.loaddata('myimage', 'raster');
- mdd.setoptions([power=1.6, aspect='flexible']);
```

---

viewerdisplaydata.type - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the type of this `tool`

Synopsis
type()

Returns
'viewerdisplaydata'

Description

This function simply returns the string `'viewerdisplaydata'`, which can be used to identify this `tool` as a Viewerdisplaydata `tool`. It is generally reserved for internal use.

---

viewerdisplaydata.wedgedd - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the Wedge displaydata agent

Synopsis
wedgedd()

Returns
proxy agent

Description
This function is for gurus only. So there won't be anby description.

---

viewerdisplaydata.viewer - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the Viewer `tool` which manages this `tool`

Synopsis
viewer()

Returns
Viewer `tool`

Description

This function simply returns the Viewer `tool` which manages this Viewerdisplaydata `tool`, and probably constructed it. It is generally reserved for internal use.

---

viewerdisplaydata.zlength - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the length of the movie axes

Synopsis
zlength()

Returns
Integer

Description

This function simply returns the length of the movie (third) axis on this Viewerdisplaydata `tool`.

Option records

Here is a detailed description of the option record elements. This is based on how they appear in the Adjust.. GUI sorted by context.
These are the options available to *raster*, *contour* and *vector* Viewerdisplaydatas. Jump to display type specific options:

- raster
- contour
- vector
- marker
- skycatalog
- measurement set
- Display axes

  In this roll-up, adjustments can be made to select how the data is sliced for display, i.e. which axes of the data are mapped to X and Y on the screen. As well, a third axis (where appropriate) can be assigned to the Z (or movie) axis, which is the data axis along which progressive frames will be shown when a Viewerdisplaypanel is playing a sequence of frames. The new selection will not take place until the Apply button at the bottom of the adjustment window is pressed. If you have accidentally selected the same axis for more than one display axis, then this will be indicated by red crosses, and you should correct your error, and press Apply again.

- Hidden axes

  This roll-up is only present when the image or array has more than three axes, and is used to set the location in the data along all of the axes

which are not mapped to either X, Y or Z (movie) axes in the Display axes roll-up. Simply sliding any of the scales should result in immediate updates to any Viewerdisplaypanel on which this Viewerdisplaydata is registered.

Note: For measurement sets, these sliders are incorporated into the Display axes rollup.

- Position tracking

  In this roll-up, adjustments can be made to how the coordinates under the pointer (mouse cursor) are displayed.

  By default, position tracking is enabled, so that as your move the pointer over the display, the data value and coordinate are shown in the status line beneath the display. This information will be displayed on any Viewerdisplaypanel on which this Viewerdisplaydata is registered.

  You can turn position tracking off with the switch called Position Tracking? at the bottom of the rollup.

  Position tracking can also be turned on for all Viewerdisplaydatas registered on a single Viewerdisplaypanel by pressing the *t* key, and turned off by pressing *Shift-t*, while the pointer is hovering over the particular Viewerdisplaypanel.

  Other options in this rollup are

  - Position tracking - Absolute or relative

    You may display absolute or relative (to the reference pixel) coordinates. This switch toggles between the two.

  - Position tracking - World or pixel coordinates

    You may display world or pixel coordinates. This switch toggles between the two.

  - Position tracking - Fractional or integral pixel coordinates

    A data lattice is discrete, but each pixel can be considered to extend over the range $i \pm 0.5$. E.g. pixel [20,30] in a 2D images covers the range [19.5,29.5] to [20.5,30.5]. By default pixel coordinates are presented integrally. If you like, this optional allows you to have them presented fractionally.

  - Position tracking - Spectral unit

    If the data contain a spectral axis, this menu will be included. It allows you to select the units for the presentation of the spectral coordinates (e.g. km/s or Hz etc). Conversion to km/s is only possible if the Spectral coordinate contains the rest frequency. In its absence, no conversion to km/s will be offered. You can set the rest frequency with the coordsys.setrestfrequency function.

  - Position tracking - Velocity type

    If the data contain a spectral axis, this menu will be included. If you ask to see the spectral coordinate as a velocity, then this allows you to select the type of velocity definition (e.g. optical, radio etc.).

- Axis labels

  The parameters in this roll-up offer basic control over the axis labels (which by default are not visible). To turn them on, select True for Axis labelling & annotation. The other parameters in this roll-up are pretty much self explanatory, and include text entries for the axis labels (`unset` will give reasonable defaults in most cases), and control of whether tick marks or grid lines are drawn.

- Axis label properties

  In this roll-up, finer control over the axis label properties is provided.

  Here you have control over whether the axis labels are shown as world coordinates or image pixel coordinates (1-relative). You may select whether the labels are relative (to the reference pixel) or absolute.

  For Direction coordinates, you may select the units of the labels if they are relative. For Spectral coordinates you may select the units of the labels, and the velocity type. For other Coordinate types, you don't yet have further control over the units.

  You can also choose the Direction coordinate reference type of the following J2000, B1950, GALACTIC, ECLIPTIC and SUPERGAL. Your coordinate grid will reflect these types.

  You can also choose where to place your axis labels. The default *Auto* is usually sufficient, but other placements are available.

  Colors can be selected for various elements of the labelling, and the character font and size can be chosen. It is generally recommended to first select character font and size, and then pop on over to the Canvas Manager (available under the File menu of the Viewerdisplaypanel) to finely tune the positioning of the displayed image on the Viewerdisplaypanel. In a future release of AIPS++, this geometry adjustment will be automatic.

- Basic Settings This roll-up is rolled down by default! It has the parameters that most dramatically alter the generation of the raster map itself from the image or array data. The elements of this roll-up are:

  - Basic Settings - Aspect ratio

    This option controls the dimensions with which data pixels are drawn to the screen. Fixed lattice means data pixels will be mapped to square pixels on the screen, in as much as the screen pixels themselves are square. Fixed world means that the aspect ratio of

the pixels according to the coordinate system of the image will be honoured. Finally, selecting flexible will allow the raster map to stretch independently in each direction to fill as much of the Viewerdisplaypanel as possible. The default setting is fixed world.

– Basic Settings - Pixel treatment

  This option controls how individual data pixels are drawn to the screen. Selecting center means that pixels at the edge of the display will be drawn only from the centres inwards. That is, down the left hand border of the raster map, only the right-hand half of the pixels will be visible, and along the bottom of the raster map, only the upper half of the pixels will be visible. So for center, the raster map is drawn from the center of the bottom left pixel in the selected data to the center of the top right pixel in the selected data.

  Selecting edge (the default) will change this behaviour such that all data pixels will be drawn fully on the screen. This would be useful for cases where only one of the display axes only has length one pixel.

– Basic Settings - Resampling mode

  This setting controls how the data are resampled to the resolution of the screen. The default, nearest, means nearest neighbour resampling, whereby each pixel on the screen is colored according to the intensity of the nearest corresponding data pixel. The alternative, bilinear, applies a bilinear interpolation to produce smooth-looking images independent of the screen pixel dimensions of the data pixels. While bilinear resampling normally produces a better looking image, it can be significantly slower than nearest neighbour resampling, so be aware of this.

– Basic Settings - Complex mode

  If the data source (image or array) is Complex, then this menu allows you to select whether you wish to see the magnitude, phase, real or imaginary part.

– Basic Settings - Image region

  This region entry is only available for raster maps of images, and allows the selection of a particular part of the image for display. A **region** can be placed in the region entry, and the easiest way to do this is to bring up the Regionmanager using the spanner menu to the right of the entry. Simply create a **region** with the Regionmanager, and send it to this control by pressing Send or Send & dismiss in the Regionmanager.

– Basic Settings - Mask expression

  This entry is only available for raster maps of images, and allows you to specify a Boolean expression mask which is applied On-The-Fly (OTF). Thus you can apply an extra mask to the image in addition to its current persistent default mask.

The syntax is that of the Lattice Expression Language and use of the mask expression entry widget is equivalent to use of the `mask` keyword that is available in many Image `tool` functions and provides an OTF mask capability. See note 223 for details on LEL. Also see the Image tool and the Image Analysis chapter of this document for extra information.

For example, let us say you have an image on disk called *3c273.im* and you have made a raster display of it with the Viewer. You then decide to mask the image as bad where all pixels are negative. You could do this with the Image tool function calcmask which would calculate and store a new mask for you. But perhaps you want to try several different masks before making one persistent. In this case, in the entry widget, you would enter the same expression that you would have provided to calcmask.

```
'3c273.im' > 0
```

and the display would be updated so that all non-negative pixels are good and all negative pixels are bad (masked). Thus, where the expression is True, the pixels are displayed. (Note: use the image filename, rather than the name of any image tool you have created, in these expressions).

When you use the Mask expression in combination with an Image region, the mask expression is applied first, and then the region. This means the shape of the mask expression must match that of the image being displayed.

Also note that if the image you are displaying has an underlying pixel mask (see the Images module), then the final mask that is applied is the union of that mask and the mask expression.

Finally, There is one important difference between using the expression mask entry widget and the `mask` keyword in Image `tool` functions. With the Image tool, you can embed Image `tool` names in your expressions. However, when you use the mask expression entry widget in the adjustment GUI, you must **always** refer to the image disk file name. You cannot use the Image `tool` name.

Now to the *raster* specific options.

- Color Wedge - Color Wedge

  Display a color wedge (bar). **Note: In RGB mode the wedge isn't supported yet.** The wedge will align itself to the image only if you set Basic Settings - Aspect Ratio to *flexible*, or the displaypanel is resized to fir the new aspect. The orientation of the wedge can be changed in the `viewercanvasmanager`.

- Color Wedge - Y axis label

  The axis label to show on the wedge. The default *unset* shows the brightness unit if available.

– Color Wedge - Y axis label color

  Color of the axis label text.

– Color Wedge - Character size

  Size of the axis label text in terms of pgplot character height.

– Color Wedge - Character font

  The (pgplot) font to use for the label.

– Basic settings - Color mode

  This option is specific to raster maps, and controls whether the pixel intensities for this Viewerdisplaydata are mapped to colors selected from a colormap (chosen elsewhere in the adjustment panel), or are instead mapped to one of the following color channels: red, green, blue, hue (color), saturation (color strength) or value (brightness). Use of any color mode requires that the Viewerdisplaydata be registered on a suitable Viewerdisplaypanel. That is, color mode *colormap* rasters must be registered on an *Index* Viewerdisplaypanels, color mode *red, green* or *blue* rasters will only draw when registered on *RGB* Viewerdisplaypanels, and color mode *hue, saturation* and *value* rasters will only draw when registered on *HSV* Viewerdisplaypanels.

– Basic Settings - Data Range

  You can use the type in entry box provided to set the minimum and or maximum data values being mapped to the colormap. Additionally, the limits can be set graphically by using the histogramgui window. This can be opened by selecting 'Histogram' from the spanner menu. For very high dynamic range images, you will need to type in a number much less than the maximum to see the low-level brightness pixels.

– Basic settings - Histogram equalisation

  The default value for this option is False. When enabled for raster maps, some internal optimisation of colors from the colormap (or color channel) is made. The number of colors allocated to a given range of data is proportional to the fraction of pixels whose pixel intensities lie in that range. Thus color cells are allocated where they are needed most.

– Basic settings - Scaling power cycles

  This option offers further control over the mapping of data values to color cells from the colormap. The overall process in selecting a color for a particular pixel is as follows: the data value of the pixel is clipped lie between the data minimum and maximum as specified elsewhere in this Basic settings roll-up. Then, according to this option which is described below, this clipped pixel intensity is mapped to an index between 0 and the number of colors available in the selected colormap. The color which is in the colormap at this

position, which itself is controlled by the current stretch, shift, contrast, brightness and invert settings of the colormap, is then drawn for this particular pixel.

So, the scaling power cycles option controls the mapping of clipped pixel intensities to color look-up values. When this is set to zero, a straight line connects `(domainMin, rangeMin)` and `(domainMax, rangeMax)` on a plot where the domain is the input data values constrained to the range given by the data min and max sliders, and the range runs from 1 to number of colors in the selected colormap. See Figure 1.3 for sample curves.

As the scaling power decreases from zero, the curve deviates from the straight line, rising *above* it, such that an increasing fraction of the available colormap is used for data values close to the data minimum. This curve is calculated using a log function.

As the scaling power increases from zero, the curve deviates from this straight line, falling *below* it, such that an increasing fraction of the available colormap is used for data values closer to the maximum. The curve is calculated using an exponential function. [1]

– Basic settings - Colormap

You can select a range of different colormaps here. Selections here only take effect if the color mode option is set to *colormap*.

– Basic settings - Colorbar

You can choose to have a color bar (wedge) displayed next to your data. At the moment the wedge can be placed to the right-hand side of the image. **Note: In RGB mode the wedge isn't supported yet.** The wedge will align itself to the image only if you set Aspect Ratio to *flexible*.

Following are the *contour* specific options.

– Basic settings - Contour levels

This option is simply where you enter the levels at which you would like contours to be drawn. There are two main ways to enter contour levels:

1. Simply place an integer vector in Contour levels, eg. `1:9`, set Level type to frac, and then Contour scale factor to 0.1. This

[1] The actual functions are computed as follows:

For negative scaling values (say -p), the data is scaled linearly from the range (dataMin – dataMax) to the range $(1 - 10^p)$. Then the program takes the log (base 10) of that value (arriving at a number from 0 to p) and scales that linearly to the number of available colors. Thus the data is treated as if it had p decades of range, with an equal number of colors assigned to each decade.

For positive scaling values, the inverse (exponential) functions are used. If p is the (positive) value chosen, The data value is scaled linearly to lie between 0 and p, and 10 is raised to this power, yielding a value in the range $(1 - 10^p)$. Finally, that value is scaled linearly to the number of available colors.

way, contours are drawn at 10, 20, ..., 90 per cent, where 0 per cent corresponds to the lowest pixel intensity in the data, and 100 per cent to the highest.

2. Set Level type to abs, Contour scale factor to 1, and then enter the absolute levels in the natural units of the data in Contour levels, separated by commas.

– Basic settings - Contour scale factor

  The contour levels given will be multiplied by the factor entered in this option before being drawn.

– Basic settings - Level type

  This option indicates whether the vector of contour levels, specified in the Contour levels box, multiplied by the scale factor, are absolute levels (that is, in the native units of the image), or are instead fractional levels, where 0 is mapped to the lowest pixel intensity in the image, and 1 is mapped to the highest.

– Basic settings - Line width

  This option controls the thickness of the contour lines. While only integer values will show any effect on-screen, PostScript output will be affected by the fractional part of the line width.

– Basic settings - Dash negative contours?

  Set this option to True if you would like contours of negative values in your data to be dashed rather than solid lines.

– Basic settings - Dash positive contours?

  Set this option to True if you would like contours of positive values in your data to be dashed rather than solid lines.

– Basic settings - Line color

  Select a color for the contours.

Now to the *vector* specific options

Vector map adjustments are based on those Basic settings. The following options are not available: resampling mode, complex mode and data minimum & maximum. In their place, the following options are added in the Basic settings section:

– Basic settings - Phase Type

  If the data source is Complex ($z = R + iI$) this option is available. It is a menu with two values: values normal and polarimetric mean the phase (position angle) is computed from $atan2(I, R)$ and $atan2(I, R)/2$, respectively. The latter is appropriate when the Complex data holds, say, the Complex linear polarization $z = Q + iU$ which has a two-fold symmetry. The default is polarimetric.

– Basic settings - Constant amplitude ?

  If the data source is Complex, this option allows you to discard the amplitude information and show all vectors with equal length.

– Basic settings - Debias amplitude ?

  If the data source is Complex ($z = R + iI$) this option is available. The default is no debiasing. The amplitude is computed from $\sqrt{(R^2 + I^2)}$. Now if the real and imaginary parts are drawn from independent gaussian variables (as is the case for the Complex polarization $z = Q + iU$), then even in the absence of signal, the mean amplitide is positive. This is noise bias. This option attempts to remove it to first order via $amp = \sqrt{(R^2 + I^2 - variance)}$.

  To do this, you need the variance of the noise (assumed the same for real and imaginary). See the next option.

– Basic settings - Variance for debiasing

  If the data source is Complex, this option is available. It is only used if the debias option is selected (see above). The debiasing is accomplished via $amp = \sqrt{(R^2 + I^2 - variance)}$. By default, the variance is worked out for you from the image (the real and imaginary 10-sigma clipped variances are worked out separately and averaged). However, even though a clipping algorithm is used, the variance may not be accurate. If you know the variance of the noise accurately, enter it here.

– Basic settings - Amplitude scale factor

  By default, the amplitudes of the vectors are scaled so that the longest vector is of length 5% the length of the smallest display window dimension. This is an extra scale factor that you can apply to all the vectors to make them longer or shorter.

– Basic settings - X and Y increment

  By default, vectors are displayed for every third pixel in x and y. You can change that with these options.

– Basic settings - Extra rotation

  You can add an extra rotation (degrees) to the position angles here (e.g. to show a magnetic field position angle from electric field vectors).

– Basic settings - Show arrow heads ?

  This option allows you to put arrow heads on the vectors. However, if the Phase type is Polarimetric arrow heads are not displayed (two-fold symmetry makes them meaningless).

– Basic settings - Arrow head shape

  This option allows you to change the arrow head shape from a triangular head (value 0) to an open head. In between, you get 'barbed' shaped heads.

- Basic settings - Line width

  This option controls the thickness of the vector lines. While only integer values will show any effect on-screen, PostScript output will be affected by the fractional part of the line width.

- Basic settings - Line color

  Select a color for the vectors.

Now to the *marker* specific options

Marker map adjustments are based on those Basic settings. The following options are not available: resampling mode, complex mode and data minimum & maximum. In their place, the following options are added in the Basic settings section:

- Basic settings - Marker Shape

  By default, the marker shape is a square. At present, this is the only marker shape available.

- Basic settings - Scale factor

  By default, the amplitudes of the shapes are scaled so that the width of the shape 5% the length of the smallest display window dimension. This is an extra scale factor that you can apply to all the shapes to make them bigger or smaller.

- Basic settings - X and Y increment

  By default, markers are displayed for every third pixel in x and y. You can change that with these options.

- Basic settings - Line width

  This option controls the thickness of the marker outline. While only integer values will show any effect on-screen, PostScript output will be affected by the fractional part of the line width.

- Basic settings - Line color

  Select a color for the marker.

Skycatalog Viewerdisplaydatas are different from the others since they don't carry a coordinate system. Therefore the option record looks quite different.

- Advanced At the moment this roll-up only contains cache size control.

  - Advanced - Cache size

    This option should rarely need to be adjusted, especially for catalog overlay Viewerdisplaydatas. The value of this option specifies how many different views of the catalog overlay are stored in a cache, and can therefore be redrawn quickly on demand. A different view is generated when one or more options are altered.

- Label properties

  This roll-up provides options for selecting which column is used to label the sources, and for setting the properties of the labels themselves.

  - Label properties - Column listing names

    This option must be changed from <none> if you want labels to appear next to the source markers. Select the column which contains the names; the options offered are *only* the columns which contain strings.

  - Label properties - Line width

    Select the line width for the text labels; this effectively controls how bold the labels appear.

  - Label properties - Character font

    This option offers the choice of font for the labels.

  - Label properties - Character size

    Adjust the size of the text labels.

  - Label properties - Character color

    This option enables the user to select a color for the labels, which can of course be different to the color of the markers themselves.

  - Label properties - Label angle

    The labels can be drawn at any rotation, so this option allows the setting of that angle in integer degrees.

  - Label properties - Label X offset (char units) & Label properties - Label Y offset (char units)

    Labels can be offset vertically and horizontally from the markers. Careful use of these options and the label angle can be used to avoid most conflicts between labels and markers.

- Marker properties

  This roll-up provides options for controlling the appearance of the markers used to mark the positions of the sources.

  - Marker properties - Marker type

    This option is a poor but functional way to choose a marker for your sources from the catalog.

  - Marker properties - Marker size

    This option controls the size of the markers.

  - Marker properties - Marker color

    Markers can be drawn in any of the available colors selected from this menu.

- Basic settings

  In this roll-up, some general properties of the catalog overlay can be set.

  - Basic settings - "Where" query The catalog in use can be "pruned" by using this option. A valid table query language (TaQL - see AIPS++ Note 199) **WHERE** clause must be provided. The default value is `unset` so that all sources in the catalog are plotted. For example, you want to plot all sources in the Longitude range 0 ¡ long ¡ 1 degrees and the longitude column in the table is named "Long", the line to enter would be:

    ```
    Long > 0 && Long < 1
    ```

    Two nice features of TaQL are:

    * **Pattern matching** say all Galaxies which start with NGC. Specify the column name (here: "Name") and the pattern:
      ```
      Name == pattern('NGC*')
      ```
    * **regular expressions** for the same example:
      ```
      Name == regex('^NGC.+')
      ```

Raster measurement set Viewerdisplaydatas have some inputs similar to those for raster image display, with some additional interface for selecting and editing (flagging) the MS data.

- MS and Visibility selection

  - Visibility Type
  - Visibility Component
  - Moving Average Size

  This roll-up provides choice boxes for Visibility Type (Observed, Corrected, Model, Residual) and Component (Amplitude, Phase, Real, or Imaginary).

  Changes to Visibility Type or Component (changing from Phase to Amplitude, for example) require the data to be retrieved again from the disk into memory, which can be a lengthy process. When a large MS is first selected for viewing, the user must trigger this retrieval manually by pressing the 'Apply' button, after selecting the data to be viewed (*Field IDs* and *Spectral Windows*, below).

  Changing visibility type between 'Observed' and 'Corrected' can also be used to assure that data and flags are reloaded from disk. You should do this if you're using another flagging tool simultaneously (such as autoflag or msplot), so that the viewer sees the other tool's new edits and doesn't overwrite them with obsolete flags. The 'Apply' button alone won't reload unless something within the viewer itself requires it; in the future, a button will be provided to reload flags from the disk unconditionally.

You can also choose to view the difference from a running mean or the local RMS deviation of either Phase or Amplitude (this will not require a disk reload). There is a slider for choosing the nominal number of time slots in the 'local neighborhood' for these displays.

(Note: **Insufficient Data** is shown in the tracking area during these displays when there is no other unflagged data in the local neighborhood to compare to the point in question. The moving time windows will not extend across changes in either field ID or scan number, so you may see this message if your scan numbers change with every time stamp. An option will be added soon to ignore scan boundaries).

- Field IDs
- Spectral Windows

You can retrieve and edit a selected portion of the MS data by entering the desired Spectral Window and Field ID numbers into these boxes. Especially with large MSs, it makes sense to edit only one or two fields at a time, and to select spectral windows which all have the same number of channels. Doing this will also greatly reduce data retrieval times and memory requirements.

You can separate the ID numbers with spaces or commas; you do not need to enter enclosing brackets. Changes to either entry box will normally cause the selected MS data to be reloaded immediately from disk. If you wish, you can turn off 'autoapply', in order to change *both* spectral window and field selections, without reloading data after each change. In that case, press 'Apply' after entering all your selections.

If you select, say, spectral windows 7, 8, 23, and 24, the animator, slice position sliders, and axis labelling will show these as 1, 2, 3, and 4 (the 'slice positions' or 'pixel coordinates' of the chosen spectral windows). Looking at the position tracking display is the best way to avoid confusion in such cases. It will show something like: **Sp Win 23 (s 3)** when you are viewing spectral window 23, the third selected spectral window.

Changes to MS selections will not be allowed until you have saved (or discarded) any previous edits you have made (see *Flagging Options – Save Edits*, below).

Initially, all fields and spectral windows are selected. To revert to this 'unselected' state, choose 'Original' under the wrench icons next to the entry boxes.

- Display Axes

This rollup is very similar to that for images: it allows the user to choose which axes (from Time, Baseline, Polarization, Channel, and Spectral Window) are are on the display and the animator. There are also sliders here for choosing positions on the remaining axes.

Changing the choice of axis on one control will automatically swap axes, maintaining different axes on each control. Changing axes or slider/animator positions does not normally require pressing 'Apply'–the new slice is shown immediately. However, the display may be partially or completely grey in areas where the required data is not currently in memory, either because no data has been loaded yet, or because not all the selected data will fit into the allowed memory (see *Max. Visibility Memory*, below). When the cursor is over such an area, the following message shows in the position tracking area:

press 'Apply' on Adjust panel to load data

Pressing 'Apply' will reload the memory buffer so that it includes the slice you are trying to view.

The message **No Data** has a different meaning; in that case, there simply *is* no data in the selected MS at the indicated position.

For large measurement sets, loading visibility data into memory is the most time-consuming step. At present it is not possible to show the usual progress meter window, but similar feedback is provided in the console window. Again, careful selection of the data to be viewed can greatly speed up retrieval.

- Flagging Options

  These options allow you to edit (flag or unflag) MS data. The crosshair and rectangle region mouse controls are used on the display to pick the data region to edit; the options below determine how the edits will be applied.

  - Flagging Options - Show Flagged Regions...

    You have the option to display flagged regions in the background color (as in msplot or TVFLG) or to highlight them with color. In the former case, flagged regions look just like regions of no data. With the color option, flags are shown in shades of blue: darker blue for flags already saved to disk, lighter blue for new flags not yet saved; regions with no data will be shown in black.

  - Flagging Options - Flag or Unflag

    This setting determines whether selected regions will be flagged or unflagged. In contrast to msplot, this does *not* affect previous edits; it only determines the effect which later edits will have. Both flagging and unflagging edits can be accumulated and then saved in one pass through the MS.

  - Flagging Options - Flag/Unflag All...

    These flagging extent checkboxes have a similar operation to those in msplot. They allow you to extend your edit over any of the five data axes. For example, to flag *all* the data in a given time range, you would check all the axes *except* Time, and then select the

desired time range with the region mouse control. Such edits will extend along the corresponding axes over the entire selected MS (whether loaded into memory or not) and optionally over unselected portions of the MS as well (*Use Entire MS*, below). Use care in selecting edit extents to assure that you're editing all the data you wish to edit.

– Flagging Options - Flag/Unflag Entire Antenna?

  This control can be used to extend subsequent edits to all baselines which include the desired antenna[s]. For example, if you set this item to 'Yes' and then click the crosshair on a visibility point with baseline 3-19, the edit would extend over baselines 1-3, 2-3, 3-3, 3-4, 3-5, 3-6, ... 3-¡nAntennas¿. Note that the second antenna of the selection (19) is irrelevant here–you can click anywhere on the display where the first antenna number is 3, to select all baselines which include antenna 3.

  This item controls the edit extent only along the baseline axis. If you wish to flag *all* the data for a given antenna, you must still check the boxes to flag all Times, Channels, Polarizations and Spectral Windows. There would be no point, however, in activating *both* this item and the 'Flag All Baselines' checkbox. You can flag an antenna in a limited range of times, etc., by using the appropriate checkboxes and selecting a rectangular region of visibilities with the mouse.

– Flagging Options - Undo Last Edit

– Flagging Options - Undo All Edits

  The 'Undo' buttons do the expected thing: completely undo the effect of the last edit (or all unsaved edits). Please note, however, that only unsaved edits can be undone. In contrast to msplot, there is no ability to revert to the flagging state at the start of the session, once flags have been saved to disk.

– Flagging Options - Use Entire MS When Saving Edits?

  "Yes" means that saving the edits will flag/unflag over the entire MS, *including* fields (and possibly spectral windows) which are not currently selected for viewing. Specifically, data within time range[s] you swept out with the mouse (even for unselected fields) will be edited.

  In addition, if "Flag/Unflag All..." boxes were checked, such edits will extend throughout the MS. Note that only unselected *times* (fields) can be edited *without* checking extent boxes for the edits as well. Unselected spectral windows, e.g., will *not* be edited unless the edit also has "Flag/Unflag All Spectral Windows" checked.

  Warning: Beware of checking "All Spectral Windows" unless you have also checked "All Channels" or turned "Entire MS" off; channel edits appropriate to the selected spectral windows may not

be appropriate to unselected ones. Set "Use Entire MS" to"No" if your edits need to apply only to the portion of the MS you have selected for viewing. *Edits can often be saved significantly faster this way as well.*

Also note that checkboxes apply to individual edits, and must be checked before making the edit with the mouse. "Use Entire MS", on the other hand, applies to all the edits saved at one time, and must be set as desired before pressing "Save Edits".

- Flagging Options - Save Edits

  The measurement set Viewerdisplaydata works like a text editor in that you see all of your edits immediately, but nothing is committed to disk until you press 'Save Edits'. Feel free to experiment with all the other controls; nothing but 'Save Edits' will alter your MS on disk. As mentioned previously, however, there is no way to undo your edits once they are saved, except by manually entering the reverse edits.

  Also, *you must save* (or discard) *your edits before changing the MS selections.* If edits are pending, the selection change will not be allowed, and a warning will appear on the console.

  If you close the tool, *unsaved edits are simply discarded*, without prior warning. It's important, therefore, to remember to save them yourself. You can distinguish unsaved flags (when using the 'Flags In Color' option), because they are in a lighter shade of blue.

  The program must make a pass through the MS on disk to save the edits. This can take a little time; progress is shown in the console window.

- Axis Drawing and Labels
- Position tracking
- Color Wedge
- Basic settings

  These four rollups contain entries similar to those for image Viewerdisplaydatas. Some of those entries were unnecessary and were omitted for simplicity here; others will be added in the future.

  Together with the brightness/contrast and colormap adjustment mouse controls on the Viewerdisplaypanel, the following Basic settings are especially important for adjusting the color display of your data:

  - Basic settings - Data minimum/maximum

    At present, these are only sliders, without the histogram capabilities of the image Viewerdisplaydatas. Lowering the data maximum will help brighten weaker data values.

- Basic settings - Scaling power cycles

  This has exactly the same usage as for image Viewerdisplaydatas. Again, lowering this value often helps make weaker data visible. If you want to view several fields with very different amplitudes simultaneously, this is probably the first adjustment you should make.

- Basic settings - Colormap

  Greyscale or Hot Metal colormaps are generally good choices for MS data.

Adding a Color Wedge to the right of your display helps to show the effect of your other color scale adjustments. Its displayed scale and units are not always up-to-date at present (a known problem, to be fixed soon). In the meantime, moving the slider for Data minimum or maximum will cause the correct scale to be displayed.

- Advanced

  These settings can help optimize your memory usage, especially for large MSs, if you feel comfortable tweaking such things. A rule of thumb is that they can be increased until response becomes sluggish, when they should be backed down again.

  You can run the unix 'top' program and hit 'M' in it (to sort by memory usage) in order to examine the effects of these settings. Look at the amount of RSS (main memory) and SWAP used by the X server and 'glishtk' processes. If that sounds familiar and easy, then fiddling with these settings is for you. Otherwise, the default settings should provide reasonable performance in most cases.

  - Advanced - Cache size

    As with the Skycatalog, the value of this option specifies the maximum number of different views of the data to save so that they can be redrawn quickly. If you run an animation or scroll around zoomed data, you will notice that the data displays noticably faster the second time through because of this feature. Often, setting this value to the number of animation frames is ideal Note, however, that on multi-panel displays, each panel counts as one cached image.

    Large images naturally take more room than small ones. The memory used for these images will show up in the X server process. If you need more Visibility Memory (below) for a really large ms, it is usually better to forgo caching a large number of views.

  - Advanced - Max. Visibility Memory

    This option specifies how many megabytes of memory may be used to store visibility data from the measurement set internally. Even if you do not adjust this entry, it is useful to look at it to see how

many megabytes are required to store your entire (selected) MS in memory. If the slider setting is above this, the whole selected MS will fit into the memory buffer. Otherwise, some data planes will be 'greyed out', and the selected data will have to be viewed one buffer at a time, which is somewhat less convenient. The largest process called 'glishtk' contains this buffer memory; it actually contains the entire display library, but the memory buffer can dominate its size.

1.3.6 viewerdisplaypanel - Tool

Package display
Module viewer

Tool for providing drawing windows (canvases)
include "viewerdisplaypanel.g"

Constructors

viewerdisplaypanel	Create a Viewerdisplaypanel `tool`

Functions

addgui	Add various GUI components to this Viewerdisplaypanel `tool`
adjust	Display adjustment panel for Viewerdisplaydata `tools`
animator	Return the Vieweranimator `tool` used by this `tool`
annotationdd	Return the agent of the annotation layer
annotator	Return the viewerannotations created by this Viewerdisplaypanel `tool` on constructi
autoregister	Set or query the auto-register state of this Viewerdisplaypanel `tool`
canvasmanager	Return the Viewercanvasmanager `tool` used by this `tool`
canvasprintmanager	Return the Viewercanvasprintmanager `tool` used by this Viewerdisplaypanel `tool`
disablecontrols	Temporarily disable the controls on this Viewerdisplaypanel `tool`
done	Close down (destroy) this Viewerdisplaypanel `tool`
dying	Is this Viewerdisplaypanel `tool` in the process of destroying itself?
enablecontrols	Re-enable the controls on this Viewerdisplaypanel `tool`
getoptions	Return a record describing some modifiable state of this `tool`
hold	Prevent refreshes occuring on this Viewerdisplaypanel `tool`
holdsdata	Returns whether this Viewerdisplaypanel `tool` can hold Viewerdisplaydata `tools`
isregistered	Query whether a Viewerdisplaydata `tool` is registered on this Viewerdisplaypanel `to`
leftframe	Return the left frame in the Viewerdisplaypanel `tool` window
lowerframe	Return the lower frame in the Viewerdisplaypanel `tool` window
newcanvasmanagergui	Create and (probably) show a new graphical user interface for this `tool`'s Viewercan
newcanvasprintmanagergui	Create and (probably) show a new graphical user interface for this `tool`'s Viewercan
newcmap	Query whether this Viewerdisplaypanel `tool` is using a new colormap
register	Register a Viewerdisplaydata `tool` for drawing

registercolormap	Explicitly register a colormap on the Viewerdisplaypanel `tool`
registerednames	Return a vector of the names of registered Viewerdisplaydata `tools`
registrationflags	Return a record describing which Viewerdisplaydata `tools` are registered
release	Allow refreshes to occur on this Viewerdisplaypanel `tool`
rightframe	Return the right frame in the Viewerdisplaypanel `tool` window
setoptions	Set the modifiable state of this `tool`
setzoom	Set a specific zoom
status	Return a record describing various attributes of the Viewerdisplaypanel
testpattern	Turn a test pattern on or off on the Viewerdisplaypanel `tool`
type	Return the type of this `tool`
unregister	Unregister a Viewerdisplaydata `tool` from this Viewerdisplaypanel `tool`
unregisterall	Unregister all Viewerdisplaydata `tools` from this Viewerdisplaypanel `too`
unregistercolormap	Explicitly unregister a colormap on the Viewerdisplaypanel `tool`
unzoom	Unzoom the current view
upperframe	Return the upper frame in the Viewerdisplaypanel `tool` window
viewer	Return the Viewer `tool` which manages this `tool`
writexpm	Write an X11 Pixmap of the Viewerdisplaypanel `tool` to disk
zoom	Zoom by a given factor

General description

Description

This `tool` is a fundamental component of the Viewer `tool`. It provides a drawing area (or "canvas") on which Viewerdisplaydata `tools` can draw themselves. Multiple Viewerdisplaypanel `tools` can be placed on the screen, and each can have multiple Viewerdisplaydata `tools` registered to draw themselves. In this way, for example, multiple views of the same data can be simultaneously displayed, and contour Viewerdisplaydata `tools` can be drawn "on top" of raster Viewerdisplaydata `tools`, for example. Exact copies of Viewerdisplaypanel `tools` can be written to X11 Pixmap files, and high-quality PostScript representations of Viewerdisplaypanel `tools` can be saved and printed.
Viewerdisplaypanel `tools` can be adorned with various graphical user interface components, including a "controlbox" which allows the user to select controls like zooming and colormap fiddling, and a "tapedeck" which enables the user to manually or automatically move through data having more than two (non-redundant) axes.
Three points related to the registration of Viewerdisplaydata `tools` on Viewerdisplaypanel `tools` are worth making here:

1. *The order of registration of Viewerdisplaydata* `tools` *on a specific Viewerdisplaypanel* `tool` *is important.* The maximum area of sky, or position-velocity space, or of whatever coordinate system is being used, is determined solely by the first registered Viewerdisplaydata `tool`. All

other Viewerdisplaydata `tools` registered for display on a particular Viewerdisplaypanel `tool` simply draw those parts of their data which ought to be visible on the Viewerdisplaypanel `tool`. A consequence of this is that the zoom setting of the Viewerdisplaypanel `tool` must be reset to the full view whenever the first registered Viewerdisplaydata `tool` is removed, replaced, or re-read. Independent of the order in which Viewerdisplaydata `tools` are registered on a given Viewerdisplaypanel `tool`, the raster-type Viewerdisplaydata `tools` are always drawn first, followed by contours, and lastly axis labels.

2. *Axis labels are drawn by Viewerdisplaydata* `tools`*, and only by the first registered Viewerdisplaydata* `tool`*.* In the Viewer `tool`, axis labelling information belongs to the Viewerdisplaydata `tools` rather than to the Viewerdisplaypanel `tools`, which is commonly seen in other applications. This is because the only entity which knows *exactly* how to draw the axis labels for a particular piece of data is the Viewerdisplaydata `tool` which draws the data itself. Control over the appearance of axis labels is therefore available via the individual Viewerdisplaydata `tools`.

3. *Registering a specific Viewerdisplaydata* `tool` *on more than one Viewerdisplaypanel* `tool` *can reduce display efficiency.* The drawings made of data by a Viewerdisplaydata `tool` are cached, so that they only need to be recalculated under various circumstances, such as the settings for the Viewerdisplaydata `tool` changing, or the size of the Viewerdisplaypanel `tool` on which they are drawing changing. The caching will in general fail to operate efficiently if a Viewerdisplaydata `tool` is registered on more than Viewerdisplaypanel `tool`. This is best avoided by making another Viewerdisplaydata `tool` from the same dataset, and registering this second `tool` on the second Viewerdisplaypanel `tool`.

Additional to the `tool functions` described for the Viewerdisplaypanel `tool`, this `tool` also possesses the `tool functions` described in the Viewerstandardgui interface documentation. Note that some functions may not be documented and therefore shouldn't be used unless you really know what you are doing.

Graphical User Interface

Description

The standard graphical user interface provided by the Viewerdisplaypanel `tool` is shown in Figure 1.5. Surrounding the actual display area itself are five GUI components: the menu bar at the top, the button bar at the bottom, the tracking bar[s] just above the buttons, the controlbox on the left, and the (currently disabled) "tapedeck" control on the right. Applications

programmers are able to control which of these components appear on the Viewerdisplaypanel `tools` that they create, and hence throughout AIPS++, Viewerdisplaypanel `tools` can appear slightly different from the example given here.
Note: All AIPS++ GUIs have a so-called popup help. Just rest your mouse cursor on top of a GUI element (e.g. a button) and a little frame will appear outlining the funtionality of this element. If you then press and hold down the right mouse button you will get a detailed description. The information given there should be sufficient to let you operate that GUI element.

The menu bar

In Figure 1.5, a typical Viewerdisplaypanel `tool` is shown in all its glory. The menu bar along the top has just three menus:

- File, offering selections to create and display new Viewerdisplaypanel `tools`, access to various managers associated with the Viewer `tool`, and selections to finish the Viewerdisplaypanel `tool` itself, and to finish use of the entire Viewer `tool`.

  The File menu is where additional Viewerdisplaypanel `tools` can be created for use. Multiple Viewerdisplaypanels can be on-screen, and to the best of their ability they will share the available color resources. Three classes of Viewerdisplaypanel can be made: *Index* panels, on which normal raster maps can be displayed, with pixel intensities mapped to colors from a colormap; and *RGB* (red-green-blue) and *HSV* (hue-saturation-value) panels, on which multi-component raster maps can be put together. Viewerdisplaydatas which are set up to use a colormap (eg. 'Rainbow', 'Greyscale', etc.) will only draw when registered on an *Index* panel. Similarly, Viewerdisplaydatas which are set up to draw to the red, green or blue color channel will only draw themselves on RGB panels. And likewise, Viewerdisplaydatas which are set up to draw to the hue, saturation or value color channel will only draw when registered on HSV panels.

  For the *RGB* and *HSV* panel types, up to three raster maps can be displayed concurrently, overlaid on each other, with each mapped to a different color component. For example, an optical image could be displayed using the green component, and an X-ray image "overlaid" using the red component. In locations of strong X-ray emission and weak optical emission, the map will be predominantly red; weak X-ray emission coincident with strong optical emission will be predominantly green, and in between will be various levels of yellow.

  Two sub-classes of Viewerdisplaypanels can be made: *non-flashing* and *most colors* panels. The latter provides the most colors available, but with the cost of ugly "color flashing" behaviour as the mouse cursor is moved in to and out of the Viewerdisplaypanel window. The former provides generally less colors, but avoids this color flashing, and is

generally more appropriate when analysis involving multiple windows is underway.

The menu also gives access to Viewerslicepanel `tools` in all color modes. Viewerdisplaydata `tool` with at least three axes can be displayed on these in slice mode. See the documentation for Viewerslicepanel `tool` for more.

- DisplayData, which is where various Viewerdisplaydata `tool` can be registered (or unregistered) for display on the Viewerdisplaypanel `tool`. Viewerdisplaydata `tool` are registered for display by ensuring that their checkbox is lit in this menu. When more than one Viewerdisplaydata `tool` is available for registration, additional convenience selections are placed at the bottom of the menu, enabling the user to register or unregister all the Viewerdisplaydata `tool` in the list.

- Tools contains a submenu ImageAnalysis ImageAnalysis tools all listen to regions and postions emitted by the region/positioning tool. The Tools menu also contains a Annotations... menu item, which opens the the Annotations GUI.

  You can interactively make a region (boxes and polygons) from the display with the cursor. First you select the appropriate button (or icon; box or polygon) from the viewer control buttons arranged vertically along the left-hand side of the display. Whatever mouse button you used to select the icon is now active for creating the region. Either drag out a box, or select vertices of a polygon.

  You create the region by double clicking inside of it. When you do this, the Viewerdisplaypanel `tool` will emit an event called `region`. See below for a description of that event.

  - ImageAnalysis This menu brings up a window with roll-ups for several image analysis operations. Each roll-up has a tab widget for each of the registered Viewerdisplaydatas. You can select the one you want to analyze by pressing the button showing the image name.

    * Statistics This roll-up shows image statistics on the selected image. It listens to event created by the region tools (of the control box. Whenever you emit such a region the fields in this GUI will be updated.
      When you generate an interactive region the statistics are computed only for the current displayed plane of the image. You can extend this spatial region over a range of animation-axis pixels via the Extend check box and by entering a pixel range in the adjacent entry box. For example, entering 10 20 would apply the spatial region to planes 10 through 20 along the animation axis.

Statistics can also be computed for the full data Full or the current plane Plane as selected with the animator tapedeck. Check the Robust button to also retrieve robust statistics. You can also bring up a pgplotter tool to show a statistic histogram by clicking the plot button or alternatively Auto-plot to have that plot updated automatically. The next row of buttons allow you to copy the values shown to the clip board (retrieve with `x := dcb.paste()`). Auto-copy automatically updates the clip board.

* Positions The Position rollup consists of a plot button (only if the image has more than 2 dimensions), a copy button, an autocopy switch and position fields to identify the current position of the cursor.
  The position rollup listen's to position events generated by it's parent Viewerdisplaypanel `tool` (via the crosshair tool) as well as motion events generated by the Viewerdisplaypanel `tool` from the movement of the mouse over the display area (when position tracking is turned on). When one of these events are received by the position rollup, the current position details are updated.
  You can plot the profile (at the cursor position and along the movie axis) with the button labelled `Plot`.
  When you submit a `region event` (see below), the plot will be updated with a new profile. This profile respresents the statistics in the submitted region along the movie axis of the image. The default statistic is `mean` and can be changed in via the `Adjust` button near the bottom of the profile window.
  You can copy the position details to the default Clipboard `tool` (**dcb**) with the `Copy` button. This copies a record to the clipboard with the format below. If you prefer, you can autocopy the positions to the Clipboard by checking the `Autocopy` button. Then, every time you generate a position, a record is automatically copied the Clipboard.
  Format of copied record:
  rec.ddname
  rec.position .pixel .world .numeric .string .intensity .value .mask
  rec.region .pixelblc .pixeltrc .stat
  rec.profile .cs .cs .x .pixel .abs .world .abs .y .data .mask .unit
  The profile can be customized via the `Adjust` button near the bottom of the profile window. Features such as autoscaling, rest frequency labelling and region profiles are available here. You can zoom in the profile window by using the left mouse button using the same method that is used to zoom in the viewer panel. Similarly, an `Unzoom` button is also available.

* Regions

Every time you generate a region, it is also inserted into the region entry widget (a check box flashes to indicate something went in) at the top of the regions rollup (roll it down to see the entry widget !). This enables you to copy it to the default Clipboard `tool` (**dcb**) and also paste into it from the default Clipboard `tool`. When a region is captured, a little indicator light next to the entry widget flashes to indicate something went in. There is also a `Show` button which prints the bounding box of the region to the logger and shows the regions already generated as a white outline. This button turns into `Hide` once it is pressed to disable the region drawing.
It is also possible to accumulate regions, and then make a compound region from all of them. This is done via the `Regions` rollup; roll it down and look at the row of buttons at the bottom. In this mode, if you have pressed the `Show` button, the regions are shown individually as they are generated.
The second button from the left controls the accumulation. If it is labelled `Start`, when you press it (and its label will then change to `Finish`), regions are internally accumulated (no `region` event is emitted and the region is not inserted in the region entry widget). When you have accumulated all the regions you want, you may press `Finish`. When accumulation mode is active, a check box to the right is lit and the number of regions accumulated displayed. Now a compound region is made from all the regions you accumulated according to the select you make from the left-most menu; you can make a union or intersection. When this region is created, the `region` event is emitted.

* Slices Every time you use the polyline tool, a `polyline` event is generated containing the points of the polyline. These points are used here to
  - Generate an interpolated 1-D slice along the polyline and optionally plot it (use the `Plot` and `Autoplot` buttons).
  - For DirectionCoordinates, the distance and position angle between the polyline vertices is computed. If the polyline contains more than two points the sum over all distances and the sum over all angles is shown.

  Use the `Copy` and `Autocopy` buttons to transfer the slice, distance and position angle information to the clip board.
* Summaries This part just shows the same Image.summary would show, wrapped up in a nice gui text area.

- Help, which provides selections with which assistance can be obtained on the Viewer `tool` itself, or more widely within AIPS++.

The button bar Along the bottom of the Viewerdisplaypanel `tool` are a few

buttons, which are available for commonly executed tasks:

- The Adjust... button is used to make adjustments to the way the various Viewerdisplaydata `tool` draw themselves. Pressing Adjust just once will create adjustment windows for all Viewerdisplaydata `tool` that are presently registered on the Viewerdisplaypanel `tool`. Here is a detailed description of the elements of the Adjust GUI.

- The Unzoom button is a short-cut to fully unzooming the display—when pressed, the Viewerdisplaypanel `tool` will re-draw itself displaying as much of the *first-registered* Viewerdisplaydata `tool` as is possible.

- The Clear button is a short-cut to unregistering all Viewerdisplaydata `tool` which are presently registered on this Viewerdisplaypanel `tool`. When pressed, the Viewerdisplaypanel `tool` will be cleared.

- The Print... button is a short-cut to selecting Print... from the File menu, and will bring up the print window for this Viewerdisplaypanel `tool`, which naturally provides printing operations. For more go to Viewercanvasprintmanager `tool`.

- Finally, the Done button will remove this Viewerdisplaypanel `tool` from the viewer.

The tracking bar[s]

Separate tracking bars are generated for each Viewerdisplaydata `tool` registered on the Viewerdisplaypanel `tool`. They continually report the value and position of the point of the display area under the mouse cursor.
Pressing `space bar` in the display area will "freeze" the tracking displays or restore them again to continual tracking. Pressing the `L` key will send the displayed values to the logger. (An even more detailed record may also be sent to the clipboard, by using the Positions rollup under the Tools/ImageAnalysis menu item).
Note that if you have "click-to-focus" set in your windowing system, you may need to click on the displaypanel's window before it will recognize the key presses above. This can be confusing, since the tracking bar[s] will respond to mouse movement in any case.

The control box

The control box (on the left hand side of most Viewerdisplaypanel `tools`) provides a more convenient way to access the various controls available on Viewerdisplaypanel `tools`. Clicking on a button will assign that control to the mouse button which was used to click, over-riding any previous assignment to that mouse button.
For most tools the following actions apply:

- "Esc" key cancels the tool.

- "Double click" of the button assigned to the tool executes it (eg. with regions it sends out an event).
- "Handles" (little boxes on the edges) can be used to resize the tool.

From the top button downwards, the controls are:

- **Zooming:** The top-most button, depicting a magnifying glass with a plus sign inside, is the zoom control button. Zooming is accomplished by pressing down the selected mouse button at the start point, dragging the mouse away from that point, and releasing the selected mouse button when the zoom box encloses the desired zoom area. Once the button is released, the zoom rectangle can be moved by clicking inside it with the selected mouse button and dragging it around, and resized by selecting one of the handles at the corners and dragging.

  To zoom in, simply double click with the selected button inside the rectangle, and when the Viewerdisplaypanel `tool` is re-drawn, the area enclosed by the rectangle will be expanded to fill the Viewerdisplaypanel `tool`, in as much as other constraints allow this. Double clicking outside the rectangle, will result in a zoom out, whereby the previous full view will be scaled down to fit within the rectangle, in as much as other constraints allow this.

- **Panning:** The next button down, whose icon is the outline of a hand, is the panning control button. Panning is accomplished by pressing down the selected mouse button at the point you wish to move, dragging the mouse to the position where you want the first point moved to, and releasing the selected mouse button.

  Note: The arrow keys, Page Up, Page Down, Home and End keys, and scroll wheel (if any) can also be used to scroll through your data once you have zoomed in. For these to work, the mouse must be over the display panel drawing area, but no mouse tool need be active. All but the left and right arrows apply only to the vertical direction. However, you can use the Crtl key in conjunction with the arrows to move one page at a time in any direction.

- **Stretch-shift colormap fiddling:**
- **Brightness-contrast colormap fiddling:**
- **Positioning:** The fifth control is a crosshair icon, which enables the user to place a crosshair marker on the Viewerdisplaypanel to indicate a position. Depending on the context, the emitted positions may be used, for example, to display image spectral profiles or flag measurement set data. After selecting a mouse button for the crosshair, click on the Viewerdisplaypanel to place it, and drag to move it around. Double-click is not needed with this control; the position is emitted whenever the crosshair's assigned mouse button is pressed or the crosshair is moved. **Note: position selection is quite different from *position tracking*.**

- **Rectangle and Polygon region drawing:** Rectangular and polygonal regions can easily be constructed. The rectangle region is generated exactly the same way as the zoom rectangle, and is emitted by double clicking within the rectangle. Polygon regions can be constructed by progressively clicking the selected mouse button at the desired location of each vertex, and clicking in the same location twice to complete the polygon. Once constructed, it can be moved by dragging inside the polygon, and reshaped by dragging the various handles at the vertices. Double click within the polygon to emit the polygonal region. Compound image regions can be built up using these controls. The rectangular control can also be used to specify areas to be flagged in measurement sets.

- **Polyline drawing:** A polyline can be constructed with this tool. This tool is almost identical to the polygon region tool. Create the points by clicking at the positions wanted and the double-click to finish the line. this generates a polyline event.

The animation tapedeck

In its 'Normal' mode, the tapedeck-style controls down the right-hand side of most Display Panels provide basic movement between image planes, along a selected third dimension of the displaydatas. This set of buttons will only be enabled when the first-registered displaydata reports that it has more than one plane along the 'Z axis' specified in its 'Adjust' panel. In the most common case the animator controls the frequency channel being viewed. From the top down, the tape deck controls allow the user to:

- rewind to the start of the sequence (i.e. the first plane)
- step backwards by one plane
- play backwards, or repetitively step backwards
- stop any current play
- play forwards, or repetitively step forwards
- step forwards by one plane
- fast forward to the end of the sequence

Below the tapedack buttons is an editable text box indicating the current frame number and a sunken label showing the total number of frames. One can type a channel number into the current frame box to jump to that channel, for example, when 'Frequency' is on the 'Z axis'.
Below these is a right-arrow button; pressing it will display a popup frame containing sliders for setting the animation speed and the number of frames to skip when playing animations.
Below all these controls is a pair of radio buttons selecting animator mode: either 'Normal' (the default mode described above) or 'Blink'. In 'Blink' mode, when

more than one raster image is registered on the Display Panel, the tapedeck will control which one is being displayed at the moment. The images registered should cover the same portion of the sky, using the same coordinate projection. Several images can also be displayed side-by-side in this mode, with the same zoom applied to all. For details on operating the Display Panel in this mode, see the Display section of the *AIPS++ Cookbook, vol. 1*.

Events

The Viewerdisplaypanel `tool` *emits several events. A few of them can be of interest for the programmer.*

- pseudoregion *The pseudoregion event is emitted after double-clicking a region tool. The record this event holds contains unfromatted information about the region. Use* region *instead.*

- pseudoposition *The pseudoposition event is emitted after double-clicking a positioning tool. The record this event holds contains unfromatted information about the positon. Use* position *instead.*

- motion *The motion event returns a record containing (screen)pixel and world coordinate (and units) of the position the cursor is moved. It is for internal use only.*

- region *Note: This event only occurs when Viewerdisplaydata* `tools` *are registered.*

  An event called `region` *is emitted when you interactively create a region with the cursor. This event is always emitted, unless region accumulation mode is active (see above). When you terminate accumulation mode, the* `region` *event is then emitted.*

  The region event $value record is shown diagramatically below.

```
\$value
  region                # The region tool
  type                  # The region type (box, polygon etc)
```

- position *Note : This event only occurs when Viewerdisplaydata* `tools` *are registered.*

  An event called `position` *is emitted when you interactively created a position with the cursor (see above).*

  The $value of the event is a record with the fields `position` *and* `profile`.

  The position event record is shown diagramatically below.

```
\$value
  position              # Location of cursor
```

```
    pixel               #   Absolute integer pixel coordinate
    world               #   Absolute world coordinate
      numeric           #     as a numeric vector
      quantity          #     as vector of quantities
      string            #     as a formatted string
      measure           #     as measures (further subfields not shown)
    intensity           #   Intensity of image
      mask              #     mask value
      value             #     value of image (quanity)
      pixel             #     absolute integer pixel location
  profile               # The z-profile (if > 2D image)
    x                   #   abcissa
      pixel             #     pixel coordinate
        abs             #       absolute
        rel             #       relative
      nativeworld       #     world coordinate in native units
        abs             #       absolute
        rel             #       relative
      name              #     name of abcissa axis
      nativeunit        #     native abcissa unit
      currentworld      #     world coordinate as currently plotted (if plotted
)
      currentunit       #     abcissa unit as currently plotted
      currentveltype    #     velocity type as currently plotted
      currentabs        #     absolute or relative as currently plotted
      isSpectral        #     is the abcissa a spectral coordinate
    y                   #   ordinate
      unit              #     unit of ordinate
      data              #     ordinate values
```

- polyline *This event is generated by the* `polyline` *tool, when the user double-clicks, or drags and drops an existing polyline or its points.*

- statistics *Note: This event only occurs when Viewerdisplaydata* `tools` *are registered.*

  If the statistics rollup is rolled down, then statistics are written to the listboxes (`Plane`, `Full` *button press or region event emitted) then a* `statistics` *event is generated.*

  The statistics event record is shown diagramatically below. The `stats` *subfield is a record with the same format as that generated by the return record of function Image.statistics.*

```
\$value
  region           # The region from which the statistics were evaluated
  stats            # The statistics record
```

```
npts
sum
sumsq
min
max
mean
sigma
rms
blc
blcf
trc
trcf
minpos
minposf
maxpos
maxposf
```

viewerdisplaypanel - Constructor

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Create a Viewerdisplaypanel `tool`

Synopsis

viewerdisplaypanel(parent, viewer, width, height, nx, nx, maptype, newcmap, mincolors, maxcolors, autoregister, holdsdata, show, widgetset)

Arguments

parent	*parent frame for the panel* *Allowed: agent or F* *Default: F*
viewer	*the Viewer* `tool` *which will manager this* `tool` *Allowed: Viewer* `tool`
width	*width in pixels of the drawing area* *Allowed: integer* *Default: 250*
height	*height in pixels of the drawing area* *Allowed: integer* *Default: 250*
nx	*number of panels in x* *Allowed: integer* *Default: 1*
nx	*number of panels in y* *Allowed: integer* *Default: 1*
maptype	*how colors will be addressed on this Viewerdisplaypanel* `tool` *Allowed: 'index' 'rgb' 'hsv'* *Default: 'index'*
newcmap	*install a private colormap? (ignored if a parent frame is provided)* *Allowed: boolean* *Default: F*
mincolors	*minimum number of colors to allocate* *Allowed: Integer or unset* *Default: unset*
maxcolors	*maximum number of colors to allocate* *Allowed: Integer or unset* *Default: unset*
autoregister	*accept auto-register data on this Viewerdisplaypanel* `tool`*?* *Allowed: boolean* *Default: F*
holdsdata	*is this Viewerdisplaypanel* `tool` *going to hold data?* *Allowed: boolean* *Default: T*
show	*show the Viewerdisplaypanel* `tool`*?* *Allowed: boolean* *Default: T*
widgetset	*Widgetserver* `tool` *to use to build the GUI* *Allowed: Widgetserver* `tool` *or unset* *Default: unset*

Returns

Viewerdisplaypanel `tool` *or fail*

Description

This is the only constructor for Viewerdisplaypanel `tools`*. It constructs (creates) a Viewerdisplaypanel* `tool` *which can subsequently be used for the display of Viewerdisplaydata* `tools`*. If* `widgetset` *is left* `unset`*, then the Viewerdisplaypanel* `tool` *will use the Widgetserver* `tool` *used by its parent Viewer* `tool`*.*
This constructor should not be called by users: *it is called by the newdisplaypanel function of the Viewer* `tool`*, which returns a Viewerdisplaypanel* `tool`*.*

---

viewerdisplaypanel.addgui - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Add various GUI components to this Viewerdisplaypanel `tool`

Synopsis
addgui(guihasmenubar, guihascontrolbox, guihasanimator, guihasbuttons, hasdismiss, hasdone, isolationmode)

Arguments

guihasmenubar	*add a menu bar (top)?* *Allowed: boolean* *Default: T*
guihascontrolbox	*add a control box (left)?* *Allowed: boolean* *Default: T*
guihasanimator	*add an animator/tapedeck (right)?* *Allowed: boolean* *Default: T*
guihasbuttons	*add buttons (bottom)?* *Allowed: boolean* *Default: T*
hasdismiss	*add a "Dismiss" button to the GUI?* *Allowed: unset or boolean* *Default: unset*
hasdone	*add a "Done" button to the GUI?* *Allowed: unset or boolean* *Default: unset*
isolationmode	*prevent user accessing other components of this Viewer* `tool`*?* *Allowed: boolean* *Default: F*

Returns
T

Description

This function can be used to add specified graphical user interface components to an existing Viewerdisplaypanel `tool`*. This function can only be called once per Viewerdisplaypanel* `tool`*, so get it right the first time!*
If both `hasdismiss` *and* `hasdone` *are given as* `T`*, then a warning will be emitted, and* `hasdismiss` *will be forcibly set to* `F`*. If both* `hasdismiss` *and* `hasdone` *are left* `unset`*, then* `hasdone` *will be forcibly set to* `T` *and* `hasdismiss` *to* `F`*. If only one of these two arguments is specified, and the other left* `unset`*, then the* `unset` *argument will default to the opposite of the given argument.*
If `isolationmode` *is* `T`*, then the GUI added to the Viewerdisplaypanel* `tool`*will not provide the user with the buttons or menu items to create new Viewerdisplaypanel* `tools` *or Viewerdatamanager* `tools`*. Typically this option should be used whenever a non-global Viewer* `tool` *is constructed.*
See the GUI description on hpw to operate the GUI.

---

viewerdisplaypanel.adjust - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Display adjustment panel for Viewerdisplaydata `tools`

Synopsis
adjust(displaydata)

Arguments

displaydata	*name of displaydata* *Allowed: string* *Default: F*

Returns
T

Description

In the default mode this function creates and shows a graphical user interface for each Viewerdisplaydata `tool` *which is registered for display on this Viewerdisplaypanel* `tool`*. If the name of a Viewerdisplaydata* `tool` *is given only it's graphical user interface will be displayed. The nice thing about creating the adjustment panels in this way is that they can be instantly remapped to the screen even if the user has hit the "Dismiss" button. Generally, other methods of obtaining the Viewerdisplaydata* `tool` *GUIs result in GUIs which have "Done" buttons.*

---

viewerdisplaypanel.animator - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the Vieweranimator `tool` *used by this* `tool`

Synopsis
animator()

Returns
Vieweranimator `tool`

Description

This function simply returns the Vieweranimator `tool` *which is used by this Viewerdisplaypanel* `tool`*. The return value can be used to manually and automatically step through the frames available to the Viewerdisplaypanel* `tool`*.*

---

viewerdisplaypanel.annotationdd - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the agent of the annotation layer

Synopsis
annotationdd()

Returns
drawingdisplaydata agent

Description
This function retruns an DDD agent which can be used to put annotations onto the canvas.

---

viewerdisplaypanel.autoregister - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Set or query the auto-register state of this Viewerdisplaypanel `tool`

Synopsis
autoregister(set)

Arguments

set	*Auto-register on or off* *Allowed: boolean or unset* *Default: unset*

Returns
boolean

Description

This function, if called with a boolean value for `set`*, will set the auto-register state of this Viewerdisplaypanel* `tool` *to the given value, and return the same value. Otherwise, it simply returns a boolean indicating whether auto-register is on (T) or off (F).*
Any Viewerdisplaydata `tools` *subsequently created via the loaddata function of the Viewer* `tool`*, with* `autoregister=T`*, will automatically be registered on this Viewerdisplaypanel* `tool` *if auto-register is true for this Viewerdisplaypanel* `tool`*.*

---

viewerdisplaypanel.canvasmanager - Function

Package display
Module viewer
Tool viewerdisplaypanel

Return the Viewercanvasmanager `tool` *used by this* `tool`

Synopsis
canvasmanager()

Returns
Viewercanvasmanager `tool`

Description

This function simply returns the Viewercanvasmanager `tool` *which was constructed and is used by this Viewerdisplaypanel* `tool`*. The return value provides functions to modify various settings of the Viewerdisplaypanel* `tool`*.*

---

viewerdisplaypanel.canvasprintmanager - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the Viewercanvasprintmanager `tool` *used by this Viewerdisplaypanel* `tool`

Synopsis
canvasprintmanager()

Returns
Viewercanvasprintmanager `tool`

Description

This function returns the Viewercanvasprintmanager `tool` *that was constructed and is used by this Viewerdisplaypanel* `tool` *to provide printing services. The return value provides functions which can be used to control printing options.*

---

viewerdisplaypanel.viewer - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the Viewer `tool` *which manages this* `tool`

Synopsis
viewer()

Returns
Viewer `tool`

Description

This function simply returns the Viewer `tool` *which manages this Viewerdisplaypanel* `tool`*, and probably constructed it.*

---

viewerdisplaypanel.dying - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Is this Viewerdisplaypanel `tool` *in the process of destroying itself?*

Synopsis
dying()

Returns
boolean

Description

This function is provided so that tools created by (and belonging to) the Viewerdisplaypanel `tool` *can establish whether a request to destroy them came from the Viewerdisplaypanel* `tool` *itself. There are some tools, for example the Viewercanvasmanager* `tool`*, which will only allow their owning Viewerdisplaypanel* `tool` *to call their* `done` *function.*

---

viewerdisplaypanel.done - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Close down (destroy) this Viewerdisplaypanel `tool`

Synopsis
done()

Returns
T

Description

If you no longer need to use this Viewerdisplaypanel `tool`*, calling this function will free up its resources. Any Viewerdisplaydata* `tools` *which are registered on this Viewerdisplaypanel* `tool` *will be unregistered. Any graphical user interfacesassociated with this* `tool` *will be removed from the screen. An event with name* `'done'` *will be emitted when the Viewerdisplaypanel* `tool` *is destroyed.*

---

viewerdisplaypanel.upperframe - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the upper frame in the Viewerdisplaypanel `tool` *window*

Synopsis
upperframe()

Returns
frame

Description

This function is generally used internally to obtain the frame into which the Viewerdisplaypanel `tool` *menubar should be placed.*

---

viewerdisplaypanel.lowerframe - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the lower frame in the Viewerdisplaypanel `tool` *window*

Synopsis
lowerframe()

Returns
frame

Description

This function is generally used internally to obtain the frame into which the Viewerdisplaypanel `tool` *buttonbar should be placed.*

---

viewerdisplaypanel.leftframe - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the left frame in the Viewerdisplaypanel `tool` *window*

Synopsis
leftframe()

Returns
frame

Description

This function is generally used internally to obtain the frame into which the Viewerdisplaypanel `tool` *controlbox should be placed.*

---

viewerdisplaypanel.rightframe - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the right frame in the Viewerdisplaypanel `tool` *window*

Synopsis
rightframe()

Returns
frame

Description

This function is generally used internally to obtain the frame into which the Viewerdisplaypanel `tool` *Vieweranimator* `tool` *tapedeck should be placed.*

---

viewerdisplaypanel.writexpm - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Write an X11 Pixmap of the Viewerdisplaypanel `tool` *to disk*

Synopsis
writexpm(filename)

Arguments

filename	*name of X11 Pixmap file* *Allowed: string*

Returns
T

Description

This function writes an exact replica of the display area part of the Viewerdisplaypanel `tool` *to disk in X11 Pixmap format. This format can be converted to "portable pixmap format" via the utility* **xpmtoppm**, *and then onwards to many other pixmap formats, for example TIFF, BMP, etc.*

---

viewerdisplaypanel.annotator - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the viewerannotations created by this Viewerdisplaypanel `tool` *on construction*

Synopsis
annotator()

Returns
This Viewerdisplaypanel `tools` *Viewerannotations* `tool`*tool*

Description

This function returns a reference to this Viewerdisplaypanel `tools` *Viewerannotations* `tool`*tool This can be used to draw shapes onto the viewerdisplaypanel, open the annotations GUI, delete shapes etc. For more information see viewerannotations*

---

viewerdisplaypanel.newcmap - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Query whether this Viewerdisplaypanel `tool` *is using a new colormap*

Synopsis
newcmap()

Returns
boolean or unset

Description

This function returns `T` *if the Viewerdisplaypanel* `tool` *was constructed using a new (private) X Windows colormap,* `F` *if the Viewerdisplaypanel* `tool` *is using the system X Windows colormap, or* `unset` *if this information is unknown. It will be unknown in cases where the Viewerdisplaypanel* `tool` *was inserted into an existing frame, in which case the Viewerdisplaypanel* `tool` *will use the same X Windows colormap as the parent frame. (Recall that the constructor argument* `newcmap` *is ignored if a parent frame is provided.)*

---

viewerdisplaypanel.holdsdata - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Returns whether this Viewerdisplaypanel `tool` *can hold Viewerdisplaydata* `tools`

Synopsis
holdsdata()

Returns
boolean

Description

This function simply returns a boolean value indicating whether this Viewerdisplaypanel `tool` *was constructed with the* `holdsdata` *argument T or F. Viewerdisplaypanel* `tools` *for which* `holdsdata=T` *will have functions for registering and unregistering Viewerdisplaydata* `tools`*. The only use of Viewerdisplaypanel* `tools` without `holdsdata=T` *is for the Viewercolormapmanager* `tool` *graphical user interface as of this writing.*

---

viewerdisplaypanel.status - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return a record describing various attributes of the Viewerdisplaypanel `tool`

Synopsis
status()

Returns
record

Description

This function returns some fairly low-level information about the Viewerdisplaypanel `tool`*. The record returned has upto two top-level fields:* `pixelcanvas`*, and* `paneldisplay` *if this Viewerdisplaypanel* `tool` *holds data. Fields which may exist in the* `pixelcanvas` *sub-record include the following (those tagged "dynamic" can change during the lifetime of a Viewerdisplaypanel* `tool`*):*

- `width`*: width of the entire display area ("pixelcanvas") in pixels (dynamic).*
- `height`*: height of the pixelcanvas in pixels (dynamic).*
- `depth`*: bit depth of the pixelcanvas, normally 8-bit.*
- `maptype`*: maptype of the pixelcanvas, normally 'index'.*
- `xdpi`*: approximate resolution in the X direction, in dots per inch (dpi).*
- `ydpi`*: approximate resolution in the Y direction.*
- `colortablesize`*: number of colors currently allocated to the pixelcanvas (dynamic).*
- `maxcolortablesize`*: maximum number of colors that could immediately be allocated to the pixelcanvas (dynamic).*
- `buildincolormaps`*: names of the colormaps that are known to the pixelcanvas.*
- `registeredcolormaps`*: a record having sub-fields* `names`*, which lists the names of the registered colormaps, and* `sizes`*, which lists the length of each of the named, registered colormaps. Normally the values in* `sizes` *should add up to* `colortablesize`*, since the colormaps will dynamically resize to fill the colortable.*

Fields which may exist in the `paneldisplay` *sub-record (they are all dynamic) include:*

- `nxpanels`*: The number of panels in x direction on this paneldisplay.*
- `nypanels`*: The number of panels in y direction on this paneldisplay.*
- `xorigin`*: The blcX of this paneldisplay on the pixelcanvas (0.0-1.0).*
- `yorigin`*: The blcY of this paneldisplay on the pixelcanvas (0.0-1.0).*
- `xsize`*: The extent in x of this paneldisplay on the pixelcanvas (0.0-1.0).*
- `xsize`*: The extent in y of this paneldisplay on the pixelcanvas (0.0-1.0).*
- `linearblc`*: linear coordinates* `[x, y]` *of the bottom-left corner of each panel in the paneldisplay.*

- `lineartrc`: *linear coordinates* `[x, y]` *of the top-right corner of each panel in the paneldisplay.*
- `worldblc`: *world coordinates* `[x, y]` *of the bottom-left corner of each panel in the paneldisplay.*
- `worldtrc`: *world coordinates* `[x, y]` *of the top-right corner of each panel in the paneldisplay.*
- *axisunits: the units of the world axes*

---

viewerdisplaypanel.getoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return a record describing some modifiable state of this `tool`

Synopsis
getoptions()

Returns
record

Description

This function returns a record which describes some of the internal state of this Viewerdisplaypanel `tool`*. The record differs from that returned by the* `status tool function` *in both format and content: it is suitable for construction of an Autogui* `tool`*, and contains explicitly modifiable parameters. A brief description of the fields which* may *exist in the returned record follows:*

- `colortablesize`: *the number of colors currently allocated to this Viewerdisplaypanel* `tools` *pixelcanvas.*
- `papercolors`: *a boolean specifying whether screen or paper colors (black/white or white/black background/foreground) should be used.*
- `leftmarginspacepg`: *left margin space in PgPlot character units.*
- `bottommarginspacepg`: *bottom margin space in PgPlot character units.*

- `rightmarginspacepg`: *right margin space in PgPlot character units.*
- `topmarginspacepg`: *top margin space in PgPlot character units.*
- `nxpanels`: *The number of panels in x direction on this paneldisplay.*
- `nypanels`: *The number of panels in y direction on this paneldisplay.*
- `xspacing`: *The space between adjacent panels in x direction.*
- `yspacing`: *The space between adjacent panels in y direction.*

---

viewerdisplaypanel.setoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Set the modifiable state of this `tool`

Synopsis
setoptions(options)

Arguments

options	*options record describing the requested state* *Allowed: record*

Returns
T
This function should be used to set the internal state of a Viewerdisplaypanel `tool`*, according to the field names of the record returned by the getoptions* `tool function`*. Installing new options on a Viewerdisplaypanel* `tool` *will generally cause the refresh (that is, a total re-drawing) of any Viewerdisplaydata* `tools` *registered on the Viewerdisplaypanel* `tool`*. This is because modifying the options will cause a change in the appearance of the data.*

---

viewerdisplaypanel.hold - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Prevent refreshes occuring on this Viewerdisplaypanel `tool`

Synopsis
hold()

Returns
T

Description

This function will prevent refreshing occuring on this Viewerdisplaypanel `tool` *until an equal number of calls to the release function have been made. Typically this should be used when multiple operations need to be carried out before the user is shown the result of the operations.*

---

viewerdisplaypanel.release - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Allow refreshes to occur on this Viewerdisplaypanel `tool`

Synopsis
release()

Returns
T

Description

This function will re-enable refreshing occuring on this Viewerdisplaypanel `tool` *provided an equal number of calls have been made to the hold function.*

---

viewerdisplaypanel.disablecontrols - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Temporarily disable the controls on this Viewerdisplaypanel `tool`

Synopsis
disablecontrols()

Returns
T

Description

This function will simply suspend the actions of controls on this Viewerdisplaypanel `tool`*. This means, for example, that the zoom and colormap fiddling controls will not work temporarily.*

---

viewerdisplaypanel.enablecontrols - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Re-enable the controls on this Viewerdisplaypanel `tool`

Synopsis
enablecontrols()

Returns
T

Description

This function re-installs the controls on this Viewerdisplaypanel `tool` *after they have been disabled with the disablecontrols function.*

---

viewerdisplaypanel.newcanvasmanagergui - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Create and (probably) show a new graphical user interface for this `tool`*'s Viewercanvasmanager* `tool`

Synopsis
newcanvasmanagergui(parent, show, hasdismiss, hasdone, widgetset)

Arguments

parent	*parent frame in which to place the Viewercanvasmanager* `tool` *Allowed: agent or F* *Default: F*
show	*show the Viewercanvasmanager* `tool`*?* *Allowed: boolean* *Default: T*
hasdismiss	*add a "Dismiss" button to the GUI?* *Allowed: boolean* *Default: F*
hasdone	*add a "Done" button to the GUI?* *Allowed: boolean* *Default: F*
widgetset	*Widgetserver* `tool` *to use to build the GUI* *Allowed: Widgetserver* `tool` *or unset* *Default: unset*

Returns
`tool` *having Viewerstandardgui interface*

Description

This function will create a new graphical user interface which can be used to control this Viewerdisplaypanel `tool`*'s Viewercanvasmanager* `tool`*. By default, the GUI is placed on the screen, although it can be constructed for later use.*

---

viewerdisplaypanel.newcanvasprintmanagergui - Function

Package display
Module viewer
Tool viewerdisplaypanel

Create and (probably) show a new graphical user interface for this `tool`*'s Viewercanvasprintmanager* `tool`

Synopsis

newcanvasprintmanagergui(parent, show, hasdismiss, hasdone, widgetset)

Arguments

parent	*parent frame in which to place the Viewercanvasprintmanager* `tool` *Allowed: agent or F* *Default: F*
show	*show the Viewercanvasprintmanager* `tool`*?* *Allowed: boolean* *Default: T*
hasdismiss	*add a "Dismiss" button to the GUI?* *Allowed: boolean* *Default: F*
hasdone	*add a "Done" button to the GUI?* *Allowed: boolean* *Default: F*
widgetset	*Widgetserver* `tool` *to use to build the GUI* *Allowed: Widgetserver* `tool` *or unset* *Default: unset*

Returns

`tool` *having Viewerstandardgui interface*

Description

This function will create a new graphical user interface which can be used to control this Viewerdisplaypanel `tool`*'s Viewercanvasprintmanager* `tool`*. By default, the GUI is placed on the screen, although it can be constructed for later use.*

---

viewerdisplaypanel.testpattern - Function

Package display
Module viewer
Tool viewerdisplaypanel

Turn a test pattern on or off on the Viewerdisplaypanel `tool`

Synopsis
testpattern(onoff)

Arguments

onoff	*turn pattern on (T) or off (F)* *Allowed: boolean*

Returns
T

Description

Calling this function with `onoff=T` *will place a "test pattern" on the Viewerdisplaypanel* `tool` *which is drawn with each refresh. The test pattern is really available for, surprise, surprise, testing! It is actually the test pattern which is used to display the registered colormaps in the graphical user interface for the Viewercolormapmanager* `tool`.

---

viewerdisplaypanel.type - Function

Package display
Module viewer
Tool viewerdisplaypanel

Return the type of this `tool`

Synopsis
type()

Returns
'viewerdisplaypanel'

Description

This function simply returns the string `'viewerdisplaypanel'`*, which can be used to identify this* `tool` *as a Viewerdisplaypanel* `tool`.

---

viewerdisplaypanel.registercolormap - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Explicitly register a colormap on the Viewerdisplaypanel `tool`

Synopsis
registercolormap(map)

Arguments

map	*name of colormap to register* *Allowed: string*

Returns
T

Description

If the given colormap name is valid, then it will be explicitly registered on this Viewerdisplaypanel `tool`*. The result of this will be a redistribution of the available color cells to the registered colormaps. A change will only be noticed if previously the named colormap was* not *registered on this Viewerdisplaypanel* `tool`.
This function is generally reserved for internal use; careless use of it could cause the Viewer `tool` or AIPS++ to fail.

---

viewerdisplaypanel.unregistercolormap - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Explicitly unregister a colormap on the Viewerdisplaypanel `tool`

Synopsis
unregistercolormap(map)

Arguments

map	*name of colormap to unregister* *Allowed: string*

Returns
T

Description

This function simply explicitly un*registers the named colormap, provided it is a valid colormap and is already registered on this Viewerdisplaypanel* `tool`.
This function is generally reserved for internal use, and careless use of it could cause the Viewer `tool` or AIPS++ to fail.

---

viewerdisplaypanel.register - Function

Package display
Module viewer
Tool viewerdisplaypanel

Register a Viewerdisplaydata `tool` *for drawing*

Synopsis
register(displaydata)

Arguments

displaydata	*Viewerdisplaydata* `tool` *to register* *Allowed: Viewerdisplaydata* `tool`

Returns
T, F or fail

Description

This function registers the given Viewerdisplaydata `tool` *on this Viewerdisplaydata* `tool`. *That means that at the next and subsequent refreshes, until it is unregistered, the given Viewerdisplaydata* `tool` *will attempt to draw itself on this Viewerdisplaypanel* `tool`.
A return value of `T` *indicates that the Viewerdisplaydata* `tool` *was successfully registered on this Viewerdisplaypanel* `tool`. *A return value of* `F` *indicates that the Viewerdisplaydata* `tool` *was already registered, so no action was performed. The return value will be* `fail` *if the Viewerdisplaydata* `tool` *was unknown to, or could not be found by, this Viewerdisplaypanel* `tool`.

---

viewerdisplaypanel.unregister - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Unregister a Viewerdisplaydata `tool` *from this Viewerdisplaypanel* `tool`

Synopsis
unregister(displaydata)

Arguments

displaydata	*Viewerdisplaydata* `tool` *to unregister* *Allowed: Viewerdisplaydata* `tool`

Returns
T, F or fail
This function removes the given Viewerdisplaydata `tool` *from the list of Viewerdisplaydata* `tools` *which will (attempt to) draw themselves on this Viewerdisplaypanel* `tool` *at each refresh.*
A return value of `T` *indicates that the Viewerdisplaydata* `tool` *was successfully unregistered from this Viewerdisplaypanel* `tool`. *A return value of* `F` *indicates that the Viewerdisplaydata* `tool` *was not previously registered, so no action was performed. The return value will be* `fail` *if the Viewerdisplaydata* `tool` *was unknown to, or could not be found by, this Viewerdisplaypanel* `tool`.

---

viewerdisplaypanel.isregistered - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Query whether a Viewerdisplaydata `tool` *is registered on this Viewerdisplaypanel* `tool`

Synopsis
isregistered(displaydata)

Arguments

displaydata	*Viewerdisplaydata* `tool` *to query* *Allowed: Viewerdisplaydata* `tool`

Returns
T, F or fail
This function simply returns `T` *or* `F` *indicating (respectively) that the given Viewerdisplaydata* `tool` *is or is not currently registered for display on this Viewerdisplaypanel* `tool`. *The return value will be* `fail` *if the Viewerdisplaydata* `tool` *was unknown to, or could not be found by, this Viewerdisplaypanel* `tool`.

---

viewerdisplaypanel.registerednames - Function

Package display
Module viewer
Tool viewerdisplaypanel

Return a vector of the names of registered Viewerdisplaydata `tools`

Synopsis
registerednames()

Returns
Vector of strings

Description

This function returns a vector of names of all Viewerdisplaydata `tools` *registered on this Viewerdisplaypanel* `tool`. This function is generally reserved for internal use.

---

viewerdisplaypanel.registrationflags - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return a record describing which Viewerdisplaydata `tools` *are registered on this* `tool`

Synopsis
registrationflags()

Returns
record

Description

This function returns a record whose top-level field names are those of the Viewerdisplaydata `tools` *known to the Viewer* `tool` *which owns this Viewerdisplaypanel* `tool`*. The value of each field is a boolean, which if T indicates that that Viewerdisplaydata* `tool` *is registered for display on this Viewerdisplaypanel* `tool`*.*
This function is generally reserved for internal use.

---

viewerdisplaypanel.unregisterall - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Unregister all Viewerdisplaydata `tools` *from this Viewerdisplaypanel* `tool`

Synopsis
unregisterall()

Returns
T

Description

This function simply calls the unregister function for every Viewerdisplaydata `tool` *that is registered on this Viewerdisplaypanel* `tool`*. It is akin to "clearing" the Viewerdisplaypanel* `tool`*. The unregistrations are wrapped in a* `hold/release` *scope, so that only one refresh is made at the end of this function.*

---

viewerdisplaypanel.unzoom - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Unzoom the current view

Synopsis
unzoom()

Returns
T

Description

This function will modify the coordinates on the Viewerdisplaypanel `tool` *such that the full view (of at least the first registered Viewerdisplaydata* `tool`*) is visible.*

---

viewerdisplaypanel.setzoom - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Set a specific zoom

Synopsis
setzoom(blc, trc)

Arguments

blc	*bottom-left linear coordinates of zoom* *Allowed: 2-vector*
trc	*top-right linear coordinates of zoom* *Allowed: 2-vector*

Returns
T

Description

This function sets the linear coordinate system of the Viewerdisplaypanel `tool` *to the specified blc and trc. Linear coordinates are generally identical to the pixel coordinates of the first registered Viewerdisplaydata* `tool`.

---

viewerdisplaypanel.zoom - Function

Package display
Module viewer
Tool viewerdisplaypanel

Zoom by a given factor

Synopsis
zoom(xfac, yfac)

Arguments

xfac	*scale factor in X direction* *Allowed: float* *Default: 2.0*
yfac	*scale factor in Y direction* *Allowed: unset or float* *Default: unset*

Returns
T

Description

If `yfac=unset`*, then* `yfac` *is forced to be identical to* `xfac`*. This function zooms in by the specified amount in each direction, centred on the centre of the current viewable area. For scale factors greater than one, the view is magnified, and for less than one, the view is demagnified. There will be some interplay between the* `aspect` *option of Viewerdisplaydata* `tools`*and the zoom setting.*

1.3.7 viewerslicedp - Tool

Package display
Module viewer

Tool for providing 3D slice canvases
include "viewerslicedp.g"

Constructors

viewerslicedp	*Create a Viewerslicepanel* `tool`

Functions

preload	*Cache all views in display memory*
setoptions	*Set the modifiable state of this* `tool`

Description

This `tool` *sets up a canvas with three panels on it. These panels present the user with different views of the Viewerdisplaydata* `tools`*. Only data which has more than two axes and more than one pixel on the third axis will be displayed on this panel. The top left panel shows the "standard" axis1 vs. axis2. The top right panel displays axis3 vs. axis2 and the bottom right panel shows axis1 vs. axis 3.*
The functions of this tool are almost identical to the Viewerdisplaypanel `tool`*. This documentation only highlights functions not available to this tool and functions additional or different to the Viewerdisplaypanel* `tool`*.*
This specialized displaypanel has no printing facility yet, although X11 pixmaps can be created using the the function viewerslicedp.writexpm.

Graphical User Interface

The standard graphical user interface provided by the Viewerslicepanel `tool` *is shown in Figure 1.6. Surrounding the actual display area itself are four GUI components: the menu bar at the top, the button bar at the bottom, the tracking*

bar/s just above the buttons and the controlbox on the left. The menu and the tracking bar are identical to the ones in the Viewerdisplaypanel `tool`.

The button bar *This* `tool` *provides a* `Preload...` *button. The* `Print...` *button is currently not available.*

The control box *The control box has one additional button, the* `Multi Panel Crosshair` *tool. If this tool gets dragged across the canvas on one panel it "animates" the other two panels since the coordinates on those two are dependend on ths first one.*

- **Multi Panel Cursor:** *Press the "Esc" key to cancel the crosshair drawn.*

All other GUIelements inherited from the Viewerdisplaypanel are described here.

viewerslicedp - Constructor

Package	**display**
Module	**viewer**
Tool	**viewerslicedp**

Create a Viewerslicepanel `tool`

Synopsis

viewerslicedp(parent, viewer, width, height, maptype, newcmap, mincolors, maxcolors, autoregister, holdsdata, show, widgetset)

Arguments

parent	*parent frame for the panel* *Allowed: agent or F* *Default: F*
viewer	*the Viewer* `tool` *which will manager this* `tool` *Allowed: Viewer* `tool`
width	*width in pixels of the drawing area* *Allowed: integer* *Default: 250*
height	*height in pixels of the drawing area* *Allowed: integer* *Default: 250*
maptype	*how colors will be addressed on this Viewerslicepanel* `tool` *Allowed: 'index' 'rgb' 'hsv'* *Default: 'index'*
newcmap	*install a private colormap? (ignored if a parent frame is provided)* *Allowed: boolean* *Default: F*
mincolors	*minimum number of colors to allocate* *Allowed: Integer or unset* *Default: unset*
maxcolors	*maximum number of colors to allocate* *Allowed: Integer or unset* *Default: unset*
autoregister	*accept auto-register data on this Viewerslicepanel* `tool`*?* *Allowed: boolean* *Default: F*
holdsdata	*is this Viewerslicepanel* `tool` *going to hold data?* *Allowed: boolean* *Default: T*
show	*show the Viewerslicepanel* `tool`*?* *Allowed: boolean* *Default: T*
widgetset	*Widgetserver* `tool` *to use to build the GUI* *Allowed: Widgetserver* `tool` *or unset* *Default: unset*

Returns
Viewerslicepanel `tool` *or fail*

Description

This is the only constructor for Viewerslicepanel `tools`*. It constructs (creates) a Viewerslicepanel* `tool` *which can subsequently be used for the display of*

Viewerdisplaydata `tools`. *If* `widgetset` *is left* `unset`, *then the Viewerslicepanel* `tool` *will use the Widgetserver* `tool` *used by its parent Viewer* `tool`.
This constructor should not be called by users: *it is called by the newdisplaypanel function of the Viewer* `tool`, *which returns a Viewerslicepanel* `tool`.

---

viewerslicedp.setoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerslicedp**

Set the modifiable state of this `tool`

Synopsis
setoptions(options)

Arguments

options	*options record describing the requested state* *Allowed: record*

Returns
T
This function is distributes the options over all views of the Viewerdisplaydata `tool`. *Otherwise it is identical to viewerdisplaypanel.setoptions*

---

viewerslicedp.preload - Function

Package	**display**
Module	**viewer**
Tool	**viewerslicedp**

Cache all views in display memory

Synopsis
preload()

Returns

T
This function animates over all views of the Viewerdisplaydata `tool`*. After this loop has been completed the Multipanel Crosshair tool will instantly cause an animation on the other panels, because there is no computation involved.*
Note: If axis labels are enabled there won't be instant animation and you will notice a delay between moving your cursor and display on the other panels.
Note: Once you have decided to execute this function you can't stop it until it has finished.

1.3.8 viewerstandardgui - Tool

Package display
Module viewer

Interface for operating Viewer `tool` *graphical user interfaces*

Constructors

Functions

disable	*Disable this graphical user interface*
dismiss	*Hide this graphical user interface*
done	*Close down (finished with) this graphical user interface*
enable	*Enable this graphical user interface*
gui	*Map this graphical user interface*
ismapped	*Is this graphical user interface mapped to the screen or its parent frame?*
map	*Map the graphical user interface to the screen*
unmap	*Remove the graphical user interface from the screen*

Description

This is not a tool. Instead, this simply defines an interface which is shared by all graphical user interfaces which are constructed by the Viewer `tool` **or by its support** `tools`**.**

viewerstandardgui.map - Function

Package display
Module viewer
Tool viewerstandardgui

Map the graphical user interface to the screen

Synopsis
map(force)

Arguments

force	*force map even if this* `tool` *seems to be mapped already* *Allowed: boolean* *Default: F*

Returns
boolean

Description

This function should be used to map a graphical user interface to the screen. If the `tool` *made its own top-level frame, then that frame will be mapped to the screen. Otherwise, the next highest level frame will be mapped into its parent frame, which itself may or may not be presently mapped to the screen.*
A boolean value of `T` *is returned if an action was taken to map the GUI. Otherwise,* `F` *is returned, implying that the* `tool` *believed itself to already be mapped.*

---

viewerstandardgui.unmap - Function

Package display
Module viewer
Tool viewerstandardgui

Remove the graphical user interface from the screen

Synopsis
unmap(force)

Arguments

force	*force unmap even if this* `tool` *seems to be unmapped already* *Allowed: boolean* *Default: F*

Returns
boolean

Description

This function should be used to remove a graphical user interface from the screen. If the `tool` *made its own top-level frame, then that entire frame will be removed from the screen. Otherwise, the next highest level frame will be removed from its parent frame, which itself may or may not be presently mapped to the screen. A boolean value of* `T` *is returned if an action was taken to unmap the GUI. Otherwise,* `F` *is returned, implying that the* `tool` *believed itself to already be unmapped.*

---

viewerstandardgui.done - Function

Package display
Module viewer
Tool viewerstandardgui

Close down (finished with) this graphical user interface

Synopsis
done()

Returns
T

Description

If you no longer need to use this graphical user interface, calling this function will remove it from the screen, disable any interaction with it, and free up its resources. This function is called when the user presses the `Dismiss` *button on the GUI* if *the GUI was constructed with* `hasdone=T` *and* `hasdismiss=F`.

---

viewerstandardgui.ismapped - Function

Package	**display**
Module	**viewer**
Tool	**viewerstandardgui**

Is this graphical user interface mapped to the screen or its parent frame?

Synopsis
ismapped()

Returns
boolean

Description

This function simply returns a boolean value of `T` *which indicates that this graphical user interface is mapped to the screen (or into its parent frame), or* `F`*, meaning it is presently unmapped.*

---

viewerstandardgui.dismiss - Function

Package	**display**
Module	**viewer**
Tool	**viewerstandardgui**

Hide this graphical user interface

Synopsis
dismiss()

Returns
boolean

Description

This function simply calls unmap with `force=F`*. The return value is the result of this invokation. This function is called when the user presses the* `Dismiss` *button on the GUI* if *the GUI was constructed with* `hasdismiss=T`*.*

---

viewerstandardgui.gui - Function

Package	**display**
Module	**viewer**
Tool	**viewerstandardgui**

Map this graphical user interface

Synopsis
gui()

Returns
boolean

Description

This function simply calls map with `force=F`. *The return value is the result of this invokation.*

---

viewerstandardgui.disable - Function

Package	**display**
Module	**viewer**
Tool	**viewerstandardgui**

Disable this graphical user interface

Synopsis
disable()

Returns
boolean

Description

If this graphical user interface `tool` *created its own top-level frame, then this function disables ("greys") that frame, sets the cursor to a "watch" icon, and returns* `T`*. Otherwise, it does nothing and returns* `F`*.*

---

viewerstandardgui.enable - Function

Package	**display**
Module	**viewer**
Tool	**viewerstandardgui**

Enable this graphical user interface

Synopsis
enable()

Returns
boolean

Description

If this graphical user interface `tool` *created its own top-level frame, then this function enables ("ungreys") that frame, returns the cursor to the standard pointer shape, and returns* `T`*. Otherwise, it does nothing and returns* `F`*.*

1.3.9 viewerdatamanager - Tool

Package	**display**
Module	**viewer**

Tool for importing data into the Viewer
include "viewerdatamanager.g"

Description

The Data manager is a graphical user interface only tool to provide a context sensitive import of data into the Viewer `tool`*. It is a front-end to the viewer.loaddata function in the Viewer* `tool`*.*

This GUI has three major control areas and a couple of buttons an a menu:

File Chooser *The top-left corner shows a "File Chooser", a text field to enter directories and a listbox to select your data files on disk. This listbox shows the filtered content of the specified directory, i.e. only types understood by the Viewer* `tool` *are listed. The file types supported are can be found in viewer.loaddata.*

Tool Chooser *Here all supported* Glish *tools are listed, e.g if you have opened an image tool with the variable name* `im` *it will show up here.*

Display Type selector *Once you have selected a supported file or tool, the box called "DisplayData Type" to the right, shows all the drawing types which are supported for this kind of data. The possible types are:*

- *Raster Image*
- *Contour Map*
- *Vector Map*
- *Marker Map*
- *Sky catalog overlay*

Update *Update the contents of the current directory and* Glish *tools.*

Autoregister *Your data will automatically be registered (displayed) on all Viewerdisplaypanel* `tools`*. In some cases like the* `Skycatalog` *you might want to disable this to make sure you don't register it by itself.*

Menu bar *The* File *entry of the menu bar is identical to the same menu item in viewerdisplaypanel gui.*

1.3.10 viewerannotations - Tool

Package **display**
Module **viewer**

Tool for drawing annotations onto a displaypanel.
include "viewerannotations.g"

General description

Description

This `tool` *allows the user to interactively draw shapes and text onto Viewerdisplaypanel* `tools`*. It has a GUI, allowing selection of the drawing tool to use, and also the ability to change settings for a specified drawing object. Annotations can also be constructed and modified via option records passed to functions on the command line. The* `tool` *also supports printing of the user's annotations onto postscript. On creation of a Viewerdisplaypanel* `tool`*, a viewerannotation is automatically created, the GUI of which can be accessed by selecting the "Annotations..." option from the "Tools" menu. Each viewerannotation* `tool` *must be associated with a Viewerdisplaypanel* `tool`*.*

Graphical User Interface

Description

The graphical user interface provided by the viewerannotations tool is designed to provide access to common settings and actions used when annotating a displaypanel. The GUI contains buttons, which are broken into two main categories. There are three control buttons, at the top left of the GUI . They can be used to assign a mouse button to control of annotations, to move / rotate shapes and to delete shapes. The other buttons are all tools, which allow the user to create new shapes. After clicking on a shape, the user can position the shape on the displaypanel. The button on the far right, which is the marker tool opens a pop up menu when clicked on. This menu shows a list of available markers, which can then be placed on the displaypanel.
The other main section of the GUI is a expandable options section. By default the roll-up is "rolled-up", and hence the options are not visible. By rolling down the options section, the user has access to a context specific menu which shows options for the shape they have currently selected.
The figure above shows the annotation control GUI. Everything below the Options roll-up is context specific, depending on which shape you have selected. The control buttons, and also the tool buttons can be seen above the roll-up.
The figure above shows annotations objects being edited. The selected object is shown using green "handles". The handles can be moved to change the size of the shape, and the shape can be moved by clicking and dragging inside the shape. Keyboard modifiers can be used to change the action performed by mouse events. Holding "Control" on the keyboard changes the effect of moving the handles to that of rotation, and the effect of dragging a shape to that of scaling the object (about its center). If more than one shape occupies a location on the screen, or if a larger object is covering a smaller object, the underlying objects can be accessed by repeatedly clicking in the same spot.
Note: *All AIPS++ GUIs have a so-called popup help. Just rest your mouse cursor on top of a GUI element (e.g. a button) and a little frame will appear outlining the funtionality of this element. If you then press and hold down the right mouse button you will get a detailed description. The information given there should be sufficient to let you operate that GUI element.*

Events

The Viewerannotations `tool`*tool does not emit any events. It is not really required to, since its job is to control event handling internally. All of the actual mouse events are handled by the underlying proxy. maxposf*

viewerannotations - Constructor

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Create a Viewerannotations `tool`*tool*

Synopsis

viewerannotations(displaypanel, title, widgetset)

Arguments

displaypanel	*displaypanel to attach to* *Allowed: agent*
title	*the title to use for the GUI frame* *Allowed: String* *Default: "Unknown Draw Area"*
widgetset	*Widgetserver* `tool` *to use to build the GUI* *Allowed: Widgetserver* `tool` *or unset* *Default: ddlws*

Returns

Viewerannotations `tool`*tool or fail*

Description

This is the only constructor for the Viewerannotations `tool`*tool.It constructs (creates) a Viewerannotations* `tool`*tool which attaches itself to the supplied Viewerdisplaypanel* `tool`*.This constructor will generally not need to be called by users, since each Viewerdisplaypanel* `tool`*creates its own Viewerannotations* `tool`*tool at construction time.*

---

viewerannotations.done - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Destroy this Viewerannotations `tool`*tool.*

Synopsis
done()

Returns
T

Description

This function is used when the Viewerannotations `tool`*tool is no longer required. When the Viewerdisplaypanel* `tool` *to which the annotator is attached is 'done', this function is called automatically.*

---

viewerannotations.dismiss - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Remove the Viewerannotations `tool`*tool GUI from the screen*

Synopsis
dismiss()

Arguments

Returns
T

Description

This function simply removes (if it is visible) the Viewerannotations `tool`*tool GUI from the screen. The GUI can be redisplayed, either by the user clicking the appropriate menu option, or by a call to the* `gui` *function.*

---

viewerannotations.print - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Print the annotations onto the supplied pixelcanvas

Synopsis
print(pixelcanvas)

Arguments

pixelcanvas	*PixelCanvas proxy to print the annotations to* *Allowed: Pixelcanvas proxy*

Returns
T

Description

This function is mainly designed for internal use. When the Viewerdisplaypanel `tool` *is to be printed onto a Postscript PixelCanvas, the pixelcavas is passed to this function, which draws all the current annotations onto it.*

---

viewerannotations.newshape - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Create a new shape based on a record

Synopsis
newshape(shapesettings)

Arguments

shapesettings	*Settings describing the shape* *Allowed: Record*

Returns

Description
This function makes a new shape based entirely on the information provided in the record passed to it. Once on screen, the shape can be manipulated by the user.

---

viewerannotations.deleteshape - Function

Package display
Module viewer
Tool viewerannotations

Delete a shape from the annotations

Synopsis
deleteshape(index)

Arguments

index	*Index of the shape* *Allowed: Integer*

Returns

Description

This function deletes the shape with the specified index. The index of the currently active shape can be determined using the `whichshape` *function.*

---

viewerannotations.whichshape - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Return the index of the currently active shape

Synopsis
whichshape()

Returns
Integer

Description

This function returns the index of the selected / active shape.

---

viewerannotations.createshape - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Create a shape, based on options and also user input

Synopsis
createshape()

Returns

Description

This function allows the user to create a shape by "dragging it out". Some of the options including shape type, and miscellaneous options must be supplied via the option record. Options regarding position of the shape can be omitted, and these

will be obtained the next time the user clicks in the draw area. This allows for the user to select a shape, and "drag it out".

---

viewerannotations.availableshape - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Return a list of the available shapes

Synopsis
availableshape()

Returns
Array of Strings

Description

This function simply returns an array which is filled with a list of the available shapes.

---

viewerannotations.setkey - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Set the key (mouse button) which the user will use to control the annotations

Synopsis
setkey(keysym)

Arguments

keysym	*New key symbol* *Allowed: Key Symbol, some are defined as glish global variables*

Returns
T

Description

Different tools can be assigned to different mouse buttons. This function allows the user to specify which button they wish to use to control the annotations.

---

viewerannotations.getalloptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Return options for all the shapes currently displayed

Synopsis
getalloptions()

Returns
Record

Description

This will return a record, which contains the information for all of the annotation shapes. The format of the record is as follows:

- *Shape 1:*
  - *Options for shape 1...*
- *Shape 2:*
  - *Options for shape 2...*
- *Shape n:*
  - *Options for shape n...*

;

---

viewerannotations.setalloptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Sets the annotations based on information provided in the record

Synopsis
setalloptions()

Returns
Record

Description

This sets the annotations based on a record. All current shapes are deleted, and replaced with those specified in the option record. The format for the record passed as a parameter is the same as that which is returned by `getalloptions`.

---

viewerannotations.getshapeoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Returns the option record for the specified shape

Synopsis
getshapeoptions(whichshape)

Arguments

whichshape	*Return options for this shape* *Allowed: Integer*

Returns
Record

Description

This function returns the option record for the shape with the specified index. The index of the current / selected shape can be determined by using the `whichshape` *function.*

Example

```
#Return the options for the currently selected shape:
annotator := mydisplaypanel.annotator();
shapeoptions := annotator.getshapeoptions(annotator.whichshape());
```

---

viewerannotations.setshapeoptions - Function

Package display
Module viewer
Tool viewerannotations

Set the options of a specific shape

Synopsis

setshapeoptions(whichshape, newoptions)

Arguments

whichshape	*Set the options for this shape* *Allowed: Integer*
newoptions	*Options to set* *Allowed: Record*

Returns

Description

This function sets options for a specific shape. The index of the currently selected / active shape can be determined using the `whichshape` *function.*

---

viewerannotations.addlockedtocurrent - Function

Package display
Module viewer
Tool viewerannotations

"Lock" a shape to the currently selected one

Synopsis
addlockedtocurrent(whichshape)

Arguments

whichshape	*Which shape do you wish to lock to the selected one* *Allowed: Integer*

Returns

Description

This function can lock a shape to the currently selected one. For example, if the currently selected object is an arrow, and the `whichshape` *variable is set to the index of text, then whenever the arrow is moved by the user, so too will the text.*

---

viewerannotations.removelockedfromcurrent - Function

Package display
Module viewer
Tool viewerannotations

Remove a locked item from the currently selected shape.

Synopsis
removelockedfromcurrent(lockeditemindex)

Arguments

lockeditemindex	*Index of shape to remove from lock* *Allowed: Integer*

Returns

Description

This functions performs the opposite of `addlockedtocurrent`*. It allows the user to 'unlock' a shape from the currently selected one.*

1.3.11 skycatalog - Tool

Package display
Module viewer

Tool to create and manipulate Viewer `tool` *readable skycatalog tables include "skycatalog.g"*

Constructors

skycatalog	*Create a skycatalog* `tool`

Functions

addunit	*Add a unit to a column*
done	*Close this* `tool`
fromascii	*Create the* `tool` *from an ascii file*
fromcomponentlist	*Create the* `tool` *from a componentlist*
mergecolumns	*Merge n columns into one*
name	*Return the name of the table*
renamecolumn	*Rename a column*
table	*Return the underling table* `tool`
type	*Return the* `tool` *type*

Description

This `tool` *allows the user to create so called skycatalog tables readable by the Viewer* `tool`*. This can be done from ASCII tables or componentlists. It also provides a front-end to the table* `tool` *to manipulate the structure of the*

skycatalog.

Example
Here is a small example on how to convert an ascii table "my.ascii" to an AIPS++ table "my.tbl":
The ASCII file looks like this:

Name	*No*	*RA*	*DEC*	*NED-ID*	*Max*	*mom0*
A	*A*	*A*	*A*	*A*	*R*	*R*
NED	*16*	*19:44:47*	*-14:46:51*	*NGC6822ee*	*24.255*	*159.8*
NED	*22*	*01:04:46*	*+02:07:04*	*IC1613e*	*13.002*	*328.1*

```
include 'skycatalog.g';
sca := skycatalog('myscat1.tbl');
sca.fromascii(asciifile='my.ascii',hasheader=T,longcol='RA',latcol='DEC');
# cosmetics
sca.mergecolumns("Name No");
sca.done()
```

Example
An example using a componentlist:

```
im := imagemaketestimage()
cl := im.findsources()
sca := skycatalog('myscat2.tbl')
sca.fromcomponentlist(cl)
sca.done()
# That's it !
```

skycatalog - Constructor

Package	**display**
Module	**viewer**
Tool	**skycatalog**

Create a skycatalog `tool`

Synopsis
skycatalog(filename)

Arguments

filename	*Name of the skycatalog table* *Allowed: String* *Default: unset*

Returns
skycatalog `tool` *or fail*

Description

Construct a skycatalog `tool`*. The specified filename will either be used to associate this* `tool` *wwith skycatalog table on disk or to create a new skycatalog table, which can be filled from a componentlist or ASCII file.*

---

skycatalog.addunit - Function

Package display
Module viewer
Tool skycatalog

Add a unit to a column

Synopsis
addunit(colname, unit)

Arguments

colname	*column name* *Allowed: String*
unit	*Unit to put into column* *Allowed: String*

Returns

Description

---

skycatalog.done - Function

Package	**display**
Module	**viewer**
Tool	**skycatalog**

Close this `tool`

Synopsis
done(delete)

Arguments

delete	*delete the table* *Allowed: Boolean* *Default: F*

Returns

Description

---

skycatalog.fromascii - Function

Package	**display**
Module	**viewer**
Tool	**skycatalog**

Create the `tool` *from an ascii file*

Synopsis
fromascii(asciifile, hasheader, longcol, latcol, dirtype, unit)

Arguments

asciifile	*The source ASCII file* *Allowed: String* *Default: unset*
hasheader	*Does the ASCII file include a header definition?* *Allowed: Boolean* *Default: F*
longcol	*The column name or number containing the Longitude component* *Allowed: String or Integer (see description)* *Default: unset*
latcol	*The column name or number containing the Latitude component* *Allowed: String or Integer (see description)* *Default: unset*
dirtype	*The direction type of the coordinate* *Allowed: String* *Default: J2000*
unit	*The unit of the coordinates* *Allowed: String* *Default: deg*

Returns
T or F

Description

This functions converts an ASCII table into an AIPS++ skycatalog table. These tables can be read by the Viewer `tool` *and displayed as a skycatalog overlay. Whitespace in the ASCII tables is interpreted as a column separator and the number of columns per row must be constant. The ASCII table can start with a description of the columns and the data types in that column. If this header is not specified the* `tool` *tries to determine the data type and will name the columns Column1..ColumnN. In that case the columns containing the coordinate can be given as integer indices.*
The argument dirtype indicates what direction coordinate reference type your coordinate has. At the moment only J2000, B1950, GALACTIC and SUPERGAL are valid types.
The unit is assumed to be the same for long and lat and only has to be specified if the coordinate is given decimals.

---

skycatalog.fromcomponentlist - Function

Package display

Module viewer
Tool skycatalog

Create the `tool` *from a componentlist*

Synopsis
fromcomponentlist(code)

Arguments

code	*The componentlist* `tool` *Allowed: componentlist* `tool` *Default: unset*

Returns
T or F

Description
Invoke this function to fill the skycatalog table with components of a componentlist.

---

skycatalog.mergecolumns - Function

Package display
Module viewer
Tool skycatalog

Merge n columns into one

Synopsis
mergecolumns(columns, name)

Arguments

columns	*Names of the columns to merge* *Allowed: String*
name	*name of the output column* *Allowed: String* *Default: unset*

Returns

Description
If the name argument is unset, the name of the output column is taken from the first column in column array. The output column will always be of type string.

---

skycatalog.name - Function

Package	**display**
Module	**viewer**
Tool	**skycatalog**

Return the name of the table

Synopsis
name()

Returns
String

Description

This function returns the name of the table.

---

skycatalog.renamecolumn - Function

Package	**display**
Module	**viewer**
Tool	**skycatalog**

Rename a column

Synopsis
renamecolumn(oldname, newname)

Arguments

oldname	*column name* *Allowed: String*
newname	*new column name* *Allowed: String*

Returns

Description

---

skycatalog.table - Function

Package display
Module viewer
Tool skycatalog

Return the underling table `tool`

Synopsis
table()

Returns
table `tool`

Description

---

skycatalog.type - Function

Package display
Module viewer
Tool skycatalog

Return the `tool` *type*

Synopsis
type()

Returns
skycatalog

Description

1.3.12 is_viewer - Function

Package display
Module viewer

Is the argument a viewer tool?

Synopsis
is_viewer(tool)

Description
This global function can be used to determine if the supplied argument is a viewer tool. If so it returns True, otherwise it returns False.

Arguments

tool	*The variable that you wish to test* *Allowed: Any glish variable* *Default: no default*

Returns
T or F, indicating if the argument is a viewer tool or not.

Example

```
include 'viewer.g';
if (is_viewer(dv)) {
  myviewer := dv;
} else {
  myviewer := viewer();
}
```

This example shows how you can use this function to determine if the default viewer (dv) is a viewer tool.

1.4 widgets - Module

Generic compound widgets based on Tk
include "widgetserver.g"

Description

The `widgets` *module contains GUI widgets built from basic Tk widgets. These widgets should be of general use to constructors of GUIs.*

Tools

actionoptionmenu	*Tk widget where the action menu label changes with selection*
autocli	*Tool to build a CLI from a record*
autogui	*Tool to build and operate a graphical form interface*
blinklabel	*Glish/Tk label which can be made to blink*
checkmenu	*Tk widget to provide a flat check button menu*
combobox	*A widget for selecting one or more items from a list.*
deleteregions	*Tk widget to delete image regions from a Table*
dialogbox	*Tk widget to get input from the user*
extendoptionmenu	*Tk widget where the (interactively extendable) menu label changes with selection*
fontchooserwidget	*AIPS++widget to help users select a font*
gchooser	*Tool for generating widgets for choosing items interactively*
gopher	*Tool for getting useful information*
guientry	*Tool for generating widgets for type-in of a GUI*
helpmenu	*Tk widget for a Help menu with the standard AIPS++ help items.*
histogramgui	*Window to display a histogram and allow selection of data min and max.*
listboxcontainer	*A scrolllistbox with an underlying itemcontainer.*
menuframes	*Tk widget to hide and expose a number of embedded frames on demand*
messageline	*Tk widget to display and log one-line messages.*
multiautogui	*Tool to manage a number of autoguis*
multiscale	*Glish/Tk widget to specify multiple values in a single domain*
optionmenu	*Tk widget where the menu label changes with selection*
pixelrange	*Widget to handle a pixel range specification*
popupselectmenu	*Tk widget to add a popup select menu to another widget*
profilewidthentry	*Widget to handle a 1-D profile width specification*
radiobuttons	*Widget handling multiple selectable radio buttons*
recordbrowserwidget	*AIPS++ widget to view Glish records*
restoreregions	*Tk widget to restore image regions of interest from a table*
rollup	*Tk widget to hide and expose an embedded frame on demand*
saveregions	*Tk widget to save image regions to a Table*
scrolllistbox	*A Tk listbox with optional scrollbars.*
scrolltext	*A Tk text widget with optional scrollbars.*
selectablelist	*Tk widget for selection from a menu or a listbox depending on list length*
synclistboxes	*A collection of synchronized scrolllistboxes.*
tabdialog	*Tk widget where the menu label changes with selection*

tapedeck	*Glish/Tk widget providing tapedeck-like control buttons*
taqlwidget	*Widget to form a TaQL command*
twodactionoptionmenu	*Pop-up, 'two-dimensional' menu*
widgetserver	*Tool to provide Tk widgets and AIPS++ megawidgets*

Functions

autoguitest	*Test the autogui*
gchoosertest	*Test the gchooser widgets*
guientrytest	*Test the construction of guientry widgets*
is_widgetserver	*Is the argument a widgetserver tool?*
tabdialogtest	*Test tabdialog widget*

1.4.1 actionoptionmenu - Tool

Package display
Module widgets

Tk widget where the action menu label changes with selection
include "optionmenu.g"

Description

This Tk widget is identical to the optionmenu widget, where you should look for function and event documentation (we give the constructor only below, with an example to show the intended usage). The `actionoptionmenu` *exists only so you can use it for "action" menus, where the relief (usually raised) and colors will in general be different from that of a "selection" option menu (relief is usually groove).*
The `actionoptionmenu` *is implemented on top of the* `optionmenu` *in the widgetserver tool. This means that it is only available via the* `widgetserver`*. Therefore you must do something like*

Example

```
- include 'widgetserver.g'
- f := dws.frame();
- m : = dws.actionoptionmenu(f);
```

You will have no success if you try to do

Example

```
- include 'actionoptionmenu.g'
- f := frame();
- m : = actionoptionmenu(f);
```

as the file `actionoptionmenu.g` *does not exist.*

actionoptionmenu - Constructor

Package	**display**
Module	**widgets**
Tool	**actionoptionmenu**

Construct an actionoptionmenu

Synopsis

actionoptionmenu(parent, labels, names, values, hlp, hlp2, nbreak, padx, pady, width, updatelabel, height, justify, font, relief, borderwidth, foreground, background, disabled, anchor, fill, widgetset)

Arguments

parent	*Parent of the menu* *Allowed: Widget*
labels	*The menu item labels* *Allowed: Vector of strings* *Default: Empty menu*
names	*The menu item names* *Allowed: Vector of strings* *Default: Empty menu*
values	*The menu item values* *Allowed: Vector of anything* *Default: Empty menu*
hlp	*Short popuphelp text* *Allowed: String* *Default: No help*
hlp2	*Long popuphelp text* *Allowed: String* *Default: No help*
nbreak	*The break point for menu or listbox usage* *Allowed: Integer* *Default: 20*
padx	*Horizontal padding around text* *Allowed: Integer* *Default: 7*
pady	*Vertical padding around text* *Allowed: Integer* *Default: 3*
width	*Width in character units* *Allowed: Integer* *Default: Auto-scale*
updatelabel	*Update menu label with selected item* *Allowed: T or F* *Default: T*
height	*Height in character units* *Allowed: Integer* *Default: 1*
justify	*Justification of text* *Allowed: String from 'center', 'left', 'right'* *Default: 'center'*
font	*Font of text* *Allowed: X font*
relief	*Edge relief* *Allowed: String from 'flat', 'ridge', 'raised', 'sunken', 'groove'* *Default: 'groove'*
borderwidth	*Border width* *Allowed: Integer* *Default: 2*
foreground	*Color of text* *Allowed: X color* *Default: 'black'*
background	*Background color* *Allowed: X color* *Default: 'lightgrey'*
disabled	*Is inactivated ?* *Allowed: Boolean*

Returns
Agent

Description

Example

```
- include 'widgetserver.g'
- names := ['Delete all files', 'Delete images', 'Delete measurement sets'];
- labels := "DelAll DelIm DelMS";
- f := dws.frame();
- hlp := 'Delete selected files';
- m := dws.actionoptionmenu(f, labels, names, hlp=hlp);
```

1.4.2 autocli - Tool

Package display
Module widgets

Tool to build a CLI from a record
include "autocli.g"

Description
This tool constructs a CLI corresponding to the parameters in the input record `params`. *The CLI is a subsequence that has a number of functions and events to allow interaction with the CLI. The CLI is designed to use various of the input widgets in the clientry. A GUI equivalent autocli is also available, and the common part of the documentation is to be found there.*

Constructors

autogui	*Constuct an autogui*

Functions

dismiss	*Dismiss the autocli*
get	*Get a record of the parameters*
help	*Show help*
loop	*Loop over inputs executing callbacks*

setcallbacks *Set the callbacks to be used in loop*

autogui - Constructor

Package	**display**
Module	**widgets**
Tool	**autocli**

Constuct an autogui

Synopsis
autogui(params, title, cliset)

Arguments

params	*Record to define the autocli* *Allowed: Record*
title	*Title* *Allowed: String* *Default: 'autocli'*
cliset	*clientry to use* *Allowed: clientry tool* *Default: dce*

Returns
subsequence

---

autocli.dismiss - Function

Package	**display**
Module	**widgets**
Tool	**autocli**

Dismiss the autocli

Synopsis
dismiss()

Description

Also available as an event.

---

autocli.loop - Function

Package	**display**
Module	**widgets**
Tool	**autocli**

Loop over inputs executing callbacks

Synopsis
loop()

Description
This function provides the prime way of using the autocli. The Glish function readline is used to read command lines from the user. A certain number of intrinsic arguments are recognized:

`inp` *Show the current inputs*

`help` *Show help for the current tool or function*

`quit` **or** `q` *Quit (without saving inputs)*

Thereafter the callbacks are invoked.

---

autocli.setcallbacks - Function

Package	**display**
Module	**widgets**
Tool	**autocli**

Set the callbacks to be used in loop

Synopsis
setcallbacks()

Description
The call backs should be presented in a record with the names of the function as the field names. Two arguments are passed: a record of the values of the inputs, and the remainder of the command line.

---

autocli.help - Function

Package	**display**
Module	**widgets**
Tool	**autocli**

Show help

Synopsis
help()

Description
This function shows the help attached to the methods and arguments.

---

autocli.get - Function

Package	**display**
Module	**widgets**
Tool	**autocli**

Get a record of the parameters

Synopsis
get()

Description
This returns a record containing the current parameters. WYSIWYG applies. The corresponding event is setoptions.

1.4.3 autogui - Tool

Package	**display**
Module	**widgets**

Tool to build and operate a graphical form interface

include "autogui.g"

Description

This `tool` *is an extremely useful one which builds a "form-like" graphical user interface (GUI) from a simple record whose fields describe individual parameters which the user may be allowed to edit. The GUI is displayed with the various parameters arranged vertically. Some of the parameters may be stored in rollup widgets. Each parameter is displayed with a text label, a widget (from the widgets module) to display the current value and optionally allow the user to edit it, a "spanner" menu (see guientry), and optionally an "auto-apply" check-button. Generally, an autogui* `tool` *will be used to show the state of some "parent"* `tool` *to the user, and/or allow them to edit the state. Consequently, the autogui* `tool` *is a* subsequence*, allowing the* `tool` *to emit events, and catch inbound events. These events can be used to ensure that the GUI and any parent* `tool` *remain synchronised as the user modifies widgets in the GUI. Furthermore, if the autogui is used in this way, multiple (duplicate) autogui* `tools` *can be used to control one and the same underlying parent. This is done, for example, in the viewerdisplaydata* `tool`*. An obvious use of this capability would be to display identical autogui* `tools` *on separate desktops (or even screens!) for a given* `tool`*. In addition to the event interface, the autogui* `tool` *provides a small number of functions to control its behaviour.*

Parameter description

Most transactions of the autogui `tool` *involve a "parameter record." This is a* Glish *record containing one or more top-level fields. In* full *form, each top-level field contains a* Glish *record which completely describes a particular parameter in terms of its name, parameter type, default value, current value, its possible values (for 'choice' or 'range' parameters), whether it can be 'unset', whether it can be 'auto-apply' and whether it is editable by the user. In* short *form, each top-level field simply contains the value for its parameter.*
The standard fields in a (full) single parameter record are:

- `dlformat`: *a string giving the name of the field of the parameter record in which this particular parameter is found.*
- `listname`: *a string giving the name of this parameter which should be displayed in the GUI to identify it.*
- `ptype`: *a string giving the type of the widget to be used for this parameter. If this field is not included, no widget will be created. Leaving this field blank will create a default entry box.*
- `value`: *the present value of this parameter.*
- `default`: *the default value of this parameter.*

The optional fields in a (full) single parameter record are:

- `context`: *if present, this parameter will be placed in a rollup widget having as its name the string contained in this field of the record.*
- `autoapply`: *if the autogui* `tool` *was constructed with* `autoapply=T` *[default], then a check-button will be placed next to this parameter. When this button is checked, any modification to this parameter will be immediately emitted from the autogui* `tool` *in a* `setoptions` *event (see below). This optional field, if present, indicates the initial state of this auto-apply button.*
- `dir`: *if present, and is* `'out'`*, then this parameter is considered to be output-only, so the user will be unable to edit it.*
- `allowunset`: *if present, and is* `T`*, then the constructed widget will allow this parameter to be 'unset'.*
- `dependency_group`: *if present, this specifies a tag which identifies this as a parameter which has a dependency on other parameters with the same tag. At present, all this means is that parameters with the same* `dependency_group` *tag must be mutually exclusive.*

We now list the available parameter types (`ptype`*), and their additional required record fields:*

- `boolean`: *a boolean guientry widget is provided with two choices:* `T` *or* `F`*. No additional required fields.*
- `choice`: *a choice guientry widget is provided for selection from a set list of items. Additional required record fields are:*
  - `popt`: *a vector of strings which are the valid options.*
- `directory`: *a file guientry widget is provided for selecting a directory. No additional required fields.*
- `file`: *a file guientry widget is provided for selecting a file. No additional required fields.*
- `floatrange`: *a range guientry widget is provided for selecting a single float or double from a range. Additional required record fields are:*
  - `pmin`: *minimum allowed real value*
  - `pmax`: *maximum allowed real value*
  - `presolution`: *resolution of provided scale widget*

  Additional optional record fields are:

  - `provideentry`: *if present, and is* `T`*, then an entry widget will be provided alongside the scale widget to allow the user to type a specific value. In this case, the scale will be "extendable" so that values outside the initial acceptable range will cause the scale to resize itself, and the allowed minimum and maximum to expand to accomodate the new specific value.*

- `intrange`: *a range guientry widget widget is provided for selecting a single integer from a given range. Additional required record fields are:*

  - `pmin`: *minimum allowed integer value*
  - `pmax`: *maximum allowed integer value*

  Additional optional record fields are:

  - `provideentry`: *if present, and is* `T`*, then an entry widget will be provided alongside the scale widget to allow the user to type a specific value. In this case, the scale will be "extendable" so that values outside the initial acceptable range will cause the scale to resize itself, and the allowed minimum and maximum to expand to accomodate the new specific value.*

- `measure`: *a measure widget is provided for selecting an AIPS++ measure. No additional required options.*

- `quantity`: *a quantity guientry widget widget is provided for selecting an AIPS++ quantity. No additional required fields.*

- `region`: *a region guientry widget is provided for selection an AIPS++ region. No additional required fields.*

- `scalar`: *a scalar guientry widget is provided for entering a scalar (integer, double, or complex). No additional required fields.*

- `table`: *a file guientry widget is provided for selecting an AIPS++ table. No additional required fields.*

- `userchoice`: *an extendoptionmenu is used for selection from a user-extendable list of items. Additional required record fields are:*

  - `popt`: *a vector of strings which are the initial valid options.*

- `minmaxhist`: *a minmaxhist guientry widget is provided for selection of minimum and maximum in a graphical way. Additional required record fields are:*

  - `minvalue` *The lower limit of the range.*
  - `maxvalue` *The upper limit of the range.*

  Additional optional record fields are:

– `default` *The default range.*

– `histarray` *The array containing a histogram to draw.*

– `imageunits` *The brightness units used as labels.*

Event description

Two important events are emitted by the autogui `tool`*, to which well-written* Glish *code will respond! The* `setoptions` *event is emitted whenever an "auto-apply" parameter is modified in the GUI. Its* `$value` *is a short-form parameter record, where the top-level fields simply contain the values of each parameter. It is also emitted when the action button is pressed, if the autogui* `tool` *was constructed without a parent frame (*`parent=F`*), and the* `actionlabel` *argument was a valid string.*
The `newuserchoicelist` *event is emitted whenever the extendoptionmenu widget for a parameter of* `ptype='userchoice'` *was extended by the user. This simply allows the programmer to detect when this happens, and modify their own local copy of the parameter set accordingly. This might be useful, for example, when they will create another autogui* `tool` *later on, and would like it to contain the additions the user made to any extendoptionmenus. The* `$value` *of this event has two fields:*

- `param:` *the internal name of the parameter which whose option list was extended (ie.* `dlformat`*)*
- `newvalue:` *the new value added to the choice list*

Example

In the example below, we build a simple autogui `tool` *with three parameters.*

```
include 'autogui.g'

# we will put a "parameter set" into "parameters":
parameters := [=];

# an example of the floatrange parameter type.  Its auto-apply
# button will be unchecked, and this parameter cannot be 'unset':
parameters.power := [=];
parameters.power.dlformat    := 'power';
parameters.power.listname    := 'Scaling power';
parameters.power.ptype       := 'floatrange';
parameters.power.pmin        := -5.0;
parameters.power.pmax        := +5.0;
parameters.power.presolution := 0.1;
parameters.power.default     := 0.0;
parameters.power.value       := 1.5;
```

```
parameters.power.autoapply   := F;
parameters.power.allowunset  := F;

# an example of the userchoice parameter type.  Its auto-apply
# button will be checked, and it also cannot be 'unset':
parameters.color := [=];
parameters.color.dlformat   := 'color';
parameters.color.listname   := 'Line color';
parameters.color.ptype      := 'userchoice';
parameters.color.popt       := "black white red yellow green";
parameters.color.default    := 'black';
parameters.color.value      := 'black';
parameters.color.autoapply  := T;
parameters.color.allowunset := F;

# an example of the region parameter type.  Its auto-apply
# button will be checked, and it CAN be 'unset':
parameters.region := [=];
parameters.region.dlformat   := 'region';
parameters.region.listname   := 'Displayed region';
parameters.region.ptype      := 'region';
parameters.region.default    := unset;
parameters.region.value      := unset;
parameters.region.autoapply  := T;
parameters.region.allowunset := T;

# construct an autogui.  We will put auto-apply=T since some of
# our parameters are auto-apply, and ask for a button with label
# 'Go!'.

myautogui := autogui(parameters, 'Demo autogui', actionlabel='Go!',
                     autoapply=T);

# and respond to the events it emits:
whenever myautogui->setoptions do {
  # do something more interesting than this though!
  print "New options for ", field_names($value), "emitted ...";
}
```

If you run this code, you should see a GUI like that shown in 1.10. Modifying either the 'Displayed region' or 'Line color' widgets should result in information being printed at the command line by the `whenever` *statement at the end of the example. Likewise, pressing the 'Go!' button will emit* all *parameters.*

autogui - Constructor

Package	**display**
Module	**widgets**
Tool	**autogui**

Create an autogui `tool`

Synopsis

autogui(params, title, toplevel, map, actionlabel, autoapply, expand, widgetset)

Arguments

params	*Record to define the autogui* *Allowed: Record*
title	*title string if top-level frame constructed internally* *Allowed: String* *Default: 'autogui'*
toplevel	*top-level parent frame* *Allowed: frame or boolean* *Default: F*
map	*map the GUI to the screen immediately?* *Allowed: boolean* *Default: T*
actionlabel	*label for action button* *Allowed: string or boolean* *Default: F*
autoapply	*Show auto-apply buttons* *Allowed: boolean* *Default: T*
expand	*Expand policy* *Allowed: 'x'—'y'—'both'—'none'* *Default: 'none'*
widgetset	*widgetserver* `tool` *to use to build the GUI* *Allowed: widgetserver* `tool` *Default: dws*

Returns

subsequence

Description

This is the only constructor for the autogui `tool`. *The first argument,* `params`, *contains the various parameters which will be used to generate the GUI. If a parent frame is not given in the* `toplevel` *argument, a top-level frame will be created with title as given in the* `title` *argument. If the* `actionlabel` *argument is given as a string, then a button will be placed at the bottom of the GUI with that string as its label. Pressing this button will cause the autogui* `tool` *to emit all parameters and their values via a* `setoptions` *event. If the* `map` *argument is* `F`, *then the GUI will be constructed but not shown to the user immediately, but can be shown via a call to autogui.map. The same effect can be controlled more carefully if a presently unmapped parent frame is given to* `toplevel`. *The final two arguments just provide some future flexibility with respect to widgetserver and guientry* `tools`.

---

autogui.buttonbar - Function

Package	**display**
Module	**widgets**
Tool	**autogui**

Return the frame in which buttons are placed

Synopsis
buttonbar()

Returns
frame

Description

This function returns a handle to the frame in which the autogui `tool` *places any buttons, such as the action button and the dismiss button. This is provided so that programmers can add their own buttons in a simple way. A more flexible way to do this is to provide a parent frame to the autogui* `tool` *constructor, and build your own GUI in the same frame.* Note that this function is only available if a top-level frame was *not* given during construction.

Example

Assuming you have already created an autogui `tool` *in the variable* **myautogui**, *perhaps as per the above example, you may do something like this to place a "Free Beer!" button near the "Go!" button:*

```
# ...
myf := myautogui.buttonbar();
myb := dws.button(myf, 'Free Beer!', type='action');
whenever myb->press do {
  # ...
}
# ...
```

---

autogui.get - Function

Package	**display**
Module	**widgets**
Tool	**autogui**

Read the GUI and return the current settings

Synopsis
get()

Returns
short-form parameter record

Description

This function reads the GUI, extracting all *parameter values, and returns the result in a* short-form *parameter record. In this form, the top-level fields simply contain the values corresponding to the parameters having the top-level field names as their* `dlformat` *parameter bit. When this function is called, a* `setoptions` *event is also emitted. This assists in keeping multiple clients aware of the state of the autogui* `tool`.

Example

Assuming you have already created an autogui `tool` *in the variable* **myautogui**, *perhaps as per the above example, you may do something like this to read the current values into a record:*

```
# ...
currvalues := myautogui.get();
# ...
```

Package	**display**
Module	**widgets**
Tool	**autogui**

Map the GUI to the screen

Synopsis
map()

Returns
T

Description

This function is only useful if at construction, the `map` *argument was forced to be* `F`*. In this case, you can use this function to map the GUI to the screen at some later stage.*

---

autogui.done - Function

Package	**display**
Module	**widgets**
Tool	**autogui**

Finish using this autogui `tool`

Synopsis
done()

Returns
T

Description

This function should be used when you have finished using a particular autogui `tool`*. It unmaps the GUI from the screen, stops it emitting and responding to events, and releases any resources being used by the autogui.*

Example

Assuming you have already created an autogui `tool` *in the variable* **myautogui**, *perhaps as per the above example, you may finish using it by doing:*

```
# ...
myautogui.done();
# ...
```

---

autogui.fillgui - Function

Package	**display**
Module	**widgets**
Tool	**autogui**

Fill the autogui `tool` *with updated values*

Synopsis
fillgui(wparams, what)

Arguments

wparams	*new parameter values* *Allowed: full-form parameter record*
what	*which sub-field to extract new values from* *Allowed: 'value' or 'default'* *Default: 'value'*

Returns
T

Description

This function is mostly used internally to construct and fill the GUI. However, it can be used externally to refresh the GUI to a new set of parameters.

Example

For example, suppose you want to add a "Restore defaults" button to your autogui. When the user presses it, then the autogui `tool` *will be told to re-fill the GUI with* `what='default'`.

```
# ... myautogui setup already ...
myf := myautogui.buttonbar();
myb := dws.button(myf, 'Restore defaults');
whenever myb->press do {
  myautogui.fillgui(parameters, 'default');
}
# ...
```

Of course, we should also update the `value` *fields of the parameters to be the same as the* `default` *fields, but that is left to the reader...*

---

autogui.modifygui - Function

Package	**display**
Module	**widgets**
Tool	**autogui**

Modify the GUI parameter set

Synopsis
modifygui(wparams)

Arguments

wparams	*new parameters* *Allowed: full-form parameter record*

Returns
T

Description

This function is not completely implemented yet. *It is provided so that an existing autogui* `tool` *can be modified in terms of the names of its parameters, and the parameter types etc. Presently, only fields of* `ptype` *'intrange', 'floatrange' and 'minmaxhist' are noticed by this function. Their labels are updated according to the* `listname` *subfield, their ranges according to the* `pmin` *and* `pmax` *subfields, and in the case of 'minmaxhist', the data being displayed can be changed via the* `histarray` *subfield.*

1.4.4 autoguitest - Function

Package display
Module widgets

Test the autogui

Synopsis
autoguitest()

Returns
T

Description

This function provides a simple test of all widgets. A frame is contructed with all possible entry widgets. The frame has button for emitting the current values (Apply), Enabling and Disabling the widgets, and Clearing the values. If a new widget is introduced into autogui, it should be represented here in the test.

1.4.5 blinklabel - Tool

Package display
Module widgets

Glish/Tk label which can be made to blink
include "blinklabel.g"

Constructors

blinklabel	*Construct a blinklabel*

Functions

blink	*Switch blinking on or off*
done	*Destroy ourselves*

Description

This Glish/Tk widget simply provides a basic Glish/Tk label that can be made to blink. That is, on request, it will swap its text and background colors at a regular interval.

blinklabel - Constructor

Package	**display**
Module	**widgets**
Tool	**blinklabel**

Construct a blinklabel

Synopsis

blinklabel(parent, text, justify, hlp, padx, pady, font, width, relief, borderwidth, foreground, background, anchor, fill, blink, interval, widgetset)

Arguments

parent	*Parent of the label* *Allowed: Tk frame*
text	*label text* *Allowed: string* *Default: 'label'*
justify	*justification of text* *Allowed: 'center' 'left' 'right'* *Default: 'left'*
hlp	*text for popuphelp* *Allowed: string* *Default: ''*
padx	*horizontal padding* *Allowed: dimension* *Default: 4*
pady	*vertical padding* *Allowed: dimension* *Default: 2*
font	*text font* *Allowed: X font* *Default: ''*
width	*width in character units* *Allowed: integer* *Default: 0*
relief	*border relief* *Allowed: 'flat' 'ridge', 'raised' 'sunken' 'groove'* *Default: 'flat'*
borderwidth	*border width* *Allowed: dimension* *Default: 2*
foreground	*color of text* *Allowed: X color* *Default: 'black'*
background	*background color* *Allowed: X color* *Default: 'lightgrey'*
anchor	*location of text* *Allowed: 'c' 'n' 's' 'e' 'w' 'ne' 'nw' 'se' 'sw'* *Default: 'c'*
fill	*how to expand when resized* *Allowed: 'x' 'y' 'both' 'none* *Default: 'none'*
blink	*whether to blink immediately* *Allowed: T F* *Default: F*
interval	*blink rate in seconds* *Allowed: float* *Default: 1*
widgetset	*widgetserver to use to make label* *Allowed: widgetserver object* *Default: dws*

Returns
blinklabel widget

Description
Constructs a blinklabel according to the specified construction arguments, and returns a ref to the widget.

Example

```
include 'blinklabel.g'
f := frame();
bl := blinklabel(f, 'flashdance');
```

---

blinklabel.blink - Function

Package	**display**
Module	**widgets**
Tool	**blinklabel**

Switch blinking on or off

Synopsis
blink(on, interval)

Arguments

on	*blinking on (T) or off (F)* *Allowed: T F* *Default: T*
interval	*blink interval in seconds* *Allowed: float* *Default: 1*

Returns
none

Description
This function is used to switch blinking on or off, and optionally specify the blink interval in seconds.

Example

```
bl := blinklabel(spareframe, 'flashdance');
bl.blink(T, 0.2); # blink every 0.2 seconds
# ... some time passes ...
bl.blink(F);      # stop blinking
```

---

blinklabel.done - Function

Package	**display**
Module	**widgets**
Tool	**blinklabel**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This tool function destroys the widget.

Example

```
- bl := blinklabel(spareframe, 'flashdance');
- bl.done();
T
- bl
F
-
```

1.4.6 checkmenu - Tool

Package	**display**
Module	**widgets**

Tk widget to provide a flat check button menu
include "checkmenu.g"

Constructors

checkmenu	*Construct a checkmenu*

Functions

disabled	*Disable or enable the checkmenu widget*
extend	*Extend the menu with more items*
findname	*Find the index of the item with the given name*
findvalue	*Find the index of the item with the given value*
getlabel	*Get the label of the menu button*
getnames	*Gets all of the names associated with the menu*
getoffstates	*Get the indices of the items which are not checked*
getonstates	*Get the indices of the items which are checked*
getstate	*Get the state of the specified check button*
getstates	*Get the states of all of the check buttons*
getvalues	*Gets all of the values associated with the menu*
isenabled	*Is the checkmenu widget enabled ?*
replace	*Replace the menu with the provided items*
reset	*Resets the state of all check buttons to F*
selectindex	*Select an item in the menu by index*
selectname	*Select an item in the menu by name*
selectvalue	*Select an item in the menu by value*
setbackground	*Set the color of the background of the checkmenu button*
setforeground	*Set the color of the foreground of the checkmenu button*
setlabel	*Set the label of the checkmenu*

Description

This Tk widget enables you to create a flat (i.e. no submenus) menu in a specified frame where all the menu items are check buttons. You can associate a name (that which appears in the menu list) and a value (whatever you like, it may be different from the name) for each menu item.
It can be treated like any of the built in Tk widgets, and has almost all of the same construction arguments as the button Tk widget.
Whenever a selection is made, this widget emits an event called `select`*. In the $value of this event, you can find a record with fields* `name`, `value`, `state` *(T or F) and* `index` *(index of item in menu) containing that information for the selected item.*
Whenever the menu is replaced with the replace tool function, this widget emits an event called `replaced` *(it has no value). Note that the tool function extend invokes the* `replace` *function, so a* `replaced` *event is generated then as well.*

checkmenu - Constructor

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Construct a checkmenu

Synopsis

checkmenu(parent, label, names, values, hlp, hlp2, padx, pady, width, height, justify, font, relief, borderwidth, foreground, background, disabled, anchor, fill, widgetset)

Arguments

parent	*Parent of the menu* *Allowed: Widget*
label	*The label of the menu* *Allowed: String* *Default: "checkmenu"*
names	*The menu item names* *Allowed: Vector of strings* *Default: Empty menu*
values	*The menu item values* *Allowed: Vector of anything* *Default: Empty menu*
hlp	*Short popuphelp text* *Allowed: String* *Default: No help*
hlp2	*Long popuphelp text* *Allowed: String* *Default: No help*
padx	*Horizontal padding around text* *Allowed: Integer* *Default: 7*
pady	*Vertical padding around text* *Allowed: Integer* *Default: 3*
width	*Width in character units* *Allowed: Integer* *Default: 0 [Auto-scale]*
height	*Height in character units* *Allowed: Integer* *Default: 1*
justify	*Justification of text* *Allowed: String from 'center', 'left', 'right'* *Default: 'center'*
font	*Font of text* *Allowed: X font*
relief	*Edge relief* *Allowed: String from 'flat', 'ridge', 'raised', 'sunken', 'groove'* *Default: 'groove'*
borderwidth	*Border width* *Allowed: Integer* *Default: 2*
foreground	*Color of text* *Allowed: X color* *Default: 'black'*
background	*Background color* *Allowed: X color* *Default: 'lightgrey'*
disabled	*Is inactivated ?* *Allowed: Boolean* *Default: F*
anchor	*Location of text* *Allowed: String from 'c', 'n', 's', 'e', 'w', 'ne', 'nw', 'se', 'sw'* *Default: 'c'*
fill	*How to expand when resized*

Returns
Agent

Description

Example

```
- include 'checkmenu.g'
- f := frame();
- hlp := 'Select type of model to fit';
- m := checkmenu(f, 'Models', "gaussian disk point", [1,2,3], hlp);
```

The values vector, if not given, defaults to the names vector.

---

checkmenu.extend - Function

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Extend the menu with more items

Synopsis
extend(names, values)

Arguments

names	*The menu item names* *Allowed: Vector of strings* *Default: Empty menu*
values	*The menu item values* *Allowed: Vector of anything* *Default: Empty menu*

Returns
Bool

Description

This tool function extends the menu with the supplied items.

Example

```
- f := frame();
- m := checkmenu(f, 'Models', "gaussian point");
- m.getnames();
gaussian point
- m.extend(names="disk")
- m.getnames();
gaussian point disk
-
```

---

checkmenu.replace - Function

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Replace the menu with the provided items

Synopsis

replace(label, names, values, width)

Arguments

label	*The label of the menu* *Allowed: String* *Default: "checkmenu"*
names	*The menu item names* *Allowed: Vector of strings* *Default: Empty menu*
values	*The menu item values* *Allowed: Vector of anything* *Default: Empty menu*
width	*Width in character units* *Allowed: Integer* *Default: 0 [Auto-scale]*

Returns

Bool

Description

This tool function replaces the menu with the supplied items. If you set `width=-1` *then the menu button width will be left as it is. Thus if you constructed with* `width=0`*, the button will resize itself to accomodate the new text. If you used a specific non-zero width at construction time, setting* `width=-1` *now will leave the button at that width. Setting* `width=0` *will now force a resize and setting* `width` *to some positive integer will resize to that width.*

Example

```
- f := frame();
- m := checkmenu(f, 'Models', "gaussian point", [1,2])
- m.getnames();
gaussian point
- m.getvalues();
[1 2]
- m.replace('Animals', "cat dog fish");
- m.getnames();
cat dog fish
- m.getvalues();
cat dog fish
-
```

---

checkmenu.selectindex - Function

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Select an item in the menu by index

Synopsis

selectindex(idx, state)

Arguments

idx	*The menu item index* *Allowed: Integer*
state	*The state for the check box* *Allowed: T of F* *Default: Opposite state*

Returns
Bool

Description

This tool function selects a menu item by its index. It is equivalent to selecting the item with the GUI interface. If the index is invalid, it returns `F`*. If you don't specify the* `state` *variable, the check button is given the opposite state to its current state.*

Example

```
- f := frame();
- m := checkmenu(f, 'Models', "gaussian point");
- m.getstates()
[F F]
- m.selectindex(2);
- m.getstates()
[F T]
- m.selectindex(2, T);
[F T]
-
```

---

checkmenu.selectname - Function

Package display
Module widgets
Tool checkmenu

Select an item in the menu by name

Synopsis
selectname(name, state)

Arguments

name	*The menu item name* *Allowed: String*
state	*The state for the check box* *Allowed: T of F* *Default: Opposite state*

Returns
Bool

Description

This tool function selects a menu item by its name. It is equivalent to selecting the item with the GUI interface. If the name is invalid, it returns `F`*. If you don't specify the* `state` *variable, the check button is given the opposite state to its current state.*

Example

```
- f := frame();
- m := checkmenu(f, 'Models', "gaussian point");
- m.getstates()
[F F]
- m.selectname('gaussian');
- m.getstates()
[T F]
- m.selectname('gaussian', T);
[T F]
-
```

---

checkmenu.selectvalue - Function

Package display
Module widgets
Tool checkmenu

Select an item in the menu by value

Synopsis

selectvalue(value, state)

Arguments

value	*The menu item value* *Allowed: Same type as at construction*
state	*The state for the check box* *Allowed: T of F* *Default: Opposite state*

Returns
Bool

Description

This tool function selects a menu item by its value. It is equivalent to selecting the item with the GUI interface. If the value is invalid, it returns `F`*. If you don't specify the* `state` *variable, the check button is given the opposite state to its current state.*

Example

```
- f := frame();
- m := checkmenu(f, 'Models', "gaussian point", [10,20])
- m.getstates()
[F F]
- m.selectvalue(10)
- m.getstates()
[T F]
- m.selectvalue(10, T);
[T F]
-
```

---

checkmenu.getlabel - Function

Package display
Module widgets
Tool checkmenu

Get the label of the menu button

Synopsis
getlabel()

Returns
String or F

Description

This tool function recovers the label of the menu button.

Example

```
- f := frame();
- m := checkmenu(f);
- m.getlabel();
checkmenu
```

---

checkmenu.getnames - Function

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Gets all of the names associated with the menu

Synopsis
getnames()

Returns
Vector of Strings

Description

This tool function recovers the names associated with each item in the menu. If the menu is empty, it returns an empty array.

Example

```
- f := frame();
- m := checkmenu(f, 'Models', "gaussian point", [1,2])
- m.getnames();
gaussian point
```

---

checkmenu.getvalues - Function

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Gets all of the values associated with the menu

Synopsis
getvalues()

Returns
The "values" array given (explicitly or implicitly) in the constructor

Description

This tool function recovers the values associated with each item in the menu. If the menu is empty, it returns an empty array.

Example

```
- f := frame();
- m := checkmenu(f, 'Models', "gaussian point", [1,2])
- m.getvalues()
[1 2]
-
```

---

checkmenu.getstate - Function

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Get the state of the specified check button

Synopsis
getstate(idx)

Arguments

idx	*The index of the menu item* *Allowed: Integer*

Returns
T or F or fail

Description

This tool function recovers the state of the specified check button. A fail is generated for an empty checkmenu or invalid index.

Example

```
- f := frame();
- m := checkmenu(f, 'Models', "gaussian disk");
- m.getstate(1);
F
```

---

checkmenu.getstates - Function

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Get the states of all of the check buttons

Synopsis
getstates()

Returns

T or F

Description

This tool function recovers the states of all of the check buttons.

Example

```
- f := frame();
- m := checkmenu(f, 'Models', "gaussian disk");
- m.getstates();
[F F]
```

---

checkmenu.getonstates - Function

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Get the indices of the items which are checked

Synopsis
getonstates()

Returns
Vector of integers

Description

This tool function recovers the indices of the check menu items which are checked.

Example

```
- f := frame();
- m := checkmenu(f, 'Models', "gaussian disk point");
- m.selectindex(1,T);
```

```
- m.selectindex(3,T);
- m.getstates();
[T F T]
- m.getonstates();
[1 3]
```

---

checkmenu.getoffstates - Function

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Get the indices of the items which are not checked

Synopsis
getoffstates()

Returns
Vector of integers

Description

This tool function recovers the indices of the check menu items which are not checked.

Example

```
- f := frame();
- m := checkmenu(f, 'Models', "gaussian disk point");
- m.selectindex(1,T);
- m.selectindex(3,T);
- m.getstates();
[T F T]
- m.getoffstates();
[2]
```

---

checkmenu.findname - Function

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Find the index of the item with the given name

Synopsis
findname(name)

Arguments

name	*The menu item name* *Allowed: String*

Returns
Integer or F

Description

This tool function find the index in the menu of the item specified by its name. If you give an invalid name, `F` *is returned.*

Example

```
- f := frame();
- m := checkmenu(f, 'Models', "gaussian point");
- m.findname('point');
2
```

---

checkmenu.findvalue - Function

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Find the index of the item with the given value

Synopsis
findvalue(value)

Arguments

value	*The menu item value* *Allowed: Same type as the array "value" used in construction*

Returns
Integer or F

Description

This tool function find the index in the menu of the item specified by its value. If you give an invalid value, `F` *is returned.*

Example

```
- f := frame();
- m := checkmenu(f, 'Models', "gaussian point", [10,20]);
- m.findvalue(20);
2
-
```

---

checkmenu.setlabel - Function

Package display
Module widgets
Tool checkmenu

Set the label of the checkmenu

Synopsis
setlabel(label, width)

Arguments

label	*The label* *Allowed: String*
width	*Width of button* *Allowed: Integer* *Default: -1*

Returns
Bool

Description

This tool function sets the label of the checkmenu. If you set `width=-1` *then the menu button width will be left as it is. Thus if you constructed with* `width=0`*, the button will resize itself to accomodate the new text. If you used a specific non-zero width at construction time, setting* `width=-1` *now will leave the button at that width. Setting* `width=0` *will now force a resize and setting* `width` *to some positive integer will resize to that width.*

Example

```
- f := frame();
- m := checkmenu(f, 'Animals', "dog cat");
- m.getlabel()
Animals
- m.setlabel('Small animals');
- m.getlabel()
Small animals
-
```

---

checkmenu.reset - Function

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Resets the state of all check buttons to F

Synopsis
reset()

Returns
Bool

Description

This tool function resets the state of all check buttons to F.

---

checkmenu.setforeground - Function

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Set the color of the foreground of the checkmenu button

Synopsis
setforeground(color)

Arguments

color	*The color* *Allowed: Any X color encoded as a String*

Returns
Bool

Description

This tool function sets the color of the foreground of the checkmenu button.

---

checkmenu.setbackground - Function

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Set the color of the background of the checkmenu button

Synopsis
setbackground(color)

Arguments

color	*The color* *Allowed: Any X color encoded as a String*

Returns
Bool

Description

This tool function sets the color of the background of the checkmenu button.

---

checkmenu.disabled - Function

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Disable or enable the checkmenu widget

Synopsis
disabled(disable)

Arguments

disable	*Disable the widget* *Allowed: T or F* *Default: T*

Returns
Bool

Description

This tool function disables or enables the checkmenu.

---

checkmenu.isenabled - Function

Package	**display**
Module	**widgets**
Tool	**checkmenu**

Is the checkmenu widget enabled ?

Synopsis
isenabled()

Returns
Bool

Description

This tool function indicated whether the checkmenu has been disabled or not.

Example

```
- f := frame();
- m := checkmenu(f, 'Animals', "dog cat");
- m.disable();
- m.isenabled()
F
-
```

1.4.7 combobox - Tool

Package **display**
Module **widgets**

A widget for selecting one or more items from a list.
include "combobox.g"

Description

The combobox combines a text entry widget with a popup listbox and a label to provide a handy GUI way to select one or more items from a list. The combobox has an internal data store – a list of strings – and a way to keep track of which of these strings is currently selected. You can think of the listbox as the visual representation of the data store, and the text entry widget as the visual representation of the current selection.

Items in the combobox data store are always strings, so the application must do whatever is necessary to interpret strings as numbers, or file names, or any other application-specific tool. The most common use of the combobox will be for the interactive selection of items by the user in the popup listbox, by using the mouse. The combobox immediately displays these selections in the text extry widget. Users may also edit and type directly into the text entry widget; a carriage return in the text entry widget usually causes the current contents of that widget to be added to the combobox data store (and the listbox) as a new item. The current contents of the text entry widget can be retrieved at any time by a "getentry ()" call. The current listbox selection may be retrieved at any time by a "get('selected')" call. The entry contents may not be the same as the listbox selection if, for example, the user has edited the entry contents but not yet hit return. The application can choose to retrieve the entry contents at any time and, if necessary, add items to the combobox (and optionally make one of these the current selection).

When the user presses the arrow button which causes the listbox to pop up, the listbox grabs the input focus. The user then usually has only two options: clicking the left mouse button on an item in the listbox, which selects that item and puts it into the combobox entry field, or clicking anywhere else which causes the listbox to be dismissed and nothing to happen. If the combobox has been set to display a clear button through the use of the canclearpopup attribute then the user can also click on the clear button. This clears all of the items from the internal data store as well as clearing the entry before dismissing the listbox. If the listbox has scrollbars, the user can manipulate those.

The combobox emits events through the agent returned by the agent member function. Standard events include "select", when the user makes a selection in the combobox or hits return in the entry, causing a new selection, "return", when the user hits return in the entry (which also generates a selection if addonreturn is T) and a "press", when the user presses the button which displays the listbox. The bind member function can be used to bind any x-event to this same agent.

One limitation in glishtk is that it is impossible to query the true size of an entry widget. The best one can do is query the requested size when the widget was constructed. This makes it impossible for the listbox popup to always have the same width as the entry widget in the combobox.

One further caveat is that the indexing into the internal data store through member functions such as get, insert, and delete, is 0-relative. This is to make it identical to the indexing in the glishtk listbox widget which the combobox uses. This means that the first item in the list is at index number 0 and if there are N items in the list the the last item in the list is at index number 'N1'. In addition to being able to index by integer, there are three string indices which are useful: 'active', 'selected', and 'end'. Of these, 'active' and 'end' also can be used in glishtk listbox indexing. The third, 'selected' was added as a natural synonym for 'active'.

The combobox emits events through the agent returned by the "agent" member function.

return *When the user hits return in the combobox entry. The value of this event is the contents of the entry box when the user hit return.*

select *When the user selects something from the popup list box or after a* **return** *if the combobox has addonreturn turned on (the default case).*

clear *When the user presses the "Clear" button on the popup.*

press *When the user presses the arrow button*

user defined *Through the use of the "bind" member function, the application can bind X-events to the entry widget. When a bound event occurs in the entry widget it is emitted through this agent with the name the user chose in the invocation of "bind".*

Example

```
f := frame()
cb := combobox(f,'Colors',"red blue green yellow amber saffron");
print cb.get('selected')
# the string ``red'' is printed out
cb.select(2)
print cb.get('selected')
# the string ``green'' is printed out due to 0-relative indexing
cb.insert('orange', 2)
a := cb.get(0,'end')
# a now contains "red blue orange green yellow amber saffron"
# The selection is maintained at green, which now has an index of 3
```

combobox - Constructor

Package	**display**
Module	**widgets**
Tool	**combobox**

Construct a combobox

Synopsis

combobox(parent, labeltext, items, addonreturn, entrydisabled, canclearpopup, autoinsertorder, disabled, borderwidth, exportselection, vscrollbar, hscrollbar, labelwidth, labelfont, labelrelief, labeljustify, labelbackground, labelforeground, labelanchor, entrywidth, entryfont, entryrelief, entryjustify, entrybackground, entryforeground, entryanchor, arrowbutton, arrowbackground, arrowforeground, listboxheight, help, widgetset)

Most of the defaulted arguments provide fine control over the appearance of the various parts of the combobox. Most users will never need to use these arguments. The arguments most likely be changed by users are placed early in the argument list.

Arguments

parent	*a frame to hold the combobox* *Allowed: glishtk frame*
labeltext	*a label for the combobox* *Allowed: string* *Default: 'label'*
items	*The initial contents of the combobox data. The first item will be selected. If F the contents will be empty* *Allowed: an array of strings* *Default: F*
addonreturn	*Should a return in the entry widget automatically add the entry value to the contents?* *Allowed: T or F* *Default: T*
entrydisabled	*When the entry is disabled, the user can not edit in the entry, the combobox otherwise functions normally.* *Allowed: T or F* *Default: F*
canclearpopup	*When this is true, the listbox comes with a "Clear" button which, when pressed, clears the contents.* *Allowed: T or F* *Default: F*
autoinsertorder	*Determines where each new item is inserted if no specific location is given.* *Allowed: 'tail','head'* *Default: 'tail'*
disabled	*Disable the combobox, turns off all graphical interaction.* *Allowed: T or F* *Default: F*
borderwidth	*The borderwidth of the outer combobox frame.* *Allowed: any integer* *Default: 2*
exportselection	*exportselection value for entry.* *Allowed: T or F* *Default: T*
vscrollbar	*The type of vertical scrollbar that the listbox has.* *Allowed: 'ondemand','always','none'* *Default: 'ondemand'*
hscrollbar	*The type of horizontal scrollbar that the listbox and the entry have.* *Allowed: 'ondemand','always','none'* *Default: 'none'*
labelwidth	*The width, in characters, of the label.* *Allowed: any integer* *Default: 0*
labelfont	*The font used in the label.* *Allowed: any X font.*
labelrelief	*The relief of the label.* *Allowed: any glishtk relief* *Default: 'flat'*
labeljustify	*The text justification of the label.* *Allowed: any glishtk text justificiation* *Default: 'left'*

Arguments

labelbackground	*The background color of the label.* *Allowed: any X color.* *Default: 'lightgrey'*
labelforeground	*The foreground color of the label.* *Allowed: any X color.* *Default: 'black'*
labelanchor	*The text anchor point within the label.* *Allowed: any glishtk anchor.* *Default: 'c'*
entrywidth	*The width, in characters, of the entry.* *Allowed: any integer* *Default: 30*
entryfont	*The font used in the entry.* *Allowed: any X font.*
entryrelief	*The relief of the entry.* *Allowed: any glishtk relief* *Default: 'sunken'*
entryjustify	*The text justification of the entry.* *Allowed: any glishtk text justificiation* *Default: 'left'*
entrybackground	*The background color of the entry.* *Allowed: any X color.* *Default: 'lightgrey'*
entryforeground	*The foreground color of the entry.* *Allowed: any X color.* *Default: 'black'*
entryanchor	*The text anchor point within the entry.* *Allowed: any glishtk anchor.* *Default: 'c'*
arrowbutton	*The file containing the bitmap to use for the arrow button.* *Allowed: any X bitmap file.* *Default: 'downarrow.xbm'*
arrowbackground	*The background color of the arrow button.* *Allowed: any X color.* *Default: 'lightgrey'*
arrowforeground	*The foreground color of the arrow button.* *Allowed: any X color.* *Default: 'black'*
listboxheight	*The height of the listbox when a horizontal scrollbar is present.* *Allowed: any integer* *Default: 6*
help	*Short popup help text* *Allowed: String*
widgetset	*Widget set to use* *Allowed: A widgetserver closure tool* *Default: dws*

Returns
a combobox tool with associated functions.

---

combobox.agent - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Get a reference to the agent which emits events from this combobox.

Synopsis
agent()

Returns
a glish agent

Description
This combobox agent emits a "return" when the user hits return in the entry, a "press" the user presses the arrow button, a "select" when then makes a selection in the listbox, and "clear" when the user presses the clear button on the listbox (which is only present if canclearpopup is T. A "select" event also follows a "return" event if addonreturn is T.

Example

```
f := frame()
cb := combobox(f,'colors')
whenever cb.agent()->* do {
    print $name, ' event seen, value is : ', $value
        }
```

---

combobox.pixelwidth - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

The true width, in pixels, of the outer combobox frame.

Synopsis
pixelwidth()

Returns
an integer

---

combobox.pixelheight - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

The true height, in pixels, of the outer combobox frame.

Synopsis
pixelheight()

Returns
an integer

---

combobox.get - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Get an item or range of items from the combobox.

Synopsis
get(first, last)

Arguments

first	*The first item to get* *Allowed: Any valid item number from 0 to the end plus 'selected' and 'end'*
last	*The last item to get, if F just get the first* *Allowed: Any index ¿= first or F* *Default: F*

Returns
A vector of strings

Description
As with the glishtk listbox, the index numbers here are 0-relative (i.e. the first item is item number 0). "selected" translates to the index of the current selection, "end" translates to the last item in the combobox. get() returns a fail if the index is out of range or otherwise does not exist (e.g. there is no current selection).

---

combobox.selection - Function

Package display
Module widgets
Tool combobox

Retrieve the index number of the current selection.

Synopsis
selection()

Returns
An integer

Description
Returns a -1 if there is no current selection.

---

combobox.select - Function

Package display
Module widgets
Tool combobox

Select the indicated item by index number and place it in the entry.

Synopsis
select(whichitem)

Arguments

whichitem	*The item to select* *Allowed: Any valid item number from 0 to the end plus 'selected' and 'end'*

Returns
T

Description
This returns a fail if there is no valid selection.

---

combobox.getentry - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Get the contents of the entry.

Synopsis
getentry()

Returns
A string

---

combobox.addonreturn - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Sets the behavior when the user presses return in the entry.

Synopsis
addonreturn(addonreturn)

Arguments

addonreturn	*The requested behavior* *Allowed: T or F*

Description
When the argument is T, a return automatically adds the contents of the entry to the combobox internal data, when it is F, nothing happens. Typically, when this attribute is F the application will wait for return events from the combobox and examine the value of the entry before deciding if it should be added to the internal combobox data or not. The default behavior is to always add values on return, or T.

Example

```
f := frame()
cb := combobox(f,'colors',"red blue green yellow amber saffron")
cb.addonreturn(F)
whenever cb.agent()->return do {
    newitem := $value
    if (newitem != cb.get('selected')) {
        # only add a new item if it isn't what is currently
        # selected
        cb.insert(newitem, select=T)
        # when added, insert it via current autoinsertorder
        # and select it
    }
}
```

---

combobox.canclearpopup - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Is there a clear button on the listbox popup.

Synopsis
canclearpopup(tOrF)

Arguments

tOrF	*Is there a clear button* *Allowed: T or F*

Description
If this is true, then a clear button is present next to the listbox popup which, when pressed, clears the contents of the combobox (deleting them all and clearing the entry). Otherwise, there is no graphical way to clear the contents of the combobox. The default behavior is to have a clear button, or F.

---

combobox.borderwidth - Function

Package display
Module widgets
Tool combobox

Change the borderwidth of the outer combobox frame.

Synopsis
borderwidth(newwidth)

Arguments

newwidth	*The new width* *Allowed: any positive integer.*

---

combobox.exportselection - Function

Package display
Module widgets
Tool combobox

Change the export selection property of the entry.

Synopsis
exportselection(tOrF)

Arguments

tOrF	*The new exportselection value* *Allowed: T or F.*

---

combobox.disabled - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Disable (T) or enable (F) the entire combobox.

Synopsis
disabled(tOrF)

Arguments

tOrF	*The new state of the combobox* *Allowed: T or F.*

Description
When the combobox is disabled, the user can not interact with it graphically although all of the functions work as always. The state of the entry is remember so that if the entry was disabled when the combobox was disabled, it will remain disabled when the combobox is enabled (via a F argument to this function).

---

combobox.delete - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Delete an item or range of items from the combobox.

Synopsis
delete(first, last)

Arguments

first	*The first item to delete* *Allowed: Any valid item number from 0 to the end plus 'selected' and 'end'*
last	*The last item to delete, if F just get the first* *Allowed: Any index ¿ = first or F* *Default: F*

Description
This deletes these items from the list. If the current selection falls within the specified range, the first item before the range is selected or the first item after the range is selected if the range includes item 0. If the entire contents are deleted, there is no current selection after this function has returned. As with the glishtk listbox, the index numbers here are 0-relative (i.e. the first item is item number 0). "selected" translates to the index of the current selection, "end" translates to the last item in the combobox. delete() returns a fail if the index is out of range or otherwise does not exist (e.g. there is no current selection) and the contents of the combobox will remain unchanged in that case.

Example

```
f := frame();
        cb := combobox(f,'colors',"red blue green yellow amber saffron")
        # delete everything from the current selection to the end
        # the new selection is whatever immediately preceeds the current
        # selection or nothing, if 'selected' is item 0.
        cb.delete('selected','end')
```

---

combobox.insert - Function

Package display
Module widgets
Tool combobox

Insert an item into the combobox.

Synopsis
insert(newitem, index, select)

Arguments

newitem	*The thing to insert* *Allowed: A string*
index	*Its new location* *Allowed: Any index ¡= largest index + 1* *Default: F*
select	*Select this item* *Allowed: T or F* *Default: F*

Returns
T

Description
If index is F then the current autoinsertorder determines which and of the list of items this item is added at. If an index is specified, all of the existing items from that index to the end are moved to the next location (i.e. an insert at location 5 results in the indexes of 5 to the end being now 6 to the new end). This may cause the index of the current selection to change. As with the glishtk listbox, the index numbers here are 0-relative (i.e. the first item is item number 0). "selected" translates to the index of the current selection, "end" translates to the last item in the combobox. insert() returns a fail if the index is out of range (more than 1 larger than the current largest index). If select is T, this item will be selected after insertion.

---

combobox.insertentry - Function

Package display
Module widgets
Tool combobox

Replace the contents of the entry.

Synopsis

insertentry(newentry)

Arguments

newentry	*The new contents of the entry* *Allowed: A string*

Description
This new entry is not selected and it does not appear in the combobox popup. In other words, this is just as if the user had typed this string and not yet hit return in the entry.

---

combobox.vscrollbar - Function

Package display
Module widgets
Tool combobox

Change the vertical scrollbar behavior of the combobox popup.

Synopsis
vscrollbar(vscrollbar)

Arguments

vscrollbar	*The new vertical scrollbar behavior* *Allowed: 'ondemand', 'always', 'none'*

Description
When 'ondemand' is requested, the vertical scrollbar only appears when the number of items is larger than listboxheight elements. When 'always' is requested, the listbox always has a scrollbar. When 'none' is requested, the listbox never has a scrollbar and it is always high enough to show all of the items in the combobox.

---

combobox.hscrollbar - Function

Package display
Module widgets
Tool combobox

Change the horizontal scrollbar behavior of the entry and the combobox popup.

Synopsis
hscrollbar(hscrollbar)

Arguments

hscrollbar	*The new horizontal scrollbar behavior* *Allowed: 'ondemand', 'always', 'none'*

Description
The entry and listbox popup share the same horizontal scrollbar behavior. When one has a horizontal scrollbar, the other does as well. When 'ondemand' is requested, the horizontal scrollbar only appears when the number longest item is longer than entrywidth elements. When 'always' is requested, the horizontal scrollbar is always present. When 'none' is requested, there is no horizontal scrollbar and the listbox is always wide enough to show all of the items in the combobox.

---

combobox.autoinserorder - Function

Package **display**
Module **widgets**
Tool **combobox**

Change the automatic insertion behavior.

Synopsis
autoinserorder(insertorder)

Arguments

insertorder	*The new automatic insertion behavior* *Allowed: 'tail', 'head'*

Description
This determines how new items are added to the combobox when no specific index location is specified - either through the use of the insert function or through the users actions in the entry field. If this is 'tail', the new item is added to the end of the list, if this is 'head', the new item is added to the head of the list.

---

combobox.bind - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Bind an X event to the combobox entry.

Synopsis

bind(xevent, eventname)

Arguments

xevent	*The X event to be bound.* *Allowed: An X event in glishtk syntax*
eventname	*The name which will be associated with this event.* *Allowed: A string*

Description

This event will be bound to the combobox entry. Events having the name eventname will be emitted from the agent returned by the agent member function each time this X event occurs within the combobox entry.

Example

```
cb.bind('<KeyPress>','keypress')
whenever cb.agent()->keypress do {
   print 'Key pressed in combobox entry : ', $value;
}
```

---

combobox.cursor - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the default cursor within the combobox.

Synopsis
cursor(xcursor)

Arguments

xcursor	*The new default cursor.* *Allowed: An X cursor*

---

combobox.popupcursor - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the default cursor within the combobox popup.

Synopsis
popupcursor(xcursor)

Arguments

xcursor	*The new default cursor in the popup.* *Allowed: An X cursor*

---

combobox.labeltext - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the text of the combobox label.

Synopsis
labeltext(newlabel)

Arguments

newlabel	*The new label text.* *Allowed: A string.*

---

combobox.labelwidth - Function

Package display
Module widgets
Tool combobox

Change the width of the combobox label.

Synopsis
labelwidth(newwidth)

Arguments

newwidth	*The new label width.* *Allowed: An integer.*

---

combobox.labelfont - Function

Package display
Module widgets
Tool combobox

Change the font of the combobox label.

Synopsis
labelfont(newfont)

Arguments

newfont	*The new label font.* *Allowed: An X font.*

---

combobox.labelrelief - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the relief of the combobox label.

Synopsis
labelrelief(newrelief)

Arguments

newrelief	*The new label relief.* *Allowed: A glishtk relief string.*

---

combobox.labeljustify - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the text justification within the combobox label.

Synopsis
labeljustify(newjustify)

Arguments

newjustify	*The new label text justification.* *Allowed: A glishtk justify string.*

---

combobox.labelforeground - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the foreground color of the combobox label.

Synopsis
labelforeground(newforeground)

Arguments

newforeground	*The new label foreground color.* *Allowed: An X color.*

---

combobox.labelbackground - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the background color of the combobox label.

Synopsis
labelbackground(newbackground)

Arguments

newbackground	*The new label background color.* *Allowed: An X color.*

---

combobox.labelanchor - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the anchor attribute of the combobox label.

Synopsis
labelanchor(newanchor)

Arguments

newanchor	*The new label anchor attribute.* *Allowed: A glishtk anchor string.*

---

combobox.entrydisabled - Function

Package display
Module widgets
Tool combobox

Disable or enable the entry.

Synopsis
entrydisabled(tOrF)

Arguments

tOrF	*The state of the entry.* *Allowed: T or F.*

Description
When the entry is disabled (T here), the user can not type into the entry part of the combobox. This makes it impossible for the user to add to the contents of the combobox graphically. The combobox functions still behave as before so an application is free to do whatever it wants to do to the combobox. This allows an application to present an unchangable set of combobox options to the user. The user can only choose from the existing set of items. They can not add to them. If the application programmer has choosen to provide a clear button in the listbox, however, it would then be possible for the user to clear the contents of the combobox.

---

combobox.entrywidth - Function

Package display
Module widgets
Tool combobox

Change the width of the combobox entry.

Synopsis
entrywidth(newwidth)

Arguments

newwidth	*The new entry width.* *Allowed: An integer.*

Description
Note that this will affect the width of the listbox popup when no horizontal scrollbar is present.

---

combobox.entryfont - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the font of the combobox entry and popup.

Synopsis
entryfont(newfont)

Arguments

newfont	*The new entry font.* *Allowed: An X font.*

---

combobox.entryrelief - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the relief of the combobox entry and popup.

Synopsis
entryrelief(newrelief)

Arguments

newrelief	*The new entry relief.* *Allowed: A glishtk relief string.*

---

combobox.entryjustify - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the text justification within the combobox entry and popup.

Synopsis
entryjustify(newjustify)

Arguments

newjustify	*The new entry text justification.* *Allowed: A glishtk justify string.*

---

combobox.entryforeground - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the foreground color of the combobox entry and popup.

Synopsis
entryforeground(newforeground)

Arguments

newforeground	*The new entry foreground color.* *Allowed: An X color.*

---

combobox.entrybackground - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the background color of the combobox entry and popup.

Synopsis
entrybackground(newbackground)

Arguments

newbackground	*The new entry background color.* *Allowed: An X color.*

---

combobox.arrowbutton - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the bitmap which is the arrow button.

Synopsis
arrowbutton(newbutton)

Arguments

newbutton	*The new arrow button bitmap.* *Allowed: The name of a file containing an X bitmap.*

---

combobox.arrowbackground - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the background color of the arrow button.

Synopsis
arrowbackground(newbackground)

Arguments

newbackground	*The new arrow button background color.* *Allowed: An X color.*

---

combobox.arrowforeground - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the foreground color of the arrow button.

Synopsis
arrowforeground(newforeground)

Arguments

newforeground	*The new arrow button foreground color.* *Allowed: An X color.*

---

combobox.listboxheight - Function

Package	**display**
Module	**widgets**
Tool	**combobox**

Change the height of the listbox.

Synopsis
listboxheight(newheight)

Arguments

newheight	*The new listbox height.* *Allowed: An integer.*

Description
This value is only important if there is a vertical scrollbar present. If the vertical scrollbar is always on, the listbox is always this height. If the vertical scrollbar is only on when needed, the listbox is this height when the number of entries in the combobox exceeds this number, otherwise the listbox is only as high as it needs to be to hold the number of items in the combobox. If there is no scrollbar, the listbox will be whatever height is necessary to see all of the combobox entries at the same time without scrolling.

1.4.8 deleteregions - Tool

Package display
Module widgets

Tk widget to delete image regions from a Table
include "deleteregions.g"

Constructors

deleteregions	*Construct a deleteregions widget*

Functions

done	*Destroy ourselves*
gui	*Restart the GUI*
refresh	*Refresh the display*
setregions	*Set new regions*
settable	*Set a (new) table*

Description

This Tk widget enables you to delete `regions-of-interest` *(or more simply, regions - see Regionmanager) stored in an AIPS++ Table. It can also be used to provide a GUI interface for deleting regions from a record or global name space (from* Glish*).*
The widget displays, in listboxes, the regions stored in the Table (or record or

Glish*). The 'Delete' button is used to delete the regions that you have selected. The 'Delete all' button deletes all regions. The 'Deselect' button deselects all regions. Optionally, you can be prompted for confirmation before deletion by selecting from the 'Confirm' menu. The 'Dismiss' button dismisses the GUI. You can recover it with the gui function.*
Under the 'File' menu, you can either dismiss the GUI or destroy the widget (same as invoking function done).
The Table can be specified in many ways (at construction and via the entry box in the GUI); a Table `tool`*, an Image)* `tool`*, the name of a Table (a string) which might be an AIPS++ image, or the name of a* Glish *symbol which is either a Table or an Image* `tool`*.*
This widget emits an event called `deleted` *(it has no value) when the regions have been deleted. It also emits an event called* `dismissed` *(it has no value) when the dismiss button on the GUI is pressed.*

deleteregions - Constructor

Package **display**
Module **widgets**
Tool **deleteregions**

Construct a deleteregions widget

Synopsis
deleteregions(parent, table, regions, source, widgetset)

Arguments

parent	*Parent for the widget* *Allowed: Widget*
table	*The table of interest* *Allowed: Table tool, image tool or string* *Default: ''*
regions	*Regions or region names (for source 'record' or 'global'* *Allowed: Record or vector of strings* *Default: ""*
source	*Source of the regions* *Allowed: String from 'table', 'record', or 'global'* *Default: 'table'*
widgetset	*The widget server to use* *Allowed: widgetserver tool* *Default: dws*

Returns
Agent

Description

This is the constructor. If `parent=F`*, then the widget will make its own parent frame.*
The argument `source` *specifies what the source of the regions is.*
If `source='table'` *then the regions are to be found in the Table specified with the argument* `table`*. The argument* `regions` *is ignored in this case. If you leave* `table` *unspecified, then you can enter the Table into the entry box provided in the GUI.*
The Table can be specified in many ways (at construction and via the entry box in the GUI); a Table `tool`*, an Image)* `tool`*, the name of a Table (a string) which might be an AIPS++ image, or the name of a* Glish *symbol which is either a Table or an Image* `tool`*.*
If `source='global'` *then the regions are to be found in the global name space of* Glish*. The argument* `table` *is ignored in this case. You must specify the names of the regions as a vector of strings with the* `regions` *argument.*
If `source='record'` *then the regions are to be found in the record given by the argument* `regions`*. The argument* `table` *is ignored in this case.*

Example

```
- include 'deleteregions.g'
- include 'regionmanager.g'
- dr1 := deleteregions(table='hcn', source='table');
- whenever dr1->deleted do {
   print 'regions deleted from table';
  }
-
- r1 := drm.quarter();
- r2 := drm.box();
- r3 := drm.box([10,10],20,20]);
- dr2 := deleteregions(source='global', regions="r1 r2 r3");
- whenever dr2->deleted do {
   print 'regions deleted from Glish';
  }
-
- r1 := drm.quarter();
- r2 := drm.box();
- r3 := drm.box([10,10],[20,20]);
- rec := [=];
```

```
- rec['r1'] := r1; rec['r2'] := r2; rec['r3'] := r3;
- dr3 := deleteregions(source='record', regions=rec);
- whenever dr3->deleted do {
   print 'regions deleted from record';
  }
-
```

We show the three ways to source the regions.

---

deleteregions.done - Function

Package	**display**
Module	**widgets**
Tool	**deleteregions**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This function destroys the widget.

Example

```
- r := deleteregions();
- r
ref [*agent*=<agent>, done=<function>, gui=<function>, settable=<function>]
- r.done();
T
- r
F
-
```

---

deleteregions.gui - Function

Package	**display**
Module	**widgets**
Tool	**deleteregions**

Restart the GUI

Synopsis
gui()

Returns
Bool

Description

When the widget is first constructed, it brings up a GUI. When you delete regions, the GUI is dismissed. This function restarts the GUI; it will have the same state as when it was dismissed.

---

deleteregions.refresh - Function

Package	**display**
Module	**widgets**
Tool	**deleteregions**

Refresh the display

Synopsis
refresh()

Returns
Bool

Description
If the source is from a Table, this function reaccesses the table to find the regions and redisplays them. Otherwise it just clears any selection.

---

deleteregions.setregions - Function

Package	**display**
Module	**widgets**
Tool	**deleteregions**

Set new regions

Synopsis
setregions(regions)

Arguments

regions	*Regions or region names (for source 'record' or 'global'* *Allowed: Record or vector of strings* *Default: ""*

Returns
Bool

Description

This function lets you set a new set of regions into the widget. The source of the regions is the same as at construction and must match.

Example

```
- include 'deleteregions.g'
- include 'regionmanager.g'
- r1 := drm.quarter();
- r2 := drm.box();
- r3 := drm.box([10,10],20,20]);
- rec := [=];
- rec[1] := r1; rec[2] := r2; rec[3] := r3;
- dr3 := deleteregions(source='record', regions=rec);
- rec := [=];
= rec[1] := r1; rec[2] := r2;
- dr3.setregions(rec);
T
```

---

deleteregions.settable - Function

Package	**display**
Module	**widgets**
Tool	**deleteregions**

Set a (new) table

Synopsis
settable(table)

Arguments

table	*The table of interest* *Allowed: Table tool, image tool or string* *Default: ''*

Returns
Bool or fail

Description

This function manually (i.e. not via the GUI interface) sets a (new) Table name. It is equivalent to doing it via the GUI. The Table can be specified in many ways (at construction and via the entry box in the GUI); a Table `tool`*, an Image)* `tool`*, the name of a Table (a string) which might be an AIPS++ image, or the name of a* Glish *symbol which is either a Table or an Image* `tool`*.*

Example

```
- include 'deleteregions.g'
- include 'image.g'
- include 'table.g'
- r := deleteregions(table='hcn');
- im := image('hcn2');
- t := table('hcn2');
- r.settable(im);
- r.settable('hcn2');
- r.settable(t);
- r.settable('im');
- r.settable('t');
```

In this example, we show all the ways you can specify the table.

1.4.9 dialogbox - Tool

Package **display**
Module **widgets**

Tk widget to get input from the user
include "dialogbox.g"

Description

This AIPS++ Tk widget enables you to create a labelled and titled entry box (with optional popup help). The user can type into the entry box, and after pressing the return key (or pressing the GO button) the widget returns the user's input string and then destroys itself.
You can select what kind of entry you want to see. We have many kinds of entry widgets, such as files, strings, regions, measures and so on. You can select the type you want to see. Some entry widgets require an initial value, which is inserted into the entry.
There is also an ABORT button. Pressing this means that a fail is returned.

dialogbox - Constructor

Package **display**
Module **widgets**
Tool **dialogbox**

Construct a dialogbox

Synopsis

dialogbox(label, title, type, value, hlp, helpOnLabel, widgetset)

Arguments

label	*The label for the entry box* *Allowed: String* *Default: 'Enter value'*
title	*The title for the entry box* *Allowed: String* *Default: 'Dialog box'*
type	*The type of entry widgets* *Allowed: 'entry', 'file', 'string', 'boolean', 'measure', 'quantity', 'record', 'region', 'scalar, 'untyped'* *Default: 'entry'*
value	*The initial value to insert* *Allowed: Appropriate type*
hlp	*Popup help for entry box* *Allowed: String*
helpOnLabel	*Is the popup help on the label or the entry box* *Allowed: T or F* *Default: T*
widgetset	*The widget server to use* *Allowed: widgetserver object* *Default: dws*

Returns
String

Description

Example

```
include 'dialogbox.g'
txt := dialogbox(label='Enter colour', title='Colour chooser', type='string', value='Blue');
if (!is_fail(txt)) {
   print 'Selected colour=', txt
} else {
  print 'Colour selection aborted'
}
```

The initial value of 'blue' is inserted.

1.4.10 extendoptionmenu - Tool

Package **display**
Module **widgets**

Tk widget where the (interactively extendable) menu label changes with selection include "extendoptionmenu.g"

Constructors

extendoptionmenu	*Construct an extendoptionmenu*

Functions

disabled	*Disable or enable the extendoptionmenu widget*
done	*Destroy ourselves*
extend	*Extend the menu with more items*
findlabel	*Find the index of the item with the given label*
geteventmask	*Get the event mask*
getindex	*Get the index of the current selection*
getlabel	*Get the label of the current selection*
getlabels	*Gets all of the labels associated with the menu*
getpreviousindex	*Get the index previous to the current selection*
replace	*Replace the menu with the provided items*
selectindex	*Select an item in the menu by index*
selectlabel	*Select an item in the menu by label*
setbackground	*Set the color of the background of the extendoptionmenu button*
seteventmask	*Set the event mask*
setforeground	*Set the color of the foreground of the extendoptionmenu button*
setwidth	*Set the width of the optionmenu button*

Description

This Tk widget enables you to create a flat (i.e. no submenus) menu in a specified frame. You can associate a label (that which appears on the top menu button), a name (that which appears in the menu list) and a value (whatever you like, it may be different from the label) for each menu item. When you select one of the items in the menu, the label of the menu button changes to reflect the label of the selected item.
If your list of items is very long, a menu may be too long to sensibly manage. Therefore, you can optionally choose (see constructor) to use a popup scrolling listbox rather than a menu (it's really implemented with a selectabelist widget). It can be treated like any of the built in Tk widgets, and has almost all of the same construction arguments as the button Tk widget.
Whenever a selection is made, this widget emits an event called `select`*. In the $value of this event, you can find a record with fields* `label`*,* `name` *and* `value` *containing that information for the selected item. The event generation can be disabled if you desire with the seteventmask tool function.*
Whenever the menu is replaced with the replace tool function, this widget emits an event called `replaced` *(it has no value). Note that the tool function extend*

invokes the `replace` *function, so a* `replaced` *event is generated then as well. The widget accepts an event called* `setwidth`*. It is the same as calling the setwidth tool function.*
This widget is very much like the optionmenu widget. There are two differences. First, the `extendoptionmenu` *menu can be extended interactively during use. When constructed, you specify a special symbol for the menu (defaults to the ellipsis). When the user selects this item, you can type in a new entry in a ahlinkdialogboxwidgets:dialogbox or use callback functions that the programmer provides. Second,* `extendoptionmenu` *only offers you labels for the menu items, whereas optionmenu offers you labels, names, and values for each menu item. The* `select` *event is also triggered when you add an item to the menu.*

extendoptionmenu - Constructor

Package	**display**
Module	**widgets**
Tool	**extendoptionmenu**

Construct an extendoptionmenu

Synopsis

extendoptionmenu(parent, labels, hlp, hlp2, nbreak, symbol, callback1, callback2, callbackdata, dialoglabel, dialogtitle, padx, pady, width, updatelabel, height, justify, font, relief, borderwidth, foreground, background, disabled, anchor, fill, widgetset)

Arguments

parent	*Parent of the menu* *Allowed: Widget*
labels	*The menu item labels* *Allowed: Vector of strings* *Default: Empty menu*
hlp	*Short popuphelp text* *Allowed: String* *Default: No help*
hlp2	*Long popuphelp text* *Allowed: String* *Default: No help*
nbreak	*The break point for menu or listbox usage* *Allowed: Integer* *Default: 20*
symbol	*Special symbol for menu extension* *Allowed: String* *Default: '...'*
callback1	*Callback function when special symbol selected* *Allowed: String* *Default: F*
callback2	*Callback function when special symbol selected* *Allowed: String* *Default: F*
callbackdata	*Data for callback2 calculations* *Allowed: Any Glish variable* *Default: F*
dialoglabel	*Label for dialogbox* *Allowed: String* *Default: 'Item'*
dialogtitle	*Title for dialogbox* *Allowed: String* *Default: 'Enter new item ¡CR¿'*
padx	*Horizontal padding around text* *Allowed: Integer* *Default: 7*
pady	*Vertical padding around text* *Allowed: Integer* *Default: 3*
width	*Width in character units* *Allowed: Integer* *Default: Auto-scale*
updatelabel	*Update menu label with selected item* *Allowed: T or F* *Default: T*
height	*Height in character units* *Allowed: Integer* *Default: 1*
justify	*Justification of text* *Allowed: String from 'center', 'left', 'right'* *Default: 'center'*
font	*Font of text* *Allowed: X font*
relief	*Edge relief* *Allowed: String from 'flat', 'ridge', 'raised', 'sunken', 'groove'*

Returns
Agent

Description

By default, the `extendoptionmenu` *will insert a "..." into the menu (you don't include it in the* `labels` *list). Use the argument* `symbol`*) to use a different special selection symbol. When the user selects this item from the menu, a few things might happen.*
First, if `callback1` *is a valid function, then it is invoked (it has no arguments), and the result is expected to be a string which is inserted into the menu (duplicates are ignored).*
Second, if `callback1` *is not a valid function, then a ahlinkdialogboxwidgets:dialogbox widget is created, and the user types the new menu item label into the entry box of this widget. If the new item is a duplicate, nothing further happens. Otherwise, if* `callback2` *is a valid function, then it is invoked with arguments* `callback2(newitem, labels, callbackdata`*. Here,* `newitem` *is the string that came from the* `dialogbox`*.* `labels` *is the list of current menu labels, and* `callbackdata` *is any Glish variable you care to pass into the* `callback2` *function. It is up to you, the programmer, whether you need to use* `callbackdata` *or not. If the result returned from* `callback2` *is* `T`*, then the* `newitem` *is placed in the menu.*
Note that the `width` *argument can be used in three ways. If* `width=-1` *(the default), the menu button will be wide enough (and of fixed width) to accomodate all of the item labels in the menu. This means that the GUI won't jitter when you write different length labels into it. If* `width=0`*, then the menu button will change its width to accomodate whatever label is written into it. If* `width > 0`*, then the button is again of fixed width, with the width that you specify.*
If there are less items in the list than `nbreak`*, you will see a flat menu. Otherwise, you will see a scrolling list box. Note that to activate the listbox you must press and release the main button. Whereas for the flatmenu, pressing is enough to activate it.*

Example

```
- include 'extendoptionmenu.g'
- labels := "m cm mm";
- f := frame();
- em := extendoptionmenu(f, labels);
```

The ellipsis is used, by default, as the menu extend symbol. A dialog box will appear if the user selects the extend symbol. If the user types in a string which is not already in the list of labels, it will be added to the menu.

Example

```
- include 'quanta.g'
- include 'extendoptionmenu.g'
- qs := quanta();
- labels := "m cm mm";
- f := frame();
- em := extendoptionmenu(f, labels, callback2=myfun, callbackdata=qs)
```

The ellipsis is used, by default, as the menu extend symbol. A dialog box will appear if the user selects the extend symbol. If the user types in a string which is not already in the list of labels, it will be added to the menu provided the function `myfun`*, which you supply, returns T. Function tt myfun is given the quanta server tool,* `qs` *for use.*

---

extendoptionmenu.extend - Function

Package	**display**
Module	**widgets**
Tool	**extendoptionmenu**

Extend the menu with more items

Synopsis
extend(labels, width)

Arguments

labels	*The menu item labels* *Allowed: Vector of strings* *Default: Empty menu*
width	*Width in character units* *Allowed: Integer* *Default: Auto-scale*

Returns
Bool

Description

This tool function extends the menu with the supplied items.

Example

```
- labels := "m cm mm";
- f := frame();
- m := extendoptionmenu(f, labels);
- m.extend(labels="dm Dm");
- m.getlabels();
m cm mm dm Dm
-
```

Note that unlike when you interactively extend the menu, no checking for the validity of the new items is done. That is your responsibility.

---

extendoptionmenu.replace - Function

Package	**display**
Module	**widgets**
Tool	**extendoptionmenu**

Replace the menu with the provided items

Synopsis
replace(labels, width)

Arguments

labels	*The menu item labels* *Allowed: Vector of strings* *Default: Empty menu*
width	*Width in character units* *Allowed: Integer* *Default: Auto-scale*

Returns
Bool

Description

This tool function replaces the menu with the supplied items.

Example

```
- labels := "m cm mm";
- f := frame();
- m := extendoptionmenu(f, labels);
m.getlabels();
m cm mm
- m.replace("dm Dm");
- m.getlabels();
dm Dm
-
```

---

extendoptionmenu.selectindex - Function

Package display
Module widgets
Tool extendoptionmenu

Select an item in the menu by index

Synopsis
selectindex(idx)

Arguments

idx	*The menu item index* *Allowed: Integer*

Returns
Bool

Description

This tool function selects a menu item by its index. It is equivalent to selecting the item with the GUI interface. If the index is invalid, it returns `F`.

Example

```
- labels := "m mm cm";
- f := frame();
- m := extendoptionmenu(f, labels);
- m.selectindex(3);
- m.getlabel();
cm
-
```

---

extendoptionmenu.selectlabel - Function

Package	**display**
Module	**widgets**
Tool	**extendoptionmenu**

Select an item in the menu by label

Synopsis
selectlabel(label)

Arguments

label	*The menu item label* *Allowed: Same type as the array "labels" used in construction*

Returns
Bool

Description

This tool function selects a menu item by its label. It is equivalent to selecting the item with the GUI interface. If the label is invalid, it returns `F`.

Example

```
- labels := "m mm cm";
- f := frame();
- m := extendoptionmenu(f, labels)
```

```
- m.selectlabel('mm');
- m.getlabel();
mm
-
```

Returns
Bool

Description

This tool function selects a menu item by its name. It is equivalent to selecting the item with the GUI interface. If the name is invalid, it returns `F`.

Example

```
- labels := "m mm cm";
- names := "A B C";
- f := frame();
- m := extendoptionmenu(f, labels, names);
- m.selectname("C");
- m.getindex();
3
-
```

---

extendoptionmenu.getindex - Function

Package	**display**
Module	**widgets**
Tool	**extendoptionmenu**

Get the index of the current selection

Synopsis
getindex()

Returns
Integer or F

Description

This tool function recovers the index of the currently selected item in the menu. If the menu is empty, it returns `F`*. If no menu item has been selected by the user, it returns unity.*

Example

```
- labels := "m mm cm";
- f := frame();
- m := extendoptionmenu(f, labels)
- m.selectindex(2);
- m.getindex();
2
```

---

extendoptionmenu.getpreviousindex - Function

Package	**display**
Module	**widgets**
Tool	**extendoptionmenu**

Get the index previous to the current selection

Synopsis
getpreviousindex()

Returns
Integer or F

Description

This tool function recovers the index that you selected previous to the current selection. If the menu is empty, it returns `F`*. If no menu item has been selected by the user, it returns unity.*

Example

```
- labels := "m mm cm";
- f := frame();
- m := extendoptionmenu(f, labels);
- m.selectindex(2);
- m.selectindex(3);
- m.getpreviousindex();
2
```

---

extendoptionmenu.getlabel - Function

Package	**display**
Module	**widgets**
Tool	**extendoptionmenu**

Get the label of the current selection

Synopsis
getlabel()

Returns
String or F

Description

This tool function recovers the label of the currently selected item in the menu. If the menu is empty, it returns `F`. *If no menu item has been selected by the user, it returns the first item.*

Example

```
- labels := "m mm cm";
- f := frame();
- m := extendoptionmenu(f, labels);
- m.selectindex(2);
- m.getlabel();
mm
```

---

extendoptionmenu.getlabels - Function

Package	**display**
Module	**widgets**
Tool	**extendoptionmenu**

Gets all of the labels associated with the menu

Synopsis
getlabels()

Returns
Vector of String

Description

This tool function recovers the labels associated with each item in the menu. If the menu is empty, it returns an empty array.

Example

```
- labels := "m mm cm";
- f := frame();
- m := extendoptionmenu(f, labels);
- m.getlabels();
m mm cm ...
```

Note the ellipsis shows in the list.

---

extendoptionmenu.findlabel - Function

Package	**display**
Module	**widgets**
Tool	**extendoptionmenu**

Find the index of the item with the given label

Synopsis

findlabel(label)

Arguments

label	*The menu item label* *Allowed: Same type as the array "labels" used in construction*

Returns
Integer or F

Description

This tool function find the index in the menu of the item specified by its label. If you give an invalid label, `F` *is returned.*

Example

```
- labels := "m mm cm";
- f := frame();
- m := extendoptionmenu(f, labels);
- m.findlabel('cm');
3
```

---

extendoptionmenu.geteventmask - Function

Package display
Module widgets
Tool extendoptionmenu

Get the event mask

Synopsis
geteventmask()

Returns
Vector of Bool

Description

This tool function recovers the event mask. This mask indicates whether the `select` *event will be emitted or not when you select a menu item. By default the mask is all T. You can modify it with the seteventmask tool function. New items added to the menu get a T value for their mask.*

Example

```
- labels := "m mm cm";
- f := frame();
- m := extendoptionmenu(f, labels);
- m.geteventmask();
[T T T T]
- m.seteventmask([T,F,F,T]);
- m.geteventmask();
[T F F T]
```

Note that an element for the ellipsis is recovered too (element 4).

---

extendoptionmenu.seteventmask - Function

Package	**display**
Module	**widgets**
Tool	**extendoptionmenu**

Set the event mask

Synopsis
seteventmask(mask)

Arguments

mask	*The event mask* *Allowed: Vector of Bool*

Returns
Bool

Description

This tool function sets the event mask. This mask indicates whether the `select` *event will be emitted or not when you select a menu item. By default the mask is all T. You can get the current mask with the geteventmask tool function. If the mask is invalid, it returns* `F`*. New items added to the menu get a T value for their mask.*

Example

```
- labels := "m mm cm";
- f := frame();
- m := extendoptionmenu(f, labels);
- m.seteventmask([T,F,T,T]);
- m.geteventmask();
[T F T T]
- whenever m->select do {print $value;};
```

In this example, when you selected the second item, no event would be emitted. Note that an element for the ellipsis is required too (element 4).

---

extendoptionmenu.setforeground - Function

Package	**display**
Module	**widgets**
Tool	**extendoptionmenu**

Set the color of the foreground of the extendoptionmenu button

Synopsis
setforeground(color)

Arguments

color	*The color* *Allowed: Any X color encoded as a String*

Returns
Bool

Description

This tool function sets the color of the foreground of the extendoptionmenu button.

---

extendoptionmenu.setbackground - Function

Package	**display**
Module	**widgets**
Tool	**extendoptionmenu**

Set the color of the background of the extendoptionmenu button

Synopsis
setbackground(color)

Arguments

color	*The color* *Allowed: Any X color encoded as a String*

Returns
Bool

Description

This tool function sets the color of the background of the extendoptionmenu button.

---

extendoptionmenu.setwidth - Function

Package	**display**
Module	**widgets**
Tool	**extendoptionmenu**

Set the width of the optionmenu button

Synopsis
setwidth(width)

Arguments

width	*The width in character units* *Allowed: Integer*

Returns
Bool

Description

This tool function sets the width of the optionmenu button in character units.

---

extendoptionmenu.disabled - Function

Package display
Module widgets
Tool extendoptionmenu

Disable or enable the extendoptionmenu widget

Synopsis
disabled(disable)

Arguments

disable	*Disable the widget* *Allowed: T or F* *Default: T*

Returns
Bool

Description

This tool function disables or enables the extendoptionmenu.

---

extendoptionmenu.done - Function

Package	**display**
Module	**widgets**
Tool	**extendoptionmenu**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This tool function destroys the widget.

Example

```
- f := frame();
- m := extendoptionmenu(f, labels="a b c");
- m.done();
T
- m
F
-
```

1.4.11 fontchooserwidget - Tool

Package	**display**
Module	**widgets**

AIPS++widget to help users select a font
g
include "fontchooserwidget.g"

Description

This AIPS++ widget allows the user to select fonts.

fontchooserwidget - Constructor

Package	**display**
Module	**widgets**
Tool	**fontchooserwidget**

Construct a fontchooserwidget

Synopsis
fontchooserwidget(parent, font, widgetset)

Arguments

parent	*Parent of the menu* *Allowed: Tk frame* *Default: F*
font	*Font of text* *Allowed: X font* *Default: F*
widgetset	*Widget set to use* *Allowed: A widgetserver closure tool* *Default: dws*

Returns
fontchooserwidget widget

Description
Constructs a fontchooserwidget according to the specified construction arguments, and returns a subsequence to the widget. Remember to use the `fontchooserwidget` *via the widgetserver if at all possible.*
You may send the fontchooserwidget 3 events

1. **reset** *cause the fontchooser to selected it's initial font.*
2. **map** *causes the fontchooser to become visible if not already.*
3. **unmap** *causes the fontchooser to become invisible if not already.*

The fontchooser widget sends one event

1. **newfont** *who's values is the font description string selected by the user.*

Example

```
include 'fontchooserwidget.g'
f := dws.frame();
myfont :=  '-adobe-courier-medium-r-normal--14-*'
fc := fontchooserwidget(f, myfont);
whenever fc->newvalue do {
  print 'font selected is ', $value;
}
```

1.4.12 gchooser - Tool

Package display
Module widgets

Tool for generating widgets for choosing items interactively
include "gchooser.g"

Description

This tool generates a Tk widget for choosing items interactively. The items must be associated with x, y coordinates, and a label. The items are plotted using the pgplotter, and the user is allowed to select items by left-clicking or to deselect items by right-clicking. The plot frame may also be printed as a postscript file or saved as an AIPS++ plot file.
An example of the use of gchooser is given in figure 1.12.
The constructor returns a subsequence with functions:

self.done *Delete this widget*

self.get *Get a record containing information on the choices*

self.insert *Insert a set of choices as a vector of the selected indices.*

The events are:

self-¿select *Returns index selected*

self-¿deselect *Returns index deselected*

self-¿values *A set of values was accepted. The returned record contains information about the choices.*

The selection works as follows:

To select a region *Click mouse button 1 (usually the left hand button) and drag over the region that you wish to select. All elements in the region delineated are selected, and the items selected are plotted in green.*

To select a given element *Click mouse button 2 (usually the middle button) near a label. The item selected in plotted in green.*

To deselect a given element *Click mouse button 3 (usually the right hand button) near a label. The item deselected in plotted in red.*

To select all items *Click the All button in the bottom left hand corner. All items will be plotted in green.*

To dselect all items *Click the None button in the bottom left hand corner. All items will be plotted in red.*

If the argument axes is set to 'sky', then the axes are labelled according to interpretation as Right Ascension (x) and Declination (y).

Example

```
include 'gchooser.g'
include 'table.g'
t:=table('3C273XC1.ms/ANTENNA');
position := t.getcol('POSITION');
labels := t.getcol('NAME');
indices := t.getcol('ANTENNA_ID')+1;
t.done();
x := position[2,];
y := -position[1,];

gc := gchooser(labels=labels, indices=indices, x=x, y=y, autoref=T,
 plottitle='Antenna locations',
 width=500, height=500);
whenever gc->values do {
  print "Values are ", $value;
}
whenever gc->select do {
  print "Selected ", $value
}
whenever gc->deselect do {
```

```
  print "Deselected ", $value
}
-
Selected 4
Selected 28
Selected 11
Selected 26
Selected 17
Selected 15
Selected 18
Selected 24
Values are [selection=[4 11 15 17 18 24 26 28] , labels=VLA:W6 VLA:OUT
VLA:N2 VLA:N4 VLA:E4 VLA:E6 VLA:E2 VLA:W2, x=[-207.86 37.1272 32.3322
25.3736 170.777 304.882 77.7937 0] , y=[-78.7424 14.7828 44.846
89.6066 -20.8221 -55.868 3.48509 0] ]
```

gchooser - Constructor

Package	**display**
Module	**widgets**
Tool	**gchooser**

Make a gchooser tool

Synopsis

gchooser(parent, labels, x, y, xref, yref, autoref, title, xlabel, ylabel, axes, plottitle, width, height, pad, widgetset)

Arguments

parent	*Parent frame: optional* *Allowed: frame agent* *Default: F*
labels	*List of labels* *Allowed: Vector of strings* *Default: ''*
x	*X positions* *Allowed: Numeric vector* *Default: []*
y	*Y positions* *Allowed: Numeric vector* *Default: []*
xref	*X reference* *Allowed: Numeric* *Default: unset*
yref	*Y reference* *Allowed: Numeric* *Default: unset*
autoref	*Determine reference position automatically* *Allowed: Boolean* *Default: F*
title	*Title of widget* *Allowed: String* *Default: 'Graphical chooser (AIPS++)'*
xlabel	*X axis label* *Allowed: String* *Default: 'X'*
ylabel	*Y axis label* *Allowed: String* *Default: 'Y'*
axes	*Type of axis* *Allowed: String* *Default: ''—'sky'*
plottitle	*Plot title* *Allowed: String* *Default: ''*
width	*Width of plot frame (pixels)* *Allowed: Integer* *Default: 500*
height	*Height of plot frame (pixels)* *Allowed: Integer* *Default: 500*
pad	*Padding around plot (fraction of axis length)* *Allowed: Float* *Default: 0.15*
widgetset	*Widget server to use* *Allowed: A widgetserver closure tool* *Default: dws*

Returns
gchooser widget

Description
Constructs a gchooser according to the specified construction arguments, and returns a ref to the tool.

1.4.13 gchoosertest - Function

Package **display**
Module **widgets**

Test the gchooser widgets

Synopsis
gchoosertest()

Returns
Boolean

Description
This demonstrates the use of the gchooser widget on field positions from a measurement set.

1.4.14 gopher - Tool

Package **display**
Module **widgets**

Tool for getting useful information
include "gopher.g"

Constructors

gopher	*Make a gopher tool*

Functions

fromimage	*Get various types of information from an image*
fromms	*Get various types of information from a measurementset*

Description

This tool generates Tk widgets for getting various types of information from images, measurement sets, etc.. *The same information can be retrieved using other non-widget functions. This tool is used in the autogui.*
The exact behavior of the generated widget depends upon the type of entry but in general the user is prompted for a source of information, such as an image, and then a tool, such as a viewer is used to get a value, such as a direction.

gopher - Constructor

Package	**display**
Module	**widgets**
Tool	**gopher**

Make a gopher tool

Synopsis
gopher(widgetset)

Arguments

widgetset	*Widget server to use* *Allowed: A widgetserver closure tool* *Default: dws*

Returns
gopher widget

Description
Constructs a gopher according to the specified construction arguments, and returns a ref to the tool. Use of this constructor should be unnecessary since the default gopher (called defaultgopher or dgo) can nearly always be used.

Example

```
include 'gopher.g'
region := dgo.fromimage('3C273XC1.clean');
```

---

gopher.fromimage - Function

Package	**display**
Module	**widgets**
Tool	**gopher**

Get various types of information from an image

Synopsis
fromimage(file, what)

Description
This function allows interactive selection of various types of information from an image. This information is returned.
The types of information returned are:

region *An image region is returned from a display of an image. To set the region, you must first select the type of region (box or polygon) by pressing the appropriate button on the left hand side of the viewer, and second, use the mouse to drag out the region, and third, double-click inside the region to complete the selection.*

position *An image position is returned from a display of an image. To set the position, you must first enable to position selection by pressing the cross hairs icon on the left hand side of the viewer, second, move the cursor to the point of interest, then third, double-click inside the cross hairs.*

statistics *Open the Statistics frame, select a region as above, and the statistics will be calculated and returned.*

Arguments

file	*Image file name* *Allowed: String*
what	*Thing to return* *Allowed: String* *Default: 'region'—'position'—'statistics'*

Returns
glish variable

---

gopher.fromms - Function

Package	**display**
Module	**widgets**
Tool	**gopher**

Get various types of information from a measurementset

Synopsis
fromms(file, what)

Description
This function allows interactive selection of various types of information from a measurement set. This information is returned.
The types of information returned are:

antennas *The antenna locations are plotted and you may select interactively.*

baselines *The possible baseline combinations are plotted and you may select interactively.*

fields *The field positions are plotted and you may select interactively.*

spectralwindows *The spectral windows are plotted and you may select interactively.*

select *You make construct a Table Query Language string for a Measurement Set.*

Arguments

file	*MeasurementSet file name* *Allowed: String*
what	*Thing to return* *Allowed: String* *Default: 'antennas'—'baselines'—'fields'—'spectralwindows'*

Returns
glish variable

1.4.15 guientry - Tool

Package display
Module widgets

Tool for generating widgets for type-in of a GUI
include "guientry.g"

Constructors

guientry	*Make a guientry tool*

Functions

array	*Make a widget for an array entry*
boolean	*Make a widget for a boolean entry*
booleanarray	*Make a widget for a boolean array entry*
check	*Make a widget for a check entry*
choice	*Make a widget for a choice entry*
direction	*Make a widget for a direction entry*
done	*Destroy this tool*
doppler	*Make a widget for a doppler measure entry*
epoch	*Make a widget for a epoch entry*
file	*Make a widget for a file entry*
frequency	*Make a widget for a frequency measure entry*
list	*Make a widget for an extensible list of entry widget*
measurecodes	*Make a widget for entry of measures codes*
minmaxhist	*Constructs a widget for the entry of a range.*
position	*Make a widget for a position entry*
quantity	*Make a widget for a quantity entry*
radialvelocity	*Make a widget for a radialvelocity measure entry*
range	*Make a widget for a range entry*
record	*Make a widget for a Record entry*
region	*Make a widget for a region entry*
scalar	*Make a widget for a scalar entry*
scalarmeasure	*Make a widget for a scalar measure entry*
string	*Make a widget for a string entry*
twoentry	*Make a widget for a range entry, providing two entry boxes next to each other labelled 'm*
type	*Return the type of this tool*
untyped	*Make a widget for a untyped entry*

Description

This tool generates Tk widgets for the entry of various types of Glish variables.

This tool is used in the autogui. Note that guientry is served by the widgetserver and so that should be used to get access to a guientry.
The exact behavior of the generated widget depends upon the type of entry but in general:

- *The user may type in a value or a variable.*
- *A value can entered as a number in the entry field.*
- *A variable is entered by name only.*
- *A value may also be* unset *by using the unset button under a popupmenu menu.*
- *Either original or default values may be restored using the buttons under the popupmenu menu.*
- *Values may also be inserted via the insert function. Either a value (*e.g. *0.7), a string (*e.g. *'0.7'), or a variable (*e.g. `x` *or* `sqrt(y)`*) may be entered.*
- *Once initialized, a given widget can be used only for one type of argument (*i.e. *integer, double, complex). An attempt to enter a value of another type will generate an error.*
- *The widget always returns a a variable of the appropriate type via the get function. If the value of the widget is illegal then an illegal value is returned. This may be check for using the* `is_illegal()` *function.*
- *Entry of a valid value by any means generates a value event that has as an argument the value entered.*
- *The widget can be constructed so that the user cannot enter values directly (*`editable=F`*). This is used in the autogui for output-only values.*

The various functions of guientry generate a subsequence that can be used to interact with the widget. The function and events supported by the subsequence are:

`.get()` *Function to get the current value. WYSIWYG applies.*

`.insert(entry)` *Function to insert a value into the widget.*

`->insert(entry)` *Event to insert a value into the widget.*

`->value(entry)` *Event to designate that a user entered a new value.*

`.insertandemit(entry)` *Function to insert a value into the widget and emit a value event if the value is valid.*

`.setwidth(width)` *Function to set the total width in characters of the entry field(s).*

`.clear()` *Function to clear the current value.*

`.addtowrench` *Function to add a callback function to the wrench.*

`hlp` *Argument to specify popup help*

Parsing of entries is performed using the entryparser tool.
Most widgets have a popup menu that allows any of a number of operations. The standard ones are:

Original *Revert the entry to the original value that was first displayed.*

Default *Revert the entry to the default value for this actual widget.*

Unset *Toggle unset value. If the current value is not unset then it is changed to be so, if it already is unset then the original value is set.*

Copy *Copy the current entry to the clipboard whence it may be retrieved using* `dcb.paste()`.

Paste *Paste the current entry from the clipboard. An entry can be inserted in the clipboard using* `dcb.copy(value)`.

In addition, other operations may be present, as appropiate. For example, those widgets for accessing files of various types can bring up an interactive file chooser. One may add functions to the wrench, using addtowrench. The signature of the function is as in the following example. Note the use of the function insertandemit to insert the value and emit a value event if the value is valid. This is obviously required if a value is to be inserted into the widget.

```
#
# Addition to the wrench
#
  private.buttons.antennas := private.guientry.array();
  private.getantennas := function(ref self, ref its) {
    include 'gopher.g';
    wider private;
    self.insertandemit(dgo.getantennas(private.ms.name()));
  }
  private.buttons.antennas.addtowrench('Interactive', private.getantennas);
```

Example

```
include 'guientry.g'
f := dws.frame();
ge := dws.guientry()
s := ge.scalar(f, value=30.0);
```

```
whenever s->value do {
  print 'Gui selected a scalar ', s.get();
}
q := ge.quantity(f, value='100Hz');
whenever q->value do {
  print 'Gui selected a quantity ', q.get();
}
m := ge.measure(f, value='dm.direction(\'b1950\', \'0deg\', \'0deg\')')
whenever q->value do {
  print 'Gui selected a measure ', m.get();
}
```

guientry - Constructor

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a guientry tool

Synopsis

guientry(width, relief, font, background, foreground, editablecolor, uneditablecolor, unsetcolor, illegalcolor, borderwidth, expand, widgetset)

Arguments

width	*Initial width of entry field (characters)* *Allowed: Integer* *Default: 30*
relief	*Edge relief* *Allowed: String from 'flat', 'ridge', 'raised', 'sunken', 'groove'* *Default: 'flat'*
font	*Font of text* *Allowed: X font* *Default: ''*
background	*Background color* *Allowed: X color* *Default: 'lightgrey'*
foreground	*Color of text* *Allowed: X color* *Default: 'black'*
editablecolor	*Color for editable values* *Allowed: X color* *Default: 'white'*
uneditablecolor	*Color for uneditabe values* *Allowed: X color* *Default: 'lightgray'*
unsetcolor	*Color for unset values* *Allowed: X color* *Default: 'yellow'*
illegalcolor	*Color for illegal values* *Allowed: X color* *Default: 'red'*
borderwidth	*Borderwidth* *Allowed: Integer* *Default: 1*
expand	*Expand policy* *Allowed: 'x'—'y'—'both'—'none'* *Default: 'none'*
widgetset	*Widget server to use* *Allowed: A widgetserver tool* *Default: dws*

Returns
guientry widget

Description
Constructs a guientry according to the specified construction arguments, and returns a ref to the tool.

Example

```
include 'guientry.g'
f := dws.frame();
ge := dws.guientry()
r := ge.scalar(f, value=30.0);
whenever r->value do {
  print 'Gui selected is ', r.get();
}
```

---

guientry.scalar - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a scalar entry

Synopsis

scalar(parent, value, default, options, allowunset, editable, hlp)

Description

Constructs a widget for the entry of scalar values. The widget can determine the type of the first entry (integer, double or complex) and ensure that the type doesn't change. If neither the value nor the default are set AND allowunset is false then the options argument can be used to set the value that will actually be used as the initial value.

The **allowed formats for user input** *are:*

- *A scalar (*e.g. `1.3245`*)*
- *The name of Glish variable or function that evaluates to a scalar.*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: String or unset*
default	*Default value used to determine type of GUI* *Allowed: String*
options	*options* *Allowed: Glish value* *Default: ''*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

Example

```
include 'guientry.g'
f := dws.frame();
ge := dws.guientry()
sangle := ge.scalar(f, value=pi/2);
sfloat := ge.scalar(f, options=0.0);
sint := ge.scalar(f, options=0);
```

---

guientry.array - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for an array entry

Synopsis

array(parent, value, default, options, allowunset, editable, maxdisplay, hlp)

Description
Constructs a widget for the entry of array values.
The **allowed formats for user input** *are:*

- *A vector of numbers (*e.g. `[1.3, 4.5, 900.32]`*)*
- *The name of Glish variable or function that evaluates to a vector of numbers.*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*options* *Allowed: string* *Default: ''*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
maxdisplay	*Maximum number of elements that will be displayed* *Allowed: Positive integer* *Default: 100*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.booleanarray - Function

Package **display**
Module **widgets**
Tool **guientry**

Make a widget for a boolean array entry

Synopsis
booleanarray(parent, value, default, options, allowunset, editable, maxdisplay, hlp)

Description
Constructs a widget for the entry of array values.
The **allowed formats for user input** *are:*

- *A vector of booleans (*e.g. `[T, F, F, T]`*)*
- *The name of Glish variable or function that evaluates to a vector of booleans.*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: T or F* *Default: T*
default	*Default value* *Allowed: T or F* *Default: T*
options	*options* *Allowed: string* *Default: ''*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
maxdisplay	*Maximum number of elements that will be displayed* *Allowed: Positive integer* *Default: 100*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.boolean - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a boolean entry

Synopsis
boolean(parent, value, default, options, allowunset, editable, hlp)

Description
Constructs a widget for the entry of boolean values.
The **allowed formats for user input** *are:*

- *Interactively by selecting the desired value*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*options* *Allowed: string* *Default: ''*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.choice - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a choice entry

Synopsis
choice(parent, value, default, options, allowunset, editable, hlp)

Description
Constructs a widget for the choice of one string from many.
The **allowed formats for user input** *are:*

- *Interactively by selecting the desired value*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*Possible choices* *Allowed: Vector of strings*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

Example

```
include 'guientry.g'
f := dws.frame();
ge := dws.guientry();
ch := ge.choice(f, value='a', options="a b c");
whenever ch->value do {
  print 'Choice selected is ', ch.get();
}
```

---

guientry.check - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a check entry

Synopsis

check(parent, value, default, options, allowunset, editable, nperline, hlp)

Description

Constructs a widget for the choice of many strings from many.

The **allowed formats for user input** *are:*

- *Interactively by selecting the desired values*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*Possible choices* *Allowed: Vector of strings*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
nperline	*Number of items per line* *Allowed: Positive integer* *Default: 4*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.file - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a file entry

Synopsis
file(parent, value, default, options, allowunset, editable, types, hlp)

Description
The **allowed format for user input** *is simply a file name, relative or absolute.*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*options* *Allowed: string* *Default: ''*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
types	*File types to show (catalog)* *Allowed: String*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

Description

Constructs a widget for the entry of files. The `types` *argument is the same as the types available in the Catalog GUI. They can be recovered by the Catalog function availabletypesutility:catalog.availabletypes.*

---

guientry.measurecodes - Function

Package **display**
Module **widgets**
Tool **guientry**

Make a widget for entry of measures codes

Synopsis
measurecodes(parent, value, default, options, type, allowunset, editable, hlp)

Description

Constructs a widget for the entry of measures reference codes. For each type of measure there is a corresponding list of reference codes. For example, `dm.listcodes(dm.frequency())` *gives the list for a 'frequency' measure. The* **allowed formats for user input** *(i.e. for* `value` *and* `default`*) are:*

- *A measure code as drawn from the list of possible codes returned by the measures tool (see above).*
- *The name of Glish variable or function that evaluates to a measure code.*

Use `options` *to specify the measure type (see Measures) for the current known list (e.g. 'frequency', 'doppler', 'direction' etc).*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*type of measure: any valid measure* *Allowed: string* *Default: ''*
type	*Type of argument: unset means to evaluate* *Allowed: Variable* *Default: unset*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.scalarmeasure - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a scalar measure entry

Synopsis

scalarmeasure(parent, value, default, options, allowunset, editable, type, hlp)

Description

Constructs a widget for the entry of scalar measure values.
The **allowed formats for user input** *are:*

- *A measure as returned by the measures tool (*e.g. `dm.frequency('LSR', '10GHz')`*)*
- *The name of Glish variable or function that evaluates to a measure.*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*options* *Allowed: string* *Default: ''*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
type	*Type of argument: unset means to evaluate* *Allowed: Variable* *Default: unset*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.frequency - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a frequency measure entry

Synopsis
frequency(parent, value, default, options, allowunset, editable, hlp)

Description
Constructs a widget for the entry of frequency measure value.
The **allowed formats for user input** *are:*

- *A measure as returned by the measures tool (*e.g. `dm.frequency('LSR', '10GHz')`*)*
- *The name of Glish variable or function that evaluates to a frequency measure.*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*options* *Allowed: string* *Default: ''*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.radialvelocity - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a radialvelocity measure entry

Synopsis
radialvelocity(parent, value, default, options, allowunset, editable, hlp)

Description
Constructs a widget for the entry of radialvelocity measure value.
The **allowed formats for user input** *are:*

- *A measure as returned by the measures tool (*e.g. `dm.radialvelocity('LSR', '-100km/s')`*)*
- *The name of Glish variable or function that evaluates to a radialvelocity measure.*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*options* *Allowed: string* *Default: ''*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.doppler - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a doppler measure entry

Synopsis
doppler(parent, value, default, options, allowunset, editable, hlp)

Description
Constructs a widget for the entry of doppler measure value.
The **allowed formats for user input** *are:*

- *A measure as returned by the measures tool (*e.g. `dm.doppler('RADIO', 0.001)`*)*
- *The name of Glish variable or function that evaluates to a doppler measure.*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*options* *Allowed: string* *Default: ''*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.position - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a position entry

Synopsis
position(parent, value, default, options, allowunset, editable, hlp)

Description
Constructs a widget for the entry of measures position values.
The **allowed formats for user input** *are:*

- *The position reference frame may be chosen using the left hand button. The two formats are WGS84 and ITRF (World Geodetic System and International Terrestrial Reference Frame). The position quantity values should be either longitude (angle), latitude(angle) and height(length); or x,y,z (length). The allowed formats for lengths and angles are described in the quanta tool.*

- *An observatory location may be selected using the ObservatoryList button under the spanner/wrench.*

- *The name of Glish variable or function that evaluates to a position measure.*

Note that changing the reference frame **does not** *change the values in the entry boxes. Instead, it defines the frame with the currently specified values. If you wish to make a conversion, use the measures position gui.*
The options string can be used to specify vertically aligned entries.

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*options 'vertical'—"* *Allowed: string* *Default: "*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.epoch - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a epoch entry

Synopsis
epoch(parent, value, default, options, allowunset, editable, hlp)

Description
Constructs a widget for the entry of measures epoch values.
The **allowed formats for user input** *are:*

- *The current epoch may be selected using the Today button under the spanner/wrench.*
- *The name of Glish variable or function that evaluates to a measure.*

The default initial reference code is 'UTC'. This may be changed using the options argument.
Note that changing the reference frame **does not** *change the value in the entry box. Instead, it defines the frame with the currently specified value. If you wish to make a conversion, use the measures epoch gui.*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*Initial reference code* *Allowed: string* *Default: 'UTC'*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.direction - Function

Package display
Module widgets
Tool guientry

Make a widget for a direction entry

Synopsis
direction(parent, value, default, options, allowunset, editable, hlp)

Description
Constructs a widget for the entry of measures direction values.
The **allowed formats for user input** *are:*

- *The direction reference frame may be chosen using the left hand button, and the latitude and longitude entered in the next two entry boxes. The allowed formats for times and angles are described in the quanta tool.*
- *A planet may be selected using the left hand button*
- *One of a list of bright radio sources may be selected using the SourceList button under the spanner/wrench.*
- *The name of Glish variable or function that evaluates to a direction measure.*

Note that changing the reference frame **does not** *change the values in the entry boxes. Instead, it defines the frame with the currently specified values. If you wish to make a conversion, use the measures direction gui.*
The options string can be used to specify vertically aligned entries.

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*options: 'vertical'—"* *Allowed: string* *Default: "*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.quantity - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a quantity entry

Synopsis
quantity(parent, value, default, type, options, allowunset, editable, hlp)

Description
Constructs a widget for the entry of quantity values.
The **allowed formats for user input** *are:*

- *A quantity as returned by the quanta tool (*e.g. `dq.quantity('10GHz')`*)*
- *The name of Glish variable or function that evaluates to a quantity.*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
type	*Type of argument: unset means to evaluate* *Allowed: Variable* *Default: unset*
options	*options* *Allowed: string* *Default: ''*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.range - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a range entry

Synopsis
range(parent, value, default, options, allowunset, editable, rmin, rmax, rresolution, provideentry, hlp)

Description
Constructs a widget for the entry of range values.

The **allowed formats for user input** *are:*

- *Interactively by moving the slider*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Value or unset* *Default: 0*
default	*Default value* *Allowed: Value* *Default: 0*
options	*options* *Allowed: string* *Default: ”*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
rmin	*Minimum value* *Allowed: Number* *Default: 0.0*
rmax	*Maximum value* *Allowed: Number* *Default: 1.0*
rresolution	*Resolution of selection* *Allowed: Number* *Default: 0.1*
provideentry	*Include a type-in entry box as well* *Allowed: Bool* *Default: F*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.record - Function

Package **display**
Module **widgets**
Tool **guientry**

Make a widget for a Record entry

Synopsis
record(parent, value, default, options, allowunset, editable, hlp)

Description
Constructs a widget for the entry of record values.
The **allowed formats for user input** *are:*

- *A record (*e.g. `[height=640, width=480]`*)*
- *The name of Glish variable of function that evaluates to is a record.*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*options* *Allowed: string* *Default: ''*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.region - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a region entry

Synopsis
region(parent, value, default, options, allowunset, editable, hlp)

Description
Constructs a widget for the entry of region values.
The **allowed formats for user input** *are:*

- *A region as returned by the regionmanager tool (*e.g. `drm.box()`*)*
- *The name of Glish variable or function that evaluates to a region.*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*options* *Allowed: string* *Default: ''*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.string - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a string entry

Synopsis
string(parent, value, default, options, allowunset, editable, hlp)

Description
Constructs a widget for the entry of string values.
The **allowed formats for user input** *are:*

- *A string*
- *A set of strings, separated by newlines.*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*options* *Allowed: string* *Default: ''*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

---

guientry.untyped - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a untyped entry

Synopsis
untyped(parent, value, default, options, allowunset, editable, hlp)

Description
Constructs a widget for the entry of untyped values. The behavior requires some explanation:

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

The **allowed formats for user input** *are:*

- *Any valid glish expression*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Variable or unset*
default	*Default value* *Allowed: Variable*
options	*options* *Allowed: string* *Default: "*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

guientry.list - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for an extensible list of entry widget

Synopsis

list(parent, types, names, values, values, defaults, options, allowunset, editable, hlp)

Description

Constructs a widget for a list of guienty widgets. This allows one to construct a list of made up of elements each containing one or more widgets. One may insert and get the contents as usual.

*The type of each widget is specified in the types argument. The corresponding names are given in the names argument. These names determine the names of the fields in the record returned by get. The arguments for each widget are passed as records with the elements taken in sequence (*i.e. *the names are immaterial. allowunset and editable apply to all widgets.*

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
types	*Types of widgets* *Allowed: String of valid widget names*
names	*Name to be attached to each widget* *Allowed: Vector String*
values	*Values to insert on start* *Allowed: Record of variables or unset*
values	*Values to insert on start* *Allowed: Record of variables or unset*
defaults	*Default values* *Allowed: Record of Variables or unset*
options	*options* *Allowed: Record of vector strings* *Default: ''*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
hlp	*Popup help string* *Allowed: String* *Default: Unset*

Returns
guientry widget

Example

```
s := dge.list(f, types="string direction",
  names="Source Direction", name='Source',
  defaults=unset, values=[Source='No name', Direction=dm.direction()],
  allowunset=T, editable=T);
print s.get();
[Source1=[Source=M87, Direction=[m0=[value=187.705929, unit=deg], m1=[value=12.3
91125, unit=deg], type=direction, refer=J2000]], Source2=[Source=0J287, Directio
n=[m0=[value=133.703646, unit=deg], m1=[value=20.1085119, unit=deg], type=direct
ion, refer=J2000]], Source3=[Source=3C273B, Direction=[m0=[value=187.277917, uni
```

---

guientry.done - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Destroy this tool

Synopsis
done()

Returns
T

Description
Destroys this Guientry tool.

---

guientry.type - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Return the type of this tool

Synopsis
type()

Returns
String

Description
Returns 'guientry'

---

guientry.twoentry - Function

Package	**display**
Module	**widgets**
Tool	**guientry**

Make a widget for a range entry, providing two entry boxes next to each other labelled 'min' and 'max'. This is used by minmaxhist, and is not intended for independant use.

Synopsis
twoentry(parent, value, default, options, allowunset, editable, min, max, widths, hlp)

Description
Constructs a widget for the entry of a range.

The **allowed formats for user input** *are:*

- *The user enters numbers into the two entry boxes.*

Arguments

parent	*Parent of the widget* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Array* *Default: unset*
default	*Default value* *Allowed: Array* *Default: 0*
options	*options* *Allowed: string* *Default: ''*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
min	*Minimum value* *Allowed: Number* *Default: -1.0*
max	*Maximum value* *Allowed: Number* *Default: 1.0*
widths	*Width of the two boxes* *Allowed: Number* *Default: 0.1*
hlp	*Help message* *Allowed: string* *Default: F*

Returns
guientry widget

---

guientry.minmaxhist - Function

Package **display**
Module **widgets**
Tool **guientry**

Constructs a widget for the entry of a range.

Synopsis

minmaxhist(parent, value, default, allowunset, editable, minvalue, maxvalue, hlp, histarray, imageunits, options)

Description

Make a widget for selection of a range with the ability to allow the user to choose the range from a histogram (using histogramgui). The histogram is displayed and the user can drag, move and resize a region corresponding to the data min and max.

The **allowed formats for user input** *are:*

- *The user enters numbers into the two entry boxes. The user can open the histogram window, and select min and max using the mouse.*

Arguments

parent	*Parent of the widget* *Allowed: Tk frame*
value	*Value to insert on start* *Allowed: Array* *Default: unset*
default	*Default value* *Allowed: Array* *Default: 0*
allowunset	*Allow unset values* *Allowed: Boolean* *Default: T*
editable	*Is the entry editable by the user* *Allowed: Boolean* *Default: T*
minvalue	*Minimum value* *Allowed: Number* *Default: -1.0*
maxvalue	*Maximum value* *Allowed: Number* *Default: 1.0*
hlp	*Help message* *Allowed: string* *Default: ''*
histarray	*Record containing info to use for histogram (in .values and .counts)* *Allowed: Record* *Default: unset*
imageunits	*Brightness units of image (for labels)* *Allowed: string* *Default: unset*
options	*Options* *Allowed: string* *Default: ''*

Returns
T

1.4.16 guientrytest - Function

Package display
Module widgets

Test the construction of guientry widgets

Synopsis
guientrytest(autodestruct, editable, allowunset, expand)

Returns
Boolean

Description
This function makes a frame and constructs a number of entry widgets inside the frame.

Arguments

autodestruct	*Destroy the frame immediately?* *Allowed: Boolean* *Default: T*
editable	*Make the entries editable?* *Allowed: Boolean* *Default: T*
allowunset	*Allow unset values?* *Allowed: Boolean* *Default: T*
expand	*Expand policy* *Allowed: 'x'—'y'—'both'—'none'* *Default: 'none'*

1.4.17 helpmenu - Tool

Package display
Module widgets

Tk widget for a Help menu with the standard AIPS++ help items.
include "helpmenu.g"

Constructors

helpmenu Construct a "Help" menu

Functions

done Destroy the Help menu

reset *Reset the Help menu*

Description

A helpmenu is a "Help" menu inserted into an already existing frame. The menu contains any help items the user designates though the helpmenu arguments followed by all of the items required by the AIPS++ GUI Guidelines.

helpmenu - Constructor

Package **display**
Module **widgets**
Tool **helpmenu**

Construct a "Help" menu

Synopsis

helpmenu(parent, menuitems, refmanitems, callback, helpitems, widgetset)

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
menuitems	*User defined help items* *Allowed: string vector* *Default: unset*
refmanitems	*Reference manual locations for each menuitems* *Allowed: string vector* *Default: unset*
callback	*Callback function for non-reference manual menuitems* *Allowed: function* *Default: unset*
helpitems	*Additional popup help items* *Allowed: string vector* *Default: unset*
widgetset	*Widget set to use* *Allowed: A widgetserver closure tool* *Default: dws*

Returns

helpmenu widget

Description

Constructs a helpmenu according to the specified construction arguments and returns a ref to the widget. The "Help" menu will begin with each of the items in menuitems. These are followed by the standard AIPS++ help items as recommended by the GUI Guidelines. If menuitems is unset, then only these standard help items will appear in the menu. If there are menuitems, then there must be an equivalent number of refmanitems. Each refman item is either a string which is used as an argument to help or it is a zero-length string. If a refman item has zero-length, then the callback function is called when the corresponding menu item button is pressed. This callback function takes a single argument. When called, that argument will be the menu item which was pressed. Finally, you can specify any additional popup help text items. These are prepended to the standard popup help items. Each item appears as a separate line in the resulting popup help with the text "Help Menu" being both the short help and the label at the top of the long help containing the individual items.

Example

```
include 'helpmenu.g'
f := frame();
cb := function(whichItem) {
    print 'help callback called :',whichItem;
}
r := helpmenu(f, menuitems=['Dish','PGPlotter','Special help'],
              refmanitems=['Refman:dish','Refman:pgplotter',''],
              helpitems=['About Dish','AboutPGPlotter', 'Special help'],
      callback=cb);
```

When the "Dish" button is pressed in this example, the users browser will be directed to the dish entry in the Reference Manual. Similarly, selecting the "PGPlotter" button will display the pgplotter entry in the Reference Manual. However, when the "Special help" item is selected the cb function will be called and whichItem will have the value "Special help".

---

helpmenu.reset - Function

Package	**display**
Module	**widgets**
Tool	**helpmenu**

Reset the Help menu

Synopsis
reset(parent, menuitems, refmanitems, callback, helpitems, widgetset)

Returns
boolean

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
menuitems	*User defined help items* *Allowed: string vector* *Default: unset*
refmanitems	*Reference manual locations for each menuitems* *Allowed: string vector* *Default: unset*
callback	*Callback function for non-reference manual menuitems* *Allowed: function* *Default: unset*
helpitems	*Additional popup help text* *Allowed: string vector* *Default: unset*
widgetset	*Widget set to use* *Allowed: A widgetserver closure tool* *Default: dws*

Description
This has the same syntax as the constructor. It is intended to be used when there is a need to change the contents of the user-set portion of the "Help" menu after the menu has been made.

---

helpmenu.done - Function

Package display
Module widgets
Tool helpmenu

Destroy the Help menu

Synopsis
done()

Returns
boolean

Description
This function destroys the help menu. Each button is individually destroyed and all whenevers used by the widget are deactivated. The main "Help" menu button is also destroyed.

1.4.18 histogramgui - Tool

Package display
Module widgets

Window to display a histogram and allow selection of data min and max.
include "histogramgui.g"

Constructors

histogramgui	*Construct a histogramgui*

Functions

disable	*Not yet implemented.*
dismiss	*Removes the GUI from screen.*
done	*Destroys the widget and disables all whenevers.*
enable	*Not yet implemented.*
gui	*Build the GUI*
newdata	*Supply the widget with new data to use to plot the histogram.*
setselection	*Set a new selection to be shown on the histogram.*
setstats	*Set new image statistics*

General description

Description

This window is designed for use by a more complex GUI responsible for displaying an image. It takes an array of data, which it plots as a histogram, and allows the user to drag a region of the histogram, which the widget then fires as an event. Once the user has selected a region, they can resize or move the region. The widget has functions to allow the external chaging of a selection, and also the supply of different histogram data.
The controlling GUI is responsible for recalculating the histogram data etc when

required and also actually setting the selection it receives from the histogramgui window.

Graphical User Interface

Description

The main screen is shown below. Several options are available to the user via drop down option menus. The 'range' option menu allows selection of a percentage of pixels. Selecting 95% causes the central 95% of pixels to be selected. The other option for automatic selection is that of +/- n Sigma. This selects +/- n standard deviations away from the mean, or median (depending on selection). The other two option menus deal with the display options. Selecting from the 'scale' menu causes the count axis to be plotted logarithmically or linearly, depending on selection. The final user option allows for the selection of a different hightlight color to the default (yellow). The reset button resets all menus to their defaults, unselects the histogram and returns to the default zoom level.
Control of the plot region is handled by the mouse. Selection of a region is obtained by using the left (Button - 1) mouse button. Clicking and dragging will cause a region to be created. Once a region is created it can be:

- Moved, *by dragging the region once the mouse is inside it. The cursor will change to a cross when the region can be moved.*
- Resized (click), *by clicking outside the region. The region will extend to the location of the click. The mouse will appear as a normal arrow when the region can be resized via a click.*
- Resized (dragged), *by holding the mouse over the ends of the region, the region can be resized by dragging either of its end points. The cursor will change to a resizing arrow when over either end.*

Zooming is handled using the middle (Button - 2) mouse button. Clicking the middle button anywhere in the plot area will cause a zoom window to be opened. By moving the mouse after clicking, the box can be resized. Clicking the middle button again causes the plot area to zoom into the selected region. To unzoom, use ¡Control - Button - 2¿.
The scroll bar can be used to move along the x - axis. You can click and drag, or simply click to either side of the scroll tab.
Note: *All AIPS++ GUIs have popup help. Just rest your mouse cursor on top of a GUI element (e.g. a button) and a little frame will appear outlining the funtionality of this element. If you then press and hold down the right mouse button you will get a detailed description. The information given there should be sufficient to let you operate that GUI element.*

Events

The most important event emitted by histogramgui is the `change` *event. It's* `$value` *field contains an array (length 2) which contains the new selection choosen by the user. The window may also emit other events, depending on the user. If the user requests the use of standard deviation, mean or median, the histogram will emit a* `newstats` *event, the value field of which will contain a string array containing the desired statistics ('stddev', 'mean', 'median'). This indicates that an update is expected via the newstats function. The only other event emitted by the gui is a* `close` *event. This lets the controlling GUI know that calculation of the histogram etc is no longer required.*

Example

```
include 'histogramgui.g'

histwindow := histogramgui(xmin=-5, xmax=5,
   array=myHistogramArray,
   units='Jy/Beam');
histwindowgui := histwindow.gui();

whenever histwindow->change do {
# $value now contains an array representing the selection
        # made by the user.
print 'New selection is: ', $value;
}

whenever somethinglocal->happens do {
# Something locally changed the selection, so update the
# histogram window
histwindow.setselection($value)
}
```

histogramgui - Constructor

Package	**display**
Module	**widgets**
Tool	**histogramgui**

Construct a histogramgui

Synopsis

histogramgui(xmin, xmax, array, units, widgetset)

Arguments

xmin	*Start of selection to begin with* *Allowed: Number* *Default: 0*
xmax	*End of selection to begin with* *Allowed: Number* *Default: 1*
array	*Record containing array to plot* *Allowed: Record containing .values and .counts* *Default: unset*
units	*Units to use on labels* *Allowed: String* *Default: 'unknown'*
widgetset	*Widgetset to use* *Allowed: Widgetset* *Default: dws*

Returns
T

Description
Constructs a new histogram gui with the specified parameters (does not build the gui).

---

histogramgui.gui - Function

Package display
Module widgets
Tool histogramgui

Build the GUI

Synopsis
gui()

Description
Maps the GUI onto screen.

---

histogramgui.newdata - Function

Package	**display**
Module	**widgets**
Tool	**histogramgui**

Supply the widget with new data to use to plot the histogram.

Synopsis
newdata(array, xmin, xmax)

Arguments

array	*Record containing histogram data (.values and .counts)* *Allowed: Record* *Default: unset*
xmin	*New start of selection (min)* *Allowed: Number*
xmax	*New end of selection (max)* *Allowed: Number*

Description
This function supplies the widget with a new array from which to plot the histogram. It can also supply a new selection range.

---

histogramgui.setstats - Function

Package	**display**
Module	**widgets**
Tool	**histogramgui**

Set new image statistics

Synopsis
setstats(mean, median, stddev)

Arguments

mean	*Mean data value* *Allowed: Number* *Default: F*
median	*Median data value* *Allowed: Number* *Default: F*
stddev	*Standard deviation of data* *Allowed: Number* *Default: F*

Description
This function accepts new statistics to use for automatic selection of a region. The histogram window will wait for this function to be called after firing a 'newstats' event.

---

histogramgui.setselection - Function

Package	**display**
Module	**widgets**
Tool	**histogramgui**

Set a new selection to be shown on the histogram.

Synopsis
setselection(sel, forcenosel, forceselection)

Arguments

sel	*New selection* *Allowed: Array*
forcenosel	*Force the gui not to create or draw a selection* *Allowed: Boolean* *Default: F*
forceselection	*Force the gui to draw a selection* *Allowed: Boolean* *Default: F*

Description
This function accepts an array, from which it updates the region displayed on the histogram.

---

histogramgui.dismiss - Function

Package	**display**
Module	**widgets**
Tool	**histogramgui**

Removes the GUI from screen.

Synopsis
dismiss()

Description
This function unmaps the frame containing the GUI from the screen.

---

histogramgui.done - Function

Package	**display**
Module	**widgets**
Tool	**histogramgui**

Destroys the widget and disables all whenevers.

Synopsis
done()

Description
This function destroys the widget.

---

histogramgui.disable - Function

Package	**display**
Module	**widgets**
Tool	**histogramgui**

Not yet implemented.

Synopsis
disable()

Description
Not yet implemented, although sending an invalid histogram array will disable the window.

---

histogramgui.enable - Function

Package	**display**
Module	**widgets**
Tool	**histogramgui**

Not yet implemented.

Synopsis
enable()

Description
Not yet implemented, although if the window has been disabled by an invalid array, sending a new, valid array will enable it again.

1.4.19 listboxcontainer - Tool

Package	**display**
Module	**widgets**

A scrolllistbox with an underlying itemcontainer.
include "listboxcontainer.g"

Constructors

listboxcontainer	*Construct a listboxcontainer*

Functions

clear	*(EVENT) clear selections*
delete	*(EVENT) delete a name-value pair*
done	*Destroy ourselves*

fromforeign	*insert items from container or record*
get	*(EVENT) return the value of the item specified*
hasitem	*check if item is present in listboxcontainer*
hscrollbar	*(EVENT) Turn on or off the horizontal scrollbar*
insert	*(EVENT) Insert a name-value pair or record elements, or overwrite values of existing pairs*
itemnames	*Get the names of all items.*
see	*(EVENT) scroll to the specified item*
seeoninsert	*(EVENT) See the end after each insert?*
select	*(EVENT) make selection*
selection	*(EVENT) return a record of items selected*
vscrollbar	*(EVENT) Turn on or off the vertical scrollbar*

Description

The listboxcontainer widget is a `scrolllistbox` *widget. It has some similarity to the interface of the* `scrolllistbox` *widget. The difference to the original listbox and scrolllistbox is, that this widgets is a wrapper for an itemcontainer. Everything inserted into the widget is stored in an itemcontainer as a name-value pair (an item). The name is displayed in the widget. Items can be accessed by name or by index. One has to be careful if using indices, because the listboxcontainer is adjusting itself (eg. deleting an item pushes all other items up by 1). Another difference to the standard* `listbox` *and* `scrolllistbox` *is the indexing. Indices are integers not strings! Indexing starts at "0" as opposed to the* `itemcontainer`*, where indexing starts the "Glish" way at "1".*
The listboxcontainer widget supports all scrolllistbox widget events. For the events "background bind borderwidth exportselection font height hscrollbar mode relief seeoninsert view vscrollbar vscrollview width" see the documentation on `scrolllistbox` *and* `listbox`*.*
All events which are emitted by the underlying `listbox` *are forwarded and will be seen to be emitted by the listboxcontainer.*
Note that most of the functions listed here are actually events.

Example

```
include 'listboxcontainer.g'
f := frame()
lbc := listboxcontainer(f)
lbc->insert('x',1)
lbc->insert('y',2)
# you want to add a record
r := [a=111,b=222,c=333]
lbc->insert(r)
# you want to get the current selection(s), if any
```

```
s := lbc->selection();
#you want to delete 'x' and 'y' by name
lbc->delete('x','y')
#or by index
lbc->delete(0:1)
# to retrieve the value bound to name 'a'
a := l->get('a')
```

listboxcontainer - Constructor

Package	**display**
Module	**widgets**
Tool	**listboxcontainer**

Construct a listboxcontainer

Synopsis

listboxcontainer(parent, container, hscrollbar, vscrollbar, vscrollbarright, seeoninsert, width, height, mode, font, relief, borderwidth, foreground, background, exportselection, fill, widgetset)

Arguments

parent	*Parent of the listboxcontainer* *Allowed: Tk frame*
container	*Create from an itemcontainer* *Allowed: itemcontainer* *Default: unset*
hscrollbar	*Show a horizontal scrollbar?* *Allowed: Boolean* *Default: T*
vscrollbar	*Show a vertical scrollbar?* *Allowed: Boolean* *Default: T*
vscrollbarright	*Show vertical scrollbar on right? (F=left)* *Allowed: Boolean* *Default: T*
seeoninsert	*See the end after each insert?* *Allowed: Boolean* *Default: T*
width	*Width in characters units* *Allowed: Integer* *Default: 20*
height	*Height in characters units* *Allowed: Integer* *Default: 6*
mode	*Listbox mode* *Allowed: String from 'browse', 'single', 'extended', 'multiple'* *Default: 'browse'*
font	*Font of text* *Allowed: X font* *Default: ''*
relief	*Edge relief* *Allowed: String from 'flat', 'ridge', 'raised', 'sunken', 'groove'* *Default: 'sunken'*
borderwidth	*Border width* *Allowed: Integer* *Default: 2*
foreground	*Color of text* *Allowed: X color* *Default: 'black'*
background	*Background color* *Allowed: X color* *Default: 'lightgrey'*
exportselection	*export to X clipboard?* *Allowed: Boolean* *Default: F*
fill	*how to expand when resized* *Allowed: String from 'x', 'y', 'both', 'none'* *Default: 'both'*
widgetset	*Widget set to use* *Allowed: A widgetserver closure tool* *Default: dws*

Returns
listboxcontainer widget

Description
Constructs a listboxcontainer according to the specified construction arguments, and returns a ref to the widget.

---

listboxcontainer.done - Function

Package	**display**
Module	**widgets**
Tool	**listboxcontainer**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This function destroys the widget.

---

listboxcontainer.hscrollbar - Function

Package	**display**
Module	**widgets**
Tool	**listboxcontainer**

(EVENT) Turn on or off the horizontal scrollbar

Synopsis
hscrollbar(torf)

Arguments

torf	*Show a horizontal scrollbar?* *Allowed: Boolean*

Description
This is an event which turns on or off the horizontal scrollbar.

Example

```
# lbc is a previously created listboxcontainer
# turn off the horizontal scrollbar
lbc->hscrollbar(F);
# turn it back on
lbc->hscrollbar(T);
```

---

listboxcontainer.vscrollbar - Function

Package display
Module widgets
Tool listboxcontainer

(EVENT) Turn on or off the vertical scrollbar

Synopsis
vscrollbar(torf)

Arguments

torf	*Show a vertical scrollbar?* *Allowed: Boolean*

Description
This is an event which turns on or off the vertical scrollbar. If turned on, the constructor argument 'vscrollbarright' determines if the scrollbar is placed on the left or right side of the listbox.

---

listboxcontainer.seeoninsert - Function

Package	**display**
Module	**widgets**
Tool	**listboxcontainer**

(EVENT) See the end after each insert?

Synopsis

seeoninsert(torf)

Arguments

torf	*See the end after each insert?* *Allowed: Boolean*

Description

This event turns on or off the action which this widget does after each insert event. When on (T) a "see" event is sent to the listbox to show the end of the listbox after each inset event. When off (T) no such "see" event is sent.

Example

```
# slb is a previously created listboxcontainer
# don't see the end after each insert event
lbc->seeoninsert(F);
# see the after each insert event
lbc->seeoninsertT);
```

---

listboxcontainer.fromforeign - Function

Package	**display**
Module	**widgets**
Tool	**listboxcontainer**

insert items from container or record

Synopsis
fromforeign(container)

Arguments

container	*Itemcontainer or record to put into listboxcontainer* *Allowed: itemcontainer or record*

Description
Insert an itemcontainer or record into an (empty) listboxcontainer. This function overwrites existing contents.

---

listboxcontainer.hasitem - Function

Package display
Module widgets
Tool listboxcontainer

check if item is present in listboxcontainer

Synopsis
hasitem(item)

Arguments

item	*Item to be checked if present* *Allowed: string*

Returns
A Boolean

Description
Equivalent to has_item in `itemcontainer`.

---

listboxcontainer.itemnames - Function

Package display
Module widgets
Tool listboxcontainer

Get the names of all items.

Synopsis
itemnames()

Returns
A vector of strings

Description
Equivalent to item_names in `itemcontainer`.

---

listboxcontainer.clear - Function

Package	**display**
Module	**widgets**
Tool	**listboxcontainer**

(EVENT) clear selections

Synopsis
clear(namesorindices)

Arguments

namesorindices	*names or indices or "all"* *Allowed: vector of strings or integers*

Description
This **event** *clears the given selections. This can be done by name or index. Vectors are accepted. "all" deselects all selections.*

---

listboxcontainer.delete - Function

Package	**display**
Module	**widgets**
Tool	**listboxcontainer**

(EVENT) delete a name-value pair

Synopsis
delete(namesorindices)

Arguments

namesorindices	*names or indices* *Allowed: vector of strings or integers*

Description
This **event** *deletes the given items from the listbox. This can be done by name or index. Vectors are accepted.* **Note:** *The functionality varies from* `listbox`*!*

Example

```
# by name
lbc->delete('test');
# or by index if test is the first(=0) item in the listboxcontainer.
lbc->delete(0)
```

---

listboxcontainer.get - Function

Package display
Module widgets
Tool listboxcontainer

(EVENT) return the value of the item specified

Synopsis
get(nameorindex)

Arguments

nameorindex	*name or index* *Allowed: string or integer*

Returns
any

Description
The items selected in the `listbox` *are returned as a record.*

---

listboxcontainer.insert - Function

Package	**display**
Module	**widgets**
Tool	**listboxcontainer**

(EVENT) Insert a name-value pair or record elements, or overwrite values of existing pairs

Synopsis
insert(stringorrecord,)

Arguments

stringorrecord	*Allowed: string or record*
	only if first argument is string *Allowed: any*

Description
This **event** *inserts a new item into the listboxcontainer or replaces the value of an existing one.* **Note:** *The functionality varies from* `listbox`*!*

Example

```
# insert
- lbc->insert('test',55)
- lbc->insert('dummy',[xyz='hello'])
- print l->get('dummy');
[xyz=hello]
# overwrite
- lbc->insert('dummy',1000);
- print l->get('dummy');
1000
```

---

listboxcontainer.see - Function

Package	**display**
Module	**widgets**
Tool	**listboxcontainer**

(EVENT) scroll to the specified item

Synopsis
see(nameorindex)

Arguments

nameorindex	*name or index* *Allowed: string or integer*

Description
See see *in scrolllistbox.*

---

listboxcontainer.select - Function

Package	**display**
Module	**widgets**
Tool	**listboxcontainer**

(EVENT) make selection

Synopsis
select(namesorindices)

Arguments

namesorindices	*names or indices* *Allowed: vector of strings or integers*

Description

This **event** *selects the given items. This can be done by name or index. Vectors are accepted.*

---

listboxcontainer.selection - Function

Package	**display**
Module	**widgets**
Tool	**listboxcontainer**

(EVENT) return a record of items selected

Synopsis
selection()

Returns
record

Description
The items selected in the `listbox` *are returned as a record.*

1.4.20 multiautogui - Tool

Package	**display**
Module	**widgets**

Tool to manage a number of autoguis
include "multiautogui.g"

Description

This `tool` *is designed for use when context sensitive menus are required. For example, it was implemented for use with annotations, since clicking on a new shape should bring up options for that shape. This allows different "types" of autoguis to be built, one for each type of shape. The autoguis are built only once, since construction of them takes a significant amount of time. Once built they are mapped, and unmapped from the screen as needed. This widget also attempts to make accessing multiple autoguis simpler, by allowing "nested" options within the*

records passed to / from it. For example, you can construct a `multiautogui`*, and have it generate multiple autoguis, all with their parameters filled, by forming the* `fillgui` *record correctly. Most of the work is still done by the underlying autoguis.*

Parameter description *In a similar manner to autoguis, most transactions occur via "parameter records". In addition to this are control functions, which control the mapping to and from the screen of the different autoguis. The actual records destined for the desired autogui are nested one level down. For example, if two autoguis are desired, "someoptions" and "otheroptions" and you wish them both to occupy the same spot on screen using a multiautogui, you would create a parameter record as follows, construct the widget, and then choose which one you wanted visible:*

- `myoptionrecord:`
  - `someoptions:` *This is the name you will use to refer to operations on this autogui, and also the name which the multiautogui will return new options for this set under*
    - `options:` *A standard option set for an autogui*
  - `otheroptions:` *This is just some other name.*
    - `options:` *Another different standard option set for an autogui*

```
include 'multiautogui.g'

mymulti := multiautogui(myoptionrecord, title = 'Test Multi');

mymulti.show('otheroptions');
```

For a complete example, see multiautoguitest in multiautogui.g . A similar technique is used to fill the GUIs, and also to receive options from them. **Event description**

Basically, anytime one of the autoguis being maintained by multiautogui fires a `setoptions` *event, it is passed on by multiautogui. The difference is multiautogui stores the parameters returned by the autogui in a record which has the name of the parameter set. The example above, in which we created two autoguis called "someoptions" and "otheroptions" provides a good way to demostrate this. We imagine that "someoptions" was visible, and the user changed a parameter. Multiautogui would emit a* `setoptions` *event, which contained a record with one field. The field name would be the originating autogui (in this case "someoptions"), and contained within that subrecord would be the new options. Similarly, when a* `newuserchoicelist` *is emitted (in fact, all events (*) are forwaded on in this manner), the same "parameter encoding" occurs, with the option record being stored in a subrecord, the name of which is determined by the originating autogui.*

Example
For a simple example, please refer to multiautogui.g. It contains a subsequence "multiautoguitest" which demonstrates multiautogui managing three autoguis.

multiautogui - Constructor

Package	**display**
Module	**widgets**
Tool	**multiautogui**

Create a multiautogui `tool`

Synopsis
multiautogui(params, title, toplevel, actionlabel, autoapply, expand, widgetset)

Arguments

params	*Record to define multiple autoguis* *Allowed: Record*
title	*title string if top-level frame constructed internally* *Allowed: String* *Default: 'autogui'*
toplevel	*top-level parent frame* *Allowed: frame or boolean* *Default: F*
actionlabel	*label for action button (passed to each autogui) TODO: It would be nice to have a common "action" button* *Allowed: string or boolean* *Default: F*
autoapply	*Show auto-apply buttons* *Allowed: boolean* *Default: T*
expand	*Expand policy* *Allowed: 'x'—'y'—'both'—'none'* *Default: 'none'*
widgetset	*widgetserver* `tool` *to use to build the GUI* *Allowed: widgetserver* `tool` *Default: dws*

Returns
subsequence

Description

This is the only constructor for the multiautogui `tool`*. The first argument,* `params`*, contains the various parameters which will be used to generate the multiple GUIs. If a parent frame is not given in the* `toplevel` *argument, a top-level frame will be created with title as given in the* `title` *argument. If the* `actionlabel` *argument is given as a string, then a button will be placed at the bottom of each of the GUIs with that string as its label. Pressing this button will cause the autogui* `tool` *to emit all parameters and their values via a* `setoptions` *event. The final two arguments just provide some future flexibility with respect to widgetserver and guientry* `tools`*.*

---

multiautogui.shownone - Function

Package	**display**
Module	**widgets**
Tool	**multiautogui**

Show no autoguis in the frame (unmap all)

Synopsis
shownone()

Returns
T

Description
This function unmaps all autoguis, so none are visible.

---

multiautogui.show - Function

Package	**display**
Module	**widgets**
Tool	**multiautogui**

Read the GUI and return the current settings

Synopsis
show(which)

Arguments

which	*which autogui to show* *Allowed: String - name of a created autogui*

Returns
T if gui name found, else F

Description

This function allows one of the autoguis to be shown. The name supplied here should be the name of one of the created autoguis.

---

multiautogui.fillgui - Function

Package display
Module widgets
Tool multiautogui

Fill one or all of the autoguis

Synopsis
fillgui(options)

Arguments

options	*options to fill autgui(s)* *Allowed: Record to use to fill autogui(s)*

Returns
T

Description
This function is used to fill the autoguis. The parameter must be a record whose field names correspond to one or more autoguis. Within these fields should be the options with which to fill the respective autogui. If one of the field names does not exist as an autogui, its contents are used to create a new one.

---

multiautogui.addautogui - Function

Package	**display**
Module	**widgets**
Tool	**multiautogui**

Add another autogui to the widget

Synopsis
addautogui(name, params)

Arguments

name	*name of the new autogui* *Allowed: String*
params	*parameter to fill autogui* *Allowed: Record with autogui information*

Returns
T if addition successful, else F

Description

This function can be used to add a new autogui to the collection. It takes a name (which will from then on be used as an identifier in all option records) and a set of parameters which will be used to construct the new autogui.

---

multiautogui.multiautoguitest - Function

Package	**display**
Module	**widgets**
Tool	**multiautogui**

Test the multiautogui

Synopsis
multiautoguitest()

Returns

T

Description

This function provides a simple test of the multiautogui.

1.4.21 multiscale - Tool

Package display
Module widgets

Glish/Tk widget to specify multiple values in a single domain
include "multiscale.g"

Constructors

multiscale	*Construct a multiscale*

Functions

disable	*Disable widget*
dismiss	*Disable and dismiss the multiscale*
enable	*Enable widget*
getvalues	*Retrieve the current scale value/s*
setrange	*Change the lower and upper limits of the scales*
setresolution	*Set the step value for the scales*
setvalues	*Set the scale value/s*

Description

Note: the following information describes what the multiscale widget *should* do. At present, only one scale is supported.
This Glish/Tk widget provides one or more basic Glish/Tk scale widgets having identical value ranges, but different values. Its intelligence lies in its ability to ensure that values remain ordered. That is, it can be used for the selection of minimum and maximum values of data, and ensures that the minimum is always less than or equal to the maximum.
Whenever the user modifies any one of the scales in the multiscale widget, an event "values" is emitted from the multiscale, whose value is a vector of floats being the position of each slider.
If you chooser to show an entry box as well (`entry=T`*) as the slider, then they are*

connected. The values selected with the slider show in the entry box. When you enter a value in the entry box, after you press carriage return, the slider will update its value.
You can also interactively extend the range of the slider (`extend=T` *and* `entry=T`*). In this case, a check box is offered. If checked, then any value you enter in the entry box which is outside of the current slider range, will extend the slider range.*

multiscale - Constructor

Package	**display**
Module	**widgets**
Tool	**multiscale**

Construct a multiscale

Synopsis
multiscale(parent, start, end, values, names, helps, constrain, entry, extend, length, resolution, orient, width, font, relief, borderwidth, foreground, background, fill, widgetset)

Arguments

parent	*Parent of the label* *Allowed: Tk frame*
start	*Lower limit of range* *Allowed: float* *Default: 0.0*
end	*Upper limit of range* *Allowed: float* *Default: 100.0*
values	*initial values* *Allowed: vector of floats* *Default: [0.0, 100.0]*
names	*text labels for each scale* *Allowed: vector of strings* *Default: ['Minimum', 'Maximum']*
helps	*popup help text for each label* *Allowed: vector of strings* *Default: ""*
constrain	*whether to enforce ordering of numbers* *Allowed: T F* *Default: T*
entry	*provide an entry box as well* *Allowed: T F* *Default: F*
extend	*extend the scale if entered value beyond current extrema* *Allowed: T F* *Default: F*
length	*length of the scale widgets* *Allowed: dimension* *Default: 110*
resolution	*step value along scales* *Allowed: float* *Default: 1.0*
orient	*orientation of scale* *Allowed: 'horizontal' 'vertical'* *Default: 'horizontal'*
width	*transverse width of the scales* *Allowed: dimension* *Default: 15*
font	*text font* *Allowed: X font* *Default: "*
relief	*border relief* *Allowed: 'flat' 'ridge', 'raised' 'sunken' 'groove'* *Default: 'flat'*
borderwidth	*border width* *Allowed: dimension* *Default: 2*
foreground	*color of text* *Allowed: X color* *Default: 'black'*
background	*background color* *Allowed: X color* *Default: 'lightgrey'*
fill	*how to expand when resized* *Allowed: 'x' 'y' 'both' 'none*

Returns
multiscale widget

Description
Constructs a multiscale according to the specified construction arguments, and returns a ref to the widget.

Example

```
include 'multiscale.g'
f := frame();
ms := multiscale(f, start=20.0, end=55.0, resolution=0.2);
```

---

multiscale.disable - Function

Package	**display**
Module	**widgets**
Tool	**multiscale**

Disable widget

Synopsis
disable()

Returns
T

Description
This function disables the widget.

Example

```
f := frame();
ms := multiscale(f);
ms.disable();
```

---

multiscale.enable - Function

Package	**display**
Module	**widgets**
Tool	**multiscale**

Enable widget

Synopsis
enable()

Returns
T

Description
This function enables the widget.

Example

```
f := frame();
ms := multiscale(f);
ms.disable();
ms.enable();
```

---

multiscale.getvalues - Function

Package	**display**
Module	**widgets**
Tool	**multiscale**

Retrieve the current scale value/s

Synopsis
getvalues()

Returns
vector of floats

Description
This function is used to retrieve the value of each of the scales. A vector of floats is returned.

Example

```
- ms := multiscale(spareframe);
- print ms.getvalues();
[0 100]
```

---

multiscale.setrange - Function

Package	**display**
Module	**widgets**
Tool	**multiscale**

Change the lower and upper limits of the scales

Synopsis
setrange(start, end)

Arguments

start	*lower limit for scale/s* *Allowed: float* *Default: Current value*
end	*upper limit for scale/s* *Allowed: float* *Default: Current value*

Returns
none

Description
This function can be used to change the range of the scale/s after construction of the widget.

Example

```
ms := multiscale(spareframe);
ms.setrange(-30.0, 30.0);
```

---

multiscale.setresolution - Function

Package	**display**
Module	**widgets**
Tool	**multiscale**

Set the step value for the scales

Synopsis
setresolution(resolution)

Arguments

resolution	*step size for scales* *Allowed: float* *Default: 1.0*

Returns
none

Description
This function should be used to modify the step size on the scale/s after construction of the widget.

Example

```
ms := multiscale(spareframe);
ms.setresolution(0.01);
```

---

multiscale.setvalues - Function

Package	**display**
Module	**widgets**
Tool	**multiscale**

Set the scale value/s

Synopsis
setvalues(values)

Arguments

values	*values for the scales* *Allowed: vector of floats*

Returns
none

Description
This function should be used to change the values shown on the scale/s after construction of the widget.

Example

```
ms := multiscale(spareframe, names="bass mid treble",
                 values=[50, 50, 50], constrain=F,
                 orient='vertical', labelside='top');
ms.setvalues([10, 40, 60]);
```

---

multiscale.dismiss - Function

Package display
Module widgets
Tool multiscale

Disable and dismiss the multiscale

Synopsis
dismiss()

Returns
None

Description
This provides a tidy way to remove a multiscale once it is finished with.

Example

```
f := frame();
ms := multiscale(f);
# ... things happen here ...
ms.dismiss();
```

1.4.22 menuframes - Tool

Package display
Module widgets

Tk widget to hide and expose a number of embedded frames on demand include "menuframes.g"

Constructors

menuframes	*Construct a menuframes*

Functions

Delete	*Delete a named frame*
add	*Add a named frame and return the frame*
disable	*Disable the menuframes*
done	*Destroy ourselves*
enable	*Enable the menuframes*
state	*Return the state of the frames in a record*
switch	*Switch the visible state of the menuframes*

Description
This widget handles switching between a number of frames, controlled by a menu button. The widget is created with a parent frame and a specified menu button. Named frames may be added using the `add` *function. Switching between these frames is possible either by issuing* `switch` *and* `delete` *commands on named frames, or by user interaction via the menu button.*

Example

```
tmenuframes := function(exclusive=T) {

  prvt := [=];

  prvt.frame:=dws.frame(side='top');
  prvt.topmenubar := dws.frame(prvt.frame,side="left", relief="raised",
     expand="x");
  prvt.filebutton := dws.button(prvt.topmenubar, "File",
      relief="flat", type="menu");
  prvt.filebutton.shorthelp := 'Do various operations';
  prvt.filemenu := [=];

  prvt.filemenu['exit'] := dws.button(prvt.filebutton, 'Exit',
      type='halt');
  prvt.filemenu['exit'].shorthelp := 'Exit this window: the tool is done and the windo

  prvt.windowbutton := dws.button(prvt.topmenubar, 'Windows',
  relief='flat', type='menu');

  prvt.rightmenubar := dws.frame(prvt.topmenubar,side='right');

  prvt.helpmenu := dws.helpmenu(prvt.rightmenubar);

  prvt.windowmenu := menuframes(prvt.frame, prvt.windowbutton, exclusive=exclusive);

  myframe:=dws.frame(prvt.windowmenu.add('My frame'), relief='sunken');
  myframe2:=dws.frame(prvt.windowmenu.add('My frame #2'), relief='sunken');

  mylabel := dws.label(myframe, 'First frame');
  done := dws.button(myframe, 'Done', type='halt');
  whenever done->press do {
    prvt.windowmenu.delete('My frame')
    deactivate;
  }
  mylabel2 := dws.label(myframe2, 'Second frame');
  dismiss := dws.button(myframe2, 'Dismiss', type='dismiss');
  whenever dismiss->press do {
    prvt.windowmenu.switch('My frame #2', on=F);
  }

  prvt.windowmenu.switch('My frame');

  prvt.bottomframe := dws.frame(prvt.frame, side='right');
  prvt.donebutton := dws.button(prvt.bottomframe, 'Done', type='halt');
```

```
  whenever prvt.donebutton->press do {
    prvt.windowmenu.done();
    prvt.frame->unmap();
    val prvt := F;
  }

  return T;
}
```

menuframes - Constructor

Package	**display**
Module	**widgets**
Tool	**menuframes**

Construct a menuframes

Synopsis
menuframes(parent, menubutton, exclusive, widgetset)

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
menubutton	*Menu button* *Allowed: Tk button*
exclusive	*Show only one frame at once?* *Allowed: Boolean* *Default: T*
widgetset	*Widget set to use* *Allowed: A widgetserver closure tool* *Default: dws*

Returns
menuframes widget

Description
Constructs a menuframes according to the specified construction arguments, and returns a ref to the widget.

Example

```
include 'menuframes.g'
f := frame();
f1 := frame();
mb := button(f1, 'Windows', type='menu')
r := menuframes(f, mb);
```

---

menuframes.add - Function

Package	**display**
Module	**widgets**
Tool	**menuframes**

Add a named frame and return the frame

Synopsis
add(title)

Returns
Tk frame

Description
This adds a frame by name and returns the actual frame. The name is that by which the user selects the frame in the menu button.

Arguments

title	*Title of the frame to be added* *Allowed: String*

Example

```
include 'menuframes.g'
f := frame();
f1 := frame();
mb := button(f1, 'Windows', type='menu')
r := menuframes(f, mb);
f1 := r.add('Frame 1')
f2 := r.add('Frame 2')
f3 := frame(f1, relief='sunken');
```

---

menuframes.Delete - Function

Package	**display**
Module	**widgets**
Tool	**menuframes**

Delete a named frame

Synopsis
Delete(title)

Returns
Tk frame

Description
This deletes a frame by name.

Arguments

title	*Title of the frame to be deleted* *Allowed: String*

Example

```
include 'menuframes.g'
f := frame();
f1 := frame();
mb := button(f1, 'Windows', type='menu')
r := menuframes(f, mb);
f1 := r.add('Frame 1')
f2 := r.add('Frame 2')
f3 := frame(f1, relief='sunken');
r.delete('Frame 1')
```

---

menuframes.switch - Function

Package	**display**
Module	**widgets**
Tool	**menuframes**

Switch the visible state of the menuframes

Synopsis
switch(title, on)

Returns
Bool

Description
A call to this function will flip the visibility of a given frame.

Arguments

title	*Title of the frame to be flipped* *Allowed: String*
on	*Should the state be on?* *Allowed: Boolean* *Default: T*

Example

```
r := menuframes(f, mb);
f1:=r.add('Frame 1');
f2:=r.add('Frame 2');
r.switch('Frame 1');
```

---

menuframes.disable - Function

Package	**display**
Module	**widgets**
Tool	**menuframes**

Disable the menuframes

Synopsis
disable()

Returns
Bool

Description
A call to this function disables the menuframes.

Example

```
r := menuframes(f, mb);
# ...
r.disable();
```

---

menuframes.enable - Function

Package	**display**
Module	**widgets**
Tool	**menuframes**

Enable the menuframes

Synopsis
enable()

Returns
Bool

Description
A call to this function enables the menuframes.

Example

```
r := menuframes(f, mb);
# ...
r.enable();
```

---

menuframes.done - Function

Package	**display**
Module	**widgets**
Tool	**menuframes**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This tool function destroys the widget.

Example

```
- r := menuframes(f, mb);
- r.done();
T
- r
F
-
```

---

menuframes.state - Function

Package	**display**
Module	**widgets**
Tool	**menuframes**

Return the state of the frames in a record

Synopsis
state()

Returns
Record

Description
This tool function returns the state of the frames in a record.

1.4.23 messageline - Tool

Package display
Module widgets

Tk widget to display and log one-line messages.
include "messageline.g"

Constructors

messageline	*Construct a messageline*

Functions

clear	*Clear the contents of the widget*
done	*Destroy the widget*
post	*Post a message to the widget and the logger*
postnoforward	*Post a message to the widget*

Description

This simple Tk widget is a place for a complex GUI to display one-line messages (typically on the status of the task being controlled by the GUI) and simultaneously log those messages to the AIPS++ logger via the note function. The messageline widget is in many ways just like an entry widget. The messageline widget supports these entry widget events: background, bind, borderwidth, exportselection, font, foreground, justify, relief, view, and width. These events are all forwarded directly to the underlying entry widget. In addition, the following three events control the displayed text:

post *This is what you use to display a message and send it off to the logger. It takes a single argument, the message.*

postnoforward *Use this to display a message which should not be logged. It takes a single argument, the message.*

clear *This clears the message.*

All events emitted by the underlying entry widget are re-emitted by this widget. These events will be any events due to binds set through the bind event plus the xscroll event. The xscroll event, along with the view event, allows a horizontal scrollbar to be associated with this widget just as you would the entry widget. The widget can be destroyed using the done function.

Example

```
include 'messageline.g'
f := frame()
ml := messageline(f)
ml->post('This message is displayed and logged.')
ml->postnoforward('This message is just displayed.')
ml->clear() # the widget is now cleared
# optionally add a scrollbar, if you think the messages might get long
sb := scrollbar(f, orient='horizontal');
whenever sb->scroll do {ml->view($value);}
whenever ml->xscroll do {sb->view($value);}
```

messageline - Constructor

Package	**display**
Module	**widgets**
Tool	**messageline**

Construct a messageline

Synopsis

messageline(parent, width, justify, font, relief, borderwidth, foreground, background, exportselection, hlp, messagenote, widgetset)

Arguments

parent	*Parent of the messageline* *Allowed: Tk frame*
width	*Width in characters units* *Allowed: Integer* *Default: 30*
justify	*Justification of text* *Allowed: String from 'center', 'left', 'right'* *Default: 'left'*
font	*Font of text* *Allowed: X font* *Default: ''*
relief	*Edge relief* *Allowed: String from 'flat', 'ridge', 'raised', 'sunken', 'groove'* *Default: 'sunken'*
borderwidth	*Border width* *Allowed: Integer* *Default: 2*
foreground	*Color of text* *Allowed: X color* *Default: 'black'*
background	*Background color* *Allowed: X color* *Default: 'lightgrey'*
exportselection	*export to X clipboard?* *Allowed: Boolean* *Default: T*
hlp	*Short popuphelp text* *Allowed: String* *Default: No help*
messagenote	*The note function to use* *Allowed: A note function* *Default: note*
widgetset	*Widget set to use* *Allowed: A widgetserver closure tool* *Default: dws*

Returns
messageline widget

Description
Constructs a messageline according to the specified construction arguments, and returns a ref to the widget.

---

messageline.clear - Function

Package	**display**
Module	**widgets**
Tool	**messageline**

Clear the contents of the widget

Synopsis
clear()

Description
This is an event. When sent to this widget the contents of the widget will be cleared.

---

messageline.done - Function

Package	**display**
Module	**widgets**
Tool	**messageline**

Destroy the widget

Synopsis
done()

Returns
boolean

Description
This function destroys the widget. It will be unusable after this function has been called.

---

messageline.post - Function

Package	**display**
Module	**widgets**
Tool	**messageline**

Post a message to the widget and the logger

Synopsis
post()

Description
This is an event. When sent to this widget the value of this event (a string) will be displayed in the widget and sent to the default logger via note.

---

messageline.postnoforward - Function

Package	**display**
Module	**widgets**
Tool	**messageline**

Post a message to the widget

Synopsis
postnoforward()

Description
This is an event. When sent to this widget the value of this event (a string) will be displayed in the widget without forwarding the message on to the logger.

1.4.24 optionmenu - Tool

Package	**display**
Module	**widgets**

Tk widget where the menu label changes with selection
include "optionmenu.g"

Constructors

optionmenu	*Construct an optionmenu*

Functions

disabled	*Disable or enable the optionmenu widget*
done	*Destroy ourselves*
extend	*Extend the menu with more items*
findlabel	*Find the index of the item with the given label*
findname	*Find the index of the item with the given name*
findvalue	*Find the index of the item with the given value*
geteventmask	*Get the event mask*
getindex	*Get the index of the current selection*
getlabel	*Get the label of the current selection*
getlabels	*Gets all of the labels associated with the menu*
getname	*Get the name of the current selection*
getnames	*Gets all of the names associated with the menu*
getpreviousindex	*Get the index previous to the current selection*
getvalue	*Get the value of the current selection*
getvalues	*Gets all of the values associated with the menu*
isenabled	*Is the optionmenu widget enabled ?*
replace	*Replace the menu with the provided items*
selectindex	*Select an item in the menu by index*
selectlabel	*Select an item in the menu by label*
selectname	*Select an item in the menu by name*
selectvalue	*Select an item in the menu by value*
setbackground	*Set the color of the background of the optionmenu button*
seteventmask	*Set the event mask*
setforeground	*Set the color of the foreground of the optionmenu button*
setlabel	*Set the label of the optionmenu*
setwidth	*Set the width of the optionmenu button*

Description

This Tk widget enables you to create a flat (i.e. no submenus) menu in a specified frame. You can associate a label (that which appears on the top menu button), a name (that which appears in the menu list) and a value (whatever you like, it may be different from the label) for each menu item. When you select one of the items in the menu, the label of the menu button changes to reflect the label of the selected item.

If your list of items is very long, a menu may be too long to sensibly manage. Therefore, you can optionally choose (see constructor) to use a popup scrolling listbox rather than a menu (it's really implemented with a selectabelist widget). This widget can be treated like any of the built in Tk widgets, and has almost all of the same construction arguments as the button Tk widget.

Note that the `width` *argument can be used in three ways. If* `width=-1` *(the default), the menu button will be wide enough (and of fixed width) to accomodate all of the item labels in the menu. This means that the GUI won't jitter when you write different length labels into it. If* `width=0`*, then the menu button will change*

its width to accomodate whatever label is written into it. If `width > 0`*, then the button is again of fixed width, with the width that you specify.*
Whenever a selection is made, this widget emits an event called `select`*. In the $value of this event, you can find a record with fields* `label`, `name`, `value`, *and* `index` *(index of item in menu) containing that information for the selected item. The event generation can be disabled if you desire with the seteventmask tool function.*
Whenever the menu is replaced with the replace tool function, this widget emits an event called `replaced` *(it has no value). Note that the tool function extend invokes the* `replace` *function, so a* `replaced` *event is generated then as well.*
The widget accepts an event called `setwidth`*. It is the same as calling the setwidth tool function.*

optionmenu - Constructor

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Construct an optionmenu

Synopsis

optionmenu(parent, labels, names, values, hlp, hlp2, nbreak, padx, pady, width, updatelabel, height, justify, font, relief, borderwidth, foreground, background, disabled, anchor, fill, widgetset)

Arguments

parent	*Parent of the menu* *Allowed: Widget*
labels	*The menu item labels* *Allowed: Vector of strings* *Default: Empty menu*
names	*The menu item names* *Allowed: Vector of strings* *Default: Empty menu*
values	*The menu item values* *Allowed: Vector of anything* *Default: Empty menu*
hlp	*Short popuphelp text* *Allowed: String* *Default: No help*
hlp2	*Long popuphelp text* *Allowed: String* *Default: No help*
nbreak	*The break point for menu or listbox usage* *Allowed: Integer* *Default: 20*
padx	*Horizontal padding around text* *Allowed: Integer* *Default: 7*
pady	*Vertical padding around text* *Allowed: Integer* *Default: 3*
width	*Width in character units* *Allowed: Integer* *Default: Auto-scale*
updatelabel	*Update menu label with selected item* *Allowed: T or F* *Default: T*
height	*Height in character units* *Allowed: Integer* *Default: 1*
justify	*Justification of text* *Allowed: String from 'center', 'left', 'right'* *Default: 'center'*
font	*Font of text* *Allowed: X font*
relief	*Edge relief* *Allowed: String from 'flat', 'ridge', 'raised', 'sunken', 'groove'* *Default: 'groove'*
borderwidth	*Border width* *Allowed: Integer* *Default: 2*
foreground	*Color of text* *Allowed: X color* *Default: 'black'*
background	*Background color* *Allowed: X color* *Default: 'lightgrey'*
disabled	*Is inactivated ?* *Allowed: Boolean*

Returns
Agent

Description
This is the constructor. Note that if you give just the `labels` *argument, then* `names` *and* `values` *are set equal to* `labels`*. If you give either of* `names` *or* `values`*, you must also give* `labels` *(it is the primary).*

Example

```
- include 'optionmenu.g'
- labels := "Absolute RelRef RelCen";
- names := ['Absolute', 'Relative to ref. pixel', 'Relative to center'];
- values := [1,2,3];
- f := frame();
- hlp := 'Select type of coordinate';
- m := optionmenu(f, labels, names, values, hlp);
```

Creates an optionmenu with a popup menu.

Example

```
- include 'optionmenu.g'
- labels := "Absolute RelRef RelCen";
- names := ['Absolute', 'Relative to ref. pixel', 'Relative to center'];
- values := [1,2,3];
- f := frame();
- hlp := 'Select type of coordinate';
- m := optionmenu(f, labels, names, values, hlp, nbreak=1)
```

Creates an optionmenu with a popup listbox.

---

optionmenu.extend - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Extend the menu with more items

Synopsis
extend(labels, names, values, width)

Arguments

labels	*The menu item labels* *Allowed: Vector of strings* *Default: Empty menu*
names	*The menu item names* *Allowed: Vector of strings* *Default: Empty menu*
values	*The menu item values* *Allowed: Vector of anything* *Default: Empty menu*
width	*Width in character units* *Allowed: Integer* *Default: Auto-scale*

Returns
Bool

Description

This tool function extends the menu with the supplied items.

Example

```
- labels := "Absolute RelRef RelCen";
- f := frame();
- m := optionmenu(f, labels);
- m.extend(labels="RelDir");
- m.getlabels();
Absolute RelRef RelCen RelDir
-
```

---

optionmenu.replace - Function

Package display
Module widgets
Tool optionmenu

Replace the menu with the provided items

Synopsis
replace(labels, names, values, width)

Arguments

labels	*The menu item labels* *Allowed: Vector of strings* *Default: Empty menu*
names	*The menu item names* *Allowed: Vector of strings* *Default: Empty menu*
values	*The menu item values* *Allowed: Vector of anything* *Default: Empty menu*
width	*Width in character units* *Allowed: Integer* *Default: Auto-scale*

Returns
Bool

Description

This tool function replaces the menu with the supplied items.

Example

```
- labels := "Absolute RelRef RelCen";
- f := frame();
- m := optionmenu(f, labels);
m.getlabels();
Absolute RelRef RelCen
- m.replace("a b c", "A B C", [10,20,40]);
- m.getlabels();
a b c
- m.getnames();
A B C
- m.getvalues();
[10 20 40]
-
```

---

optionmenu.selectindex - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Select an item in the menu by index

Synopsis
selectindex(idx)

Arguments

idx	*The menu item index* *Allowed: Integer*

Returns
Bool

Description

This tool function selects a menu item by its index. It is equivalent to selecting the item with the GUI interface. If the index is invalid, it returns `F`.

Example

```
- labels := "Absolute RelRef RelCen";
- f := frame();
- m := optionmenu(f, labels);
- m.selectindex(3);
- m.getname();
RelCen
-
```

---

optionmenu.selectlabel - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Select an item in the menu by label

Synopsis
selectlabel(label)

Arguments

label	*The menu item label* *Allowed: Same type as the array "labels" used in construction*

Returns
Bool

Description

This tool function selects a menu item by its label. It is equivalent to selecting the item with the GUI interface. If the label is invalid, it returns `F`.

Example

```
- labels := "Absolute RelRef RelCen";
- values := [10, 20, 30];
- f := frame();
- m := optionmenu(f, labels, values=values);
- m.selectlabel('RelRef');
- m.getlabel();
RelRef
-
```

---

optionmenu.selectname - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Select an item in the menu by name

Synopsis
selectname(name)

Arguments

name	*The menu item name* *Allowed: Same type as the array "names" used in construction*

Returns
Bool

Description

This tool function selects a menu item by its name. It is equivalent to selecting the item with the GUI interface. If the name is invalid, it returns `F`.

Example

```
- labels := "Absolute RelRef RelCen";
- names := "A B C";
- f := frame();
- m := optionmenu(f, labels, names);
- m.selectname("C");
- m.getindex();
3
-
```

---

optionmenu.selectvalue - Function

Package display
Module widgets
Tool optionmenu

Select an item in the menu by value

Synopsis
selectvalue(value)

Arguments

value	*The menu item value* *Allowed: Same type as the array "values" used in construction*

Returns
Bool

Description

This tool function selects a menu item by its value. It is equivalent to selecting the item with the GUI interface. If the value is invalid, it returns `F`.

Example

```
- labels := "Absolute RelRef RelCen";
- values := [10, 20, 30];
- f := frame();
- m := optionmenu(f, labels, values=values);
- m.selectvalue(20);
- m.getname();
RelRef
-
```

---

optionmenu.getindex - Function

Package display
Module widgets
Tool optionmenu

Get the index of the current selection

Synopsis
getindex()

Returns
Integer or F

Description

This tool function recovers the index of the currently selected item in the menu. If the menu is empty, it returns `F`*. If no menu item has been selected by the user, it returns unity.*

Example

```
- labels := "Absolute RelRef RelCen";
- f := frame();
- m := optionmenu(f, labels=labels);
- m.selectindex(2);
- m.getindex();
2
```

---

optionmenu.getpreviousindex - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Get the index previous to the current selection

Synopsis
getpreviousindex()

Returns
Integer or F

Description

This tool function recovers the index that you selected previous to the current selection. If the menu is empty, it returns `F`*. If no menu item has been selected by the user, it returns unity.*

Example

```
- labels := "Absolute RelRef RelCen";
- f := frame();
- m := optionmenu(f, labels=labels);
- m.selectindex(2);
- m.selectindex(3);
- m.getpreviousindex();
2
```

---

optionmenu.getlabel - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Get the label of the current selection

Synopsis
getlabel()

Returns
String or F

Description

This tool function recovers the label of the currently selected item in the menu. If the menu is empty, it returns `F`. *If no menu item has been selected by the user, it returns the first item.*

Example

```
- labels := "Absolute RelRef RelCen";
- f := frame();
- m := optionmenu(f, labels=labels);
- m.selectindex(2);
- m.getlabel();
RelRef
```

---

optionmenu.getlabels - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Gets all of the labels associated with the menu

Synopsis
getlabels()

Returns
Vector of String

Description

This tool function recovers the labels associated with each item in the menu. If the menu is empty, it returns an empty array.

Example

```
- labels := "Absolute RelRef RelCen";
- f := frame();
- m := optionmenu(f, labels);
- m.getlabels();
Absolute RelRef RelCen
```

---

optionmenu.getname - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Get the name of the current selection

Synopsis
getname()

Returns
String or F

Description

This tool function recovers the name of the currently selected item in the menu. If the menu is empty, it returns `F`. *If no menu item has been selected by the user, it returns the first item.*

Example

```
- labels := "Absolute RelRef RelCen";
- names := ['Absolute', 'Relative to ref. pix', 'Relative to center'];
- f := frame();
- m := optionmenu(f, labels=labels, names=names);
- m.selectindex(2);
- m.getname();
Relative to ref. pix
```

---

optionmenu.getnames - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Gets all of the names associated with the menu

Synopsis
getnames()

Returns
Vector of String

Description

This tool function recovers the names associated with each item in the menu. If the menu is empty, it returns an empty array.

Example

```
- labels := "Absolute RelRef RelCen";
- names := ['Absolute', 'Relative to ref. pix', 'Relative to center'];
- f := frame();
- m := optionmenu(f, labels=labels, names=names);
- m.selectindex(2);
- m.getnames();
Absolute Relative to ref. pix Relative to center'
```

---

optionmenu.getvalue - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Get the value of the current selection

Synopsis
getvalue()

Returns
Same type as "values" array in constructor or Fail

Description

This tool function recovers the value of the currently selected item in the menu. If the menu is empty, it returns a `Fail`*. If no menu item has been selected by the user, it returns the first item.*

Example

```
- labels := "Absolute RelRef RelCen";
- values := [1,2,3];
- f := frame();
- m := optionmenu(f, labels=labels, values=values);
- m.selectindex(2);
- m.getvalue();
2
```

---

optionmenu.getvalues - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Gets all of the values associated with the menu

Synopsis
getvalues()

Returns
The "values" array given (explicitly or implicitly) in the constructor

Description

This tool function recovers the values associated with each item in the menu. If the menu is empty, it returns an empty array.

Example

```
- labels := "Absolute RelRef RelCen";
- values := [1,2,3]
- f := frame();
- m := optionmenu(f, labels, values=values);
- m.getvalues();
[1 2 3]
```

---

optionmenu.findlabel - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Find the index of the item with the given label

Synopsis
findlabel(label)

Arguments

label	*The menu item label* *Allowed: Same type as the array "labels" used in construction*

Returns
Integer or F

Description

This tool function find the index in the menu of the item specified by its label. If you give an invalid label, `F` *is returned.*

Example

```
- labels := "Absolute RelRef RelCen";
- values := [10, 20, 30];
- f := frame();
- m := optionmenu(f, labels, values=values);
- m.findlabel('Absolute');
1
```

---

optionmenu.findname - Function

Package display
Module widgets
Tool optionmenu

Find the index of the item with the given name

Synopsis
findname(name)

Arguments

name	*The menu item name* *Allowed: Same type as the array "names" used in construction*

Returns
Integer or F

Description

This tool function find the index in the menu of the item specified by its name. If you give an invalid name, `F` *is returned.*

Example

```
- labels := "Abs RelRef RelCen";
- names := ['Absolute', 'Relative to ref. pixel', 'Relative to center'];
- values := [10, 20, 30];
- f := frame();
- m := optionmenu(f, labels, names, values);
- m.findname('Relative to ref. pixel');
2
```

---

optionmenu.findvalue - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Find the index of the item with the given value

Synopsis
findvalue(value)

Arguments

value	*The menu item value* *Allowed: Same type as the array "value" used in construction*

Returns
Integer or F

Description

This tool function find the index in the menu of the item specified by its value. If you give an invalid value, `F` *is returned.*

Example

```
- labels := "Absolute RelRef RelCen";
- values := [10, 20, 30];
- f := frame();
- m := optionmenu(f, labels, values=values);
- m.findvalue(20);
2
-
```

---

optionmenu.geteventmask - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Get the event mask

Synopsis
geteventmask()

Returns
Vector of Bool

Description

This tool function recovers the event mask. This mask indicates whether the `select` *event will be emitted or not when you select a menu item. By default the mask is all T. You can modify it with the seteventmask tool function.*

Example

```
- labels := "Absolute RelRef RelCen";
- values := [1,2,3];
- f := frame();
- m := optionmenu(f, labels=labels, values=values);
- m.geteventmask();
[T T T]
- m.seteventmask([T,F,T]);
- m.geteventmask();
[T F T]
```

---

optionmenu.seteventmask - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Set the event mask

Synopsis

seteventmask(mask)

Arguments

mask	*The event mask* *Allowed: Vector of Bool*

Returns

Bool

Description

This tool function sets the event mask. This mask indicates whether the `select` *event will be emitted or not when you select a menu item. By default the mask is all T. You can get the current mask with the geteventmask tool function. If the mask is invalid, it returns* `F`.

Example

```
- labels := "Absolute RelRef RelCen";
- values := [1,2,3];
- f := frame();
- m := optionmenu(f, labels=labels, values=values);
- m.seteventmask([T,F,T]);
- m.geteventmask();
[T F T]
- whenever m->select do {print $value;};
```

In this example, when you selected the second item, no event would be emitted.

---

optionmenu.setlabel - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Set the label of the optionmenu

Synopsis
setlabel(label)

Arguments

label	*The label* *Allowed: String*

Returns
Bool

Description

This tool function sets the label of the optionmenu. This label can be anything, it does not have to be in the optionmenu list of labels.

Example

```
- labels :=
- values := [1,2,3];
```

```
- f := frame();
- m := optionmenu(f, labels=labels, values=values);
- m.setlabel('fish');
- m.getlabel()
fish
- m.getlabels()
Absolute RelRef RelCen
```

Although the label of the menu is now "fish", the labels of the items in the menu are unchanged.

---

optionmenu.setforeground - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Set the color of the foreground of the optionmenu button

Synopsis

setforeground(color)

Arguments

color	*The color* *Allowed: Any X color encoded as a String*

Returns

Bool

Description

This tool function sets the color of the foreground of the optionmenu button.

---

optionmenu.setbackground - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Set the color of the background of the optionmenu button

Synopsis
setbackground(color)

Arguments

color	*The color* *Allowed: Any X color encoded as a String*

Returns
Bool

Description

This tool function sets the color of the background of the optionmenu button.

---

optionmenu.setwidth - Function

Package display
Module widgets
Tool optionmenu

Set the width of the optionmenu button

Synopsis
setwidth(width)

Arguments

width	*The width in character units* *Allowed: Integer*

Returns
Bool

Description

This tool function sets the width of the optionmenu button in character units.

---

optionmenu.disabled - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Disable or enable the optionmenu widget

Synopsis
disabled(disable)

Arguments

disable	*disable the widget* *Allowed: T or F* *Default: T*

Returns
Bool

Description

This tool function disables or enables the optionmenu.

---

optionmenu.isenabled - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Is the optionmenu widget enabled ?

Synopsis
isenabled()

Returns

Bool

Description

This tool function indicated whether the optionmenu has been disabled or not.

Example

```
- f := frame();
- m := optionmenu(f, "dog cat");
- m.disable();
- m.isenabled()
F
-
```

---

optionmenu.done - Function

Package	**display**
Module	**widgets**
Tool	**optionmenu**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This tool function destroys the widget.

Example

```
- f := frame();
```

```
- m := optionmenu(f, "dog cat");
- m.done();
T
- m
F
-
```

1.4.25 pixelrange - Tool

Package display
Module widgets

Widget to handle a pixel range specification
include "pixelrange.g"

Constructors

pixelrange	*Construct a pixelrange widget*

Functions

disableallbutton	*Disable the 'all' radio button*
disabled	*Disable or enable the widget*
done	*Destroy ourselves*
enableallbutton	*Enable the 'all' radio button*
getradiovalue	*Get the value of the radio buttons*
getslidervalues	*Get the values of the sliders*
setradiovalue	*Set the state of a specific radio button*
setrange	*Set new values for the sliders*

Description

This Tk widget creates three radio buttons which the user uses to indicate whether they wish to select pixels via an inclusion range (button 'include'), an exclusion range (button 'exclude') or select all (button 'all') pixels. For the inclusion/exclusion ranges, two sliders with entry boxes to type into are provided for the user to set the range. The sliders are set to minimum and maximum and deactivated if the user selects the 'all' radio button.
Currently, this widget is rather specialized. It will probably be made more generic at some point.

pixelrange - Constructor

Package	**display**
Module	**widgets**
Tool	**pixelrange**

Construct a pixelrange widget

Synopsis
pixelrange(parent, min, max, labels, widgetset)

Arguments

parent	*Parent of the menu* *Allowed: Tk frame* *Default: F*
min	*The minimum value for the sliders* *Allowed: Float* *Default: 0*
max	*The maximum value for the sliders* *Allowed: Float* *Default: 1*
labels	*The slider labels* *Allowed: String* *Default: "include exclude all"*
widgetset	*The widget server to use* *Allowed: widgetserver tool* *Default: dws*

Returns
Agent

Description

Construct the widget. The resolution of the sliders is 1% of the range.
The argument `labels` *gives the labels for each slider. It may be that the context of your use of this widget requires names different from the default.*

Example

```
- include 'pixelrange.g'
- include 'widgetserver.g'
- f := dws.frame();
- r := pixelrange(parent=f, min=-100, max=20, widgetset=dws);
```

---

pixelrange.disabled - Function

Package	**display**
Module	**widgets**
Tool	**pixelrange**

Disable or enable the widget

Synopsis
disabled(which, sliders)

Arguments

which	*disable the radio buttons* *Allowed: T, F, or unset* *Default: unset*
sliders	*disable the sliders* *Allowed: T, F, or unset* *Default: unset*

Returns
T

Description

This function disables or enables the widget (does not need matching pair). You can disable/enable the radio buttons part of the widget and the sliders part of the widget independently. If either `which` *or* `sliders` *is unset, then their state is not affected. You must set these arguments to T or F to disable or enable, respectively.*

---

pixelrange.disableallbutton - Function

Package	**display**
Module	**widgets**
Tool	**pixelrange**

Disable the 'all' radio button

Synopsis
disableallbutton()

Returns
T

Description

This function specifically disables only the 'all' radio button

---

pixelrange.done - Function

Package	**display**
Module	**widgets**
Tool	**pixelrange**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This tool function destroys the widget.

---

pixelrange.enableallbutton - Function

Package	**display**
Module	**widgets**
Tool	**pixelrange**

Enable the 'all' radio button

Synopsis
enableallbutton()

Returns
T

Description

This function specifically enables only the 'all' radio button.

---

pixelrange.getslidervalues - Function

Package	**display**
Module	**widgets**
Tool	**pixelrange**

Get the values of the sliders

Synopsis
getslidervalues()

Returns
Vector of length 2

Description

This function gets the current values of the sliders.

---

pixelrange.getradiovalue - Function

Package	**display**
Module	**widgets**
Tool	**pixelrange**

Get the value of the radio buttons

Synopsis
getradiovalue()

Returns
Record

Description

This function gets the current value of the radio buttons. If one of the buttons is selected, it returns a record with two fields: 'name' and 'index' containing the name of the button and its index (1, 2 or 3 for 'include', 'exclude' and 'all'). If no buttons are selected, it returns an empty record.

---

pixelrange.setrange - Function

Package	**display**
Module	**widgets**
Tool	**pixelrange**

Set new values for the sliders

Synopsis
setrange(min, max)

Arguments

min	*The minimum value for the sliders* *Allowed: Float*
max	*The maximum value for the sliders* *Allowed: Float*

Returns
T

Description

This function sets a new range for each of the sliders.

---

pixelrange.setradiovalue - Function

Package	**display**
Module	**widgets**
Tool	**pixelrange**

Set the state of a specific radio button

Synopsis
setradiovalue(idx, state)

Arguments

idx	*Button index or name* *Allowed: Integer or string*
state	*State of button* *Allowed: T or F*

Returns
T

Description

This function sets the state of a specific button. You can specify the 'include' button as `index=1` *or* `index='include`. *You can specify the 'exclude' button as* `index=2` *or* `index='exclude`. *You can specify the 'all' button as* `index=3` *or* `index='all'`. *Any other values are ignored. When you set a button's state to T, any other button that is currently selected will deselect.*

1.4.26 popupselectmenu - Tool

Package	**display**
Module	**widgets**

Tk widget to add a popup select menu to another widget

include "popupselectmenu.g"

Constructors

popupselectmenu	*Construct a popup select menu*

Functions

disable	*Disable the popupselectmenu*
enable	*Enable the popupselectmenu*
kill	*Destroy ourselves*

Description

This Tk widget will add a popup select menu to a frame. When button-3 (usually the right-most one) is clicked, a menu of choices will appear. If the user clicks on one of these, then a select event is returned and the value of the event is the choice string.

popupselectmenu - Constructor

Package	**display**
Module	**widgets**
Tool	**popupselectmenu**

Construct a popup select menu

Synopsis

popupselectmenu(fr, labels, font, relief, foreground, background, widgetset)

Arguments

fr	*Frame to attach to* *Allowed: Tk frame*
labels	*Choices* *Allowed: String* *Default: ''*
font	*Font of text* *Allowed: X font* *Default: ''*
relief	*Edge relief* *Allowed: String from 'flat', 'ridge', 'raised', 'sunken', 'groove'* *Default: 'flat'*
foreground	*Color of text* *Allowed: X color* *Default: 'black'*
background	*Background color* *Allowed: X color* *Default: 'lightgrey'*
widgetset	*Widget set to use* *Allowed: A widgetserver closure tool* *Default: dws*

Returns
popupselectmenu subsequence

Description
Constructs a popupselectmen according to the specified construction arguments, and returns a ref to the widget. This is a subsequence and you should remember to catch the relevant events.

Example

```
include 'popupselectmenu.g'
f := frame();
r := popupselectmenu(f, labels=['fish', 'fowl']);
whenever r->* do {print $name, $value}
```

---

popupselectmenu.disable - Function

Package display
Module widgets
Tool popupselectmenu

Disable the popupselectmenu

Synopsis
disable()

Returns
Bool

Description
A call to this function disables the popupselectmenu.

Example

```
r := popupselectmenu(f, label=['fish', 'fowl']);
# ...
r.disable();
```

---

popupselectmenu.enable - Function

Package	**display**
Module	**widgets**
Tool	**popupselectmenu**

Enable the popupselectmenu

Synopsis
enable()

Returns
Bool

Description
A call to this function enables the popupselectmenu.

Example

```
r := popupselectmenu(f, label=['fish', 'fowl']);
# ...
r.enable();
```

---

popupselectmenu.kill - Function

Package	**display**
Module	**widgets**
Tool	**popupselectmenu**

Destroy ourselves

Synopsis
kill()

Returns
Bool

Description

This tool function destroys the widget.

Example

```
r := popupselectmenu(f, label=['fish', 'fowl']);
# ...
r.kill();
- r
F
-
```

1.4.27 profilewidthentry - Tool

Package	**display**
Module	**widgets**

Widget to handle a 1-D profile width specification
include "profilewidthentry.g"

Constructors

profilewidthentry	*Construct a profilewidthentry widget*

Functions

disable	*Disable the widget*
done	*Destroy ourselves*
enable	*Enable the widget*
getvalue	*Get the value of the widget*

Description

This widget supports specifying the averaging width when generating a 1-D profile. For example, you may be displaying an image of RA/DEC/Frequency. You wish to find the profile along the Frequency axis and spatially average in RA and DEC. This widget creates an optionmenu specifying the averaging shape and an entry box into which the user types the averaging half width.
Emits event `select` *when the averaging shape is selected. The value of the event is the shape type. Emits event* `value` *when the user types in a width and pressed return. The value of the event is the width.*

profilewidthentry - Constructor

Package	**display**
Module	**widgets**
Tool	**profilewidthentry**

Construct a profilewidthentry widget

Synopsis
profilewidthentry(parent, relief, width, widgetset)

Arguments

parent	*Parent of the menu* *Allowed: Tk frame* *Default: F*
relief	*Relief for holder frame* *Allowed: Standard Tk relief string* *Default: 'raised'*
width	*Width for entry box* *Allowed: Integer* *Default: 10*
widgetset	*The widget server to use* *Allowed: widgetserver tool* *Default: dws*

Returns
Agent

Description

Construct the profilewidthentry widget.

Example

```
include 'profilewidthentry.g'
f := dws.frame();
r := profilewidthentry(parent=f)
```

---

profilewidthentry.disable - Function

Package display
Module widgets
Tool profilewidthentry

Disable the widget

Synopsis
disable()

Returns
T

Description

This function disables the widget (does not need matching pair).

---

profilewidthentry.enable - Function

Package	**display**
Module	**widgets**
Tool	**profilewidthentry**

Enable the widget

Synopsis
enable()

Returns
T

Description

This function enables the widget (does not need matcing pair).

---

profilewidthentry.done - Function

Package	**display**
Module	**widgets**
Tool	**profilewidthentry**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This function destroys the widget.

---

profilewidthentry.getvalue - Function

Package	**display**
Module	**widgets**
Tool	**profilewidthentry**

Get the value of the widget

Synopsis
getvalue()

Returns
Record

Description

This function gets the current averaging type and the value into a record.

Example

```
include 'profilewidthentry.g'
f := dws.frame();
r := profilewidthentry(parent=f)
v := r.getvalue()
print 'value=', v
#<< [value=0, type=box]
```

1.4.28 radiobuttons - Tool

Package	**display**
Module	**widgets**

Widget handling multiple selectable radio buttons

include "radiobuttons.g"

Constructors

radiobuttons	*Construct a radiobuttons widget*

Functions

disablebutton	*Disable specific button*
disabled	*Disable or enable the widget*
done	*Destroy ourselves*
enablebutton	*Enable a specific button*
getvalue	*Get the value of the widget*
reset	*Reset the widget*
setstate	*Set the state of a specific button*

Description

This Tk widget creates many radio buttons (horizontal or vertical) and then allows you to select them remotely, disable one or all of them, reset them all, and recover the selection. It is rather specialized at present but may grow into a more generic widget in the future.

radiobuttons - Constructor

Package	**display**
Module	**widgets**
Tool	**radiobuttons**

Construct a radiobuttons widget

Synopsis

radiobuttons(parent, names, default, side, widgetset)

Arguments

parent	*Parent of the menu* *Allowed: Widget* *Default: Standalone*
names	*The names of the buttons* *Allowed: Vector of strings* *Default: None*
default	*Index of names for button that is initially selected* *Allowed: Integer* *Default: None*
side	*Side to pack buttons in* *Allowed: String from 'left', 'right', 'top', 'bottom'* *Default: 'left'*
widgetset	*The widget server to use* *Allowed: widgetserver tool* *Default: dws*

Returns
Agent

Description

Example

```
- include 'radiobuttons.g'
- include 'widgetserver.g'
- names := "a b c";
- f := dws.frame();
- r := radiobuttons(parent=f, names=names, default=2, side='top', widgetset=dws);
```

---

radiobuttons.disabled - Function

Package display
Module widgets
Tool radiobuttons

Disable or enable the widget

Synopsis

disabled(disable, allbuttons)

Arguments

disable	*disable widget* *Allowed: T or F* *Default: T*
allbuttons	*apply to all buttons* *Allowed: T or F* *Default: F*

Returns
T

Description

This function disables or enables the widget (does not need matching pair). It does this by disabling/enabling a holder frame into which the buttons are put. By default, this means that the buttons themselves are not enabled or disabled. However, if you set `allbuttons=T`*, then all the buttons are enabled or disabled as well.*

---

radiobuttons.disablebutton - Function

Package display
Module widgets
Tool radiobuttons

Disable specific button

Synopsis
disablebutton(idx)

Arguments

idx	*Button number* *Allowed: Integer*

Returns
T

Description

This function disables a specific button in the widget.

---

radiobuttons.done - Function

Package	**display**
Module	**widgets**
Tool	**radiobuttons**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This tool function destroys the widget.

---

radiobuttons.enablebutton - Function

Package	**display**
Module	**widgets**
Tool	**radiobuttons**

Enable a specific button

Synopsis
enablebutton(idx)

Arguments

idx	*Button number* *Allowed: Integer*

Returns
T

Description

This function enables a specific button in the widget.

---

radiobuttons.getvalue - Function

Package	**display**
Module	**widgets**
Tool	**radiobuttons**

Get the value of the widget

Synopsis
getvalue()

Returns
Record

Description

This function gets the state of the widget. If one of the buttons is selected, it returns a record with two fields: 'name' and 'index' containing the name of the button and its index. If no buttons are selected, it returns an empty record.

---

radiobuttons.reset - Function

Package	**display**
Module	**widgets**
Tool	**radiobuttons**

Reset the widget

Synopsis
reset(idx)

Arguments

idx	*Set this button to T* *Allowed: Integer* *Default: -1*

Returns
T

Description

This function deselects all of the buttons. You can optionally nominate one button to have its state set to T with the `idx` *index (default of* -1 *means select none).*

---

radiobuttons.setstate - Function

Package display
Module widgets
Tool radiobuttons

Set the state of a specific button

Synopsis
setstate(idx, state)

Arguments

idx	*Button number* *Allowed: Integer*
state	*State of button* *Allowed: T or F*

Returns
T

Description

This function sets the state of a specific button. When you set a button's state to T, any other button that is currently selected will deselect.

1.4.29 recordbrowserwidget - Tool

Package **display**
Module **widgets**

AIPS++ widget to view Glish records
include "recbrowserwidget.g"

Description

This AIPS++ widget allows the user to view a Glish record in a canvas. The recordbrowserwidget is AIPS++ aware and will display tables and arrays (beyond certain limits) using the tablebrowser and arraybrowser respectively.

recordbrowserwidget - Constructor

Package **display**
Module **widgets**
Tool **recordbrowserwidget**

Construct a recordbrowserwidget

Synopsis

recordbrowserwidget(parent, therecord, readonly, show, width, font, widgetset, displaytable)

Arguments

parent	*Parent of the widget* *Allowed: Tk frame* *Default: F*
therecord	*A valid Glish record (required)* *Allowed: a Glish record*
readonly	*Flag to disallow editing* *Allowed: boolean* *Default: T*
show	*Flag to display widget after contruction* *Allowed: boolean* *Default: T*
width	*Initial width of the canvas* *Allowed: Integer* *Default: 400*
font	*Font of text* *Allowed: X font* *Default: ''*
widgetset	*Widget set to use* *Allowed: A widgetserver closure tool* *Default: dws*
displaytable	*Flag indicating table awareness* *Allowed: boolean* *Default: T*

Returns
recordbrowserwidget widget

Description
Constructs a recordbrowserwidget according to the specified construction arguments, and returns the subsequence of the widget. Remember to use the `recordbrowserwidget` *via the widgetserver if at all possible.*
The recordbrowser widget must always be initialized with a valid record.
If no parent is specified then a minmal recordbrowser is displayed.
You may send the recordbrowserwidget many events

1. **newrecord, record** *causes the widget to display the specified record i.e. rbw− >newrecord(myrec).*

2. **font** *cause the recordbrowser to use the new font sent to it, rbw− >font(myfont).*

3. **show** *causes the recordbrowser to become visible if not already.*

4. **close** *causes the recordbrowser to become invisible if not already.*

5. **readonly** *toggles the readonly flag enabling editing if false, rbw− >readonly() or rbw− >readonly(F).*
6. **reset** *resets the record to it's original state (only valid if readonly=F), rbw− >reset().*
7. **vectorsize** *sets the limits of arrays that are displayed, rbw− >vectorsize([60,60] (The examples will let you look at arrays up to 60 by 60 in the recordbrowser, larger arrays will be viewed in in the array browser).*
8. **showproperties** *causes a properties dialog for the the recordbrowser widget to be displayed, rbw− >showproperties(). The dialog window allows the interactive setting of vectorsize and font.*
9. **newfield** *if readonly is F, then you may added field to the record, rb− >newfield(fieldName='field1.field2', fieldValue='evalable string'). Note the fieldValue must be an evalable string otherwise bad things happen.*
10. **deletefield** *if readonly is F, then you may delete fields from the record, rb− >deletefield('field1.field2'). Just specify the fields in the record you want to delete.*

The recordbrowser widget does not signal any events.

Example

```
include 'recordbrowserwidget.g'
rb := dws.recordbrowserwidget(therecord=system);
```

Will produce a generic record-browser dialog.

1.4.30 restoreregions - Tool

Package display
Module widgets

Tk widget to restore image regions of interest from a table
include "restoreregions.g"

Constructors

restoreregions Construct a restoreregions widget

Functions

done	*Destroy ourselves*
gui	*Restart the GUI*
refresh	*Refresh the display*
regions	*Recover the selected regions from the Table into a record*
settable	*Set a (new) table*

Description

This widget enables you to restore the image `regions-of-interest` *(or more simply, regions - see Regionmanager) stored in an AIPS++ Table. You can restore the regions into a record, or into global name space (i.e. into* Glish*).*
The widget displays, in listboxes, the regions stored in the Table. If you select specific regions, then the GUI expands to enable you to rename them during the restore. The 'Restore' button restores only selected regions. The 'Restore all' button restores all regions with no renaming option. If you select 'Confirm', you will be prompted for confirmation if you attempt to overwrite a pre-existing Glish *symbol. The 'Dismiss' button dismisses the GUI. You can recover it with the gui function.*
Under the 'File' menu, you can either dismiss the GUI or destroy the widget (same as invoking function done).
You can change the Table you are accessing at any time by entering the new Table name in the appropriate entry box and entering a carriage return.
The Table can be specified in many ways (at construction and via the entry box in the GUI); a Table `tool`*, an Image)* `tool`*, the name of a Table (a string) which might be an AIPS++ image, or the name of a* Glish *symbol which is either a Table or an Image* `tool`*.*
This widget emits an event called `restored` *(it has no value) when the regions have been restored. It also emits an event called* `dismissed` *(it has no value) when the dismiss button on the GUI is pressed.*

restoreregions - Constructor

Package	**display**
Module	**widgets**
Tool	**restoreregions**

Construct a restoreregions widget

Synopsis

restoreregions(parent, table, changenames, globalrestore, widgetset)

Arguments

parent	*Parent of the widget* *Allowed: Widget* *Default: F*
table	*The table of interest* *Allowed: Table tool, image tool or string* *Default: ''*
changenames	*Entry boxes to enable region name changes are provided* *Allowed: T or F* *Default: T*
globalrestore	*Restore to global name space or a record* *Allowed: T or F* *Default: F*
widgetset	*The widget server to use* *Allowed: widgetserver tool* *Default: dws*

Returns
Agent

Description

This is the constructor. If `parent=F`*, then the widget will make its own parent frame.*
The Table can be specified in many ways; a Table `tool`*, an Image)* `tool`*, the name of a Table (a string) which might be an AIPS++ image, or the name of a* Glish *symbol which is either a Table or an Image* `tool`*.*
If `changenames=T`*, when you select regions, entry boxes which enable you to change the names of the regions on restore are provided.*
If `globalrestore=T`*, the regions are restored to global name space (i.e. to* Glish*). Otherwise they are restored to a record which you recover with function regions.*

Example

```
- include 'restoreregions.g'
- r := restoreregions(table='hcn')
- whenever rr->restored do {
   regions := rr.regions();
  }
```

In this example, the widget is contructed with a Table called hcn*. When the user has selected the regions, they are recovered into a record (because*

`globalrestore=F`*). Each field of the record contains one region* `tool` *(see Regionmanager). The fields of that record are named according to the names of the regions in the table.*

---

restoreregions.done - Function

Package	**display**
Module	**widgets**
Tool	**restoreregions**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This tool function destroys the widget.

Example

```
- r := restoreregions();
- r
ref [*agent*=<agent>, done=<function>, gui=<function>,
regions=<function>, setnewtable=<function>]
- r.done();
T
- r
F
-
```

---

restoreregions.gui - Function

Package	**display**
Module	**widgets**
Tool	**restoreregions**

Restart the GUI

Synopsis
gui()

Returns
Bool

Description

When the widget is first constructed, it brings up a GUI. When you restore regions, he GUI is dismissed. This function restarts the GUI; it will have the same state as when it was dismissed.

---

restoreregions.refresh - Function

Package	**display**
Module	**widgets**
Tool	**restoreregions**

Refresh the display

Synopsis
refresh()

Returns
Bool

Description

This function reaccesses the table to find the regions and redisplays them in the GUI.

---

restoreregions.regions - Function

Package	**display**
Module	**widgets**
Tool	**restoreregions**

Recover the selected regions from the Table into a record

Synopsis
regions()

Returns
Record of regions

Description

If you constructed with `globalrestore=T`*, this function returns an empty record. Otherwise, it recovers all the selected regions into a record. Each field of the record contains one region* `tool` *(see Regionmanager). The fields are named according to the names of the regions in the Table.*

Example

```
- include 'restoreregions.g'
- r := restoreregions();
- whenever rr->restored do {
   regions := rr.regions();
  }
```

---

restoreregions.settable - Function

Package	**display**
Module	**widgets**
Tool	**restoreregions**

Set a (new) table

Synopsis
settable(table)

Arguments

table	*The table of interest* *Allowed: Table tool, image tool or string* *Default: ''*

Returns
Bool or fail

Description

This function manually (i.e. not via the GUI interface) sets a (new) Table name. It is equivalent to doing it via the GUI.
The Table can be specified in many ways; a Table `tool`*, an Image)* `tool`*, the name of a Table (a string) which might be an AIPS++ image, or the name of a* Glish *symbol which is either a Table or an Image* `tool`*.*

Example

```
- include 'restoreregions.g'
- include 'image.g'
- include 'table.g'
- r := restoreregions(table='hcn');
- im := image('hcn2');
- t := table('hcn2');
- r.settable(im);
- r.settable('hcn2');
- r.settable(t);
- r.settable('im');
- r.settable('t');
```

In this example, we show all the ways you can specify the table.

1.4.31 rollup - Tool

Package display
Module widgets

Tk widget to hide and expose an embedded frame on demand
include "rollup.g"

Constructors

rollup Construct a rollup

Functions

disable	*Disable the rollup*
done	*Destroy ourselves*
down	*Show the rollup contents*
enable	*Enable the rollup*
frame	*Return the embedded frame for use*
hidden	*Is the rollup hidden ?*
setpopuphelp	*Set new popuphelp*
switch	*Flip the visible state of the rollup*
up	*Hide the rollup contents*

Description

This Tk widget will manage the hiding and exposure of an embedded frame. The idea is that you create a rollup, and then place GUI items (eg. frames, buttons, listboxes) inside the frame that the rollup provides. The widget builds itself with a titlebar, which the user can press to switch the view between hidden and exposed. Functions exist to control the visibility of the rollup from the command line. The rollup widget can be treated like many of the built in Tk widgets, except that a frame is not returned: the programmer must explicitly extract the embedded frame with the "frame" function. Many of the construction arguments are identical to those of the built in Tk widgets.

rollup - Constructor

Package	**display**
Module	**widgets**
Tool	**rollup**

Construct a rollup

Synopsis

rollup(parent, font, relief, borderwidth, side, padx, pady, expand, foreground, background, title, titleforeground, titlebackground, show, widgetset)

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
font	*Font of text* *Allowed: X font* *Default: ''*
relief	*Edge relief* *Allowed: String from 'flat', 'ridge', 'raised', 'sunken', 'groove'* *Default: 'flat'*
borderwidth	*Border width* *Allowed: Integer* *Default: 2*
side	*Direction for child placement* *Allowed: String from 'top', 'left', 'right', 'bottom'* *Default: 'top'*
padx	*Horizontal padding around text* *Allowed: Integer* *Default: 0*
pady	*Vertical padding around text* *Allowed: Integer* *Default: 0*
expand	*How to expand when resized* *Allowed: String from 'x', 'y', 'both', 'none'* *Default: 'both'*
foreground	*Color of text* *Allowed: X color* *Default: 'black'*
background	*Background color* *Allowed: X color* *Default: 'lightgrey'*
title	*String displayed at top of widget* *Allowed: String* *Default: ''*
titleforeground	*Color of title text* *Allowed: X color* *Default: 'black'*
titlebackground	*Background title color* *Allowed: X color* *Default: 'lightgrey'*
show	*Whether to expose frame immediately* *Allowed: Boolean* *Default: T*
widgetset	*Widget set to use* *Allowed: A widgetserver closure tool* *Default: dws*

Returns
rollup widget

Description
Constructs a rollup according to the specified construction arguments, and returns a ref to the widget.

Example

```
include 'rollup.g'
f := frame();
r := rollup(f, title='My rollup');
```

---

rollup.frame - Function

Package	**display**
Module	**widgets**
Tool	**rollup**

Return the embedded frame for use

Synopsis
frame()

Returns
Tk frame

Description
This simply returns the frame that will be exposed and hidden by the rollup widget.

Example

```
f := frame();
r := rollup(f, title='My rollup');
embedded := r.frame();
b := button(embedded, 'A button');
```

---

rollup.switch - Function

Package	**display**
Module	**widgets**
Tool	**rollup**

Flip the visible state of the rollup

Synopsis
switch()

Returns
Bool

Description
A call to this function flips the state of the rollup to hidden if it was previously exposed, and vice versa.

Example

```
r := rollup(f, title='My rollup');
r.switch();
```

---

rollup.down - Function

Package	**display**
Module	**widgets**
Tool	**rollup**

Show the rollup contents

Synopsis
down()

Returns
Bool

Description
A call to this function makes the rollup expose the embedded frame.

Example

```
r := rollup(f, title='My rollup', show=F);
# ...
r.down();
```

---

rollup.up - Function

Package	**display**
Module	**widgets**
Tool	**rollup**

Hide the rollup contents

Synopsis
up()

Returns
Bool

Description
A call to this function makes the rollup hide the embedded frame.

Example

```
r := rollup(f, title='My rollup', show=T);
# ...
r.up();
```

---

rollup.disable - Function

Package	**display**
Module	**widgets**
Tool	**rollup**

Disable the rollup

Synopsis
disable()

Returns
Bool

Description
A call to this function disables the rollup.

Example

```
r := rollup(f, title='My rollup', show=T);
# ...
r.disable();
```

---

rollup.enable - Function

Package	**display**
Module	**widgets**
Tool	**rollup**

Enable the rollup

Synopsis
enable()

Returns
Bool

Description
A call to this function enables the rollup.

Example

```
r := rollup(f, title='My rollup', show=T);
# ...
r.enable();
```

---

rollup.hidden - Function

Package	**display**
Module	**widgets**
Tool	**rollup**

Is the rollup hidden ?

Synopsis
hidden()

Returns
Bool

Description

This tool function returns T if the rollup is hidden (rolled up) and F if it is exposed (rolled down).

Example

```
- r := rollup(f, title='My rollup', show=T);
- r.up()
- r.hidden();
T
- r.down()
- r.hidden();
F
```

---

rollup.done - Function

Package	**display**
Module	**widgets**
Tool	**rollup**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This tool function destroys the widget.

Example

```
- r := rollup(f, title='My rollup', show=T);
- r.done();
T
- r
F
-
```

---

rollup.setpopuphelp - Function

Package	**display**
Module	**widgets**
Tool	**rollup**

Set new popuphelp

Synopsis
setpopuphelp(short, long, width)

Arguments

short	*Short help* *Allowed: String*
long	*Long help* *Allowed: String* *Default: Unset*
width	*Width of short help* *Allowed: Integer* *Default: 60*

Returns
T or fail

Description

This tool function allows you to set the popuphelp that is attached to the rollup. By default, when the rollup is contructed, you get a basic popuphelp. This enables you to write something more informative to the user if appropriate. As with standard popuphelp, you can specify a short (required) and long (optional) help (i.e. when you click the right button the long help is shown in addition to the short). Likewise the standard popuphelp short-help `width` *argument is also provided.*

Example

```
- r := rollup(f, title='My rollup', show=T);
- r.setpopuphelp('This is my rollup', 'You should know what it does')
```

1.4.32 saveregions - Tool

Package display
Module widgets

Tk widget to save image regions to a Table
include "saveregions.g"

Constructors

saveregions	*Construct a saveregions widget*

Functions

done	*Destroy ourselves*
gui	*Restart the GUI*
setregions	*Set new regions to save*
settable	*Set a (new) table*

Description

This Tk widget enables you to save image `regions-of-interest` *(or more simply, regions - see regionmanager) stored to an AIPS++ Table.*
The widget displays, in listboxes, the regions passed in. If you select specific regions, then the GUI expands to enable you to rename them during the save. The 'Save' button saves only selected regions. The 'Save all' button saves all regions with no renaming option. If you select 'Confirm', you will be prompted for confirmation if you attempt to overwrite a region in the Table. The 'Dismiss' button dismisses the GUI. You can recover it with the gui function.
Under the 'File' menu, you can either dismiss the GUI or destroy the widget (same as invoking function done).
The Table can be specified in many ways (at construction and via the entry box in the GUI); a Table `tool`*, an Image)* `tool`*, the name of a Table (a string) which might be an AIPS++ image, or the name of a* Glish *symbol which is either a Table or an Image* `tool`*. If the Table does not exist, it will be created. It will have no useful structure beyond that needed to store the regions.*
This widget emits an event called `saved` *(it has no value) when the regions have been restored. It also emits an event called* `dismissed` *(it has no value) when the dismiss button on the GUI is pressed.*

saveregions - Constructor

Package	**display**
Module	**widgets**
Tool	**saveregions**

Construct a saveregions widget

Synopsis

saveregions(parent, table, regions, changenames, globalsave, widgetset)

Arguments

parent	*Parent of the widget* *Allowed: Widget* *Default: F*
table	*The table of interest* *Allowed: Table tool, image tool or string* *Default: ”*
regions	*The regions to save, or the region names* *Allowed: Record of regions or vector of strings*
changenames	*Entry boxes to enable region name changes are provided* *Allowed: T or F* *Default: T*
globalsave	*The regions are saved from global name space or a record* *Allowed: T or F* *Default: F*
widgetset	*The widget server to use* *Allowed: widgetserver tool* *Default: dws*

Returns
saveregions tool or fail

Description

This is the constructor. If `parent=F`*, then the widget will make its own parent frame. The Table can be specified in many ways; a Table* `tool`*, an Image)* `tool`*, the name of a Table (a string) which might be an AIPS++ image, or the name of a* Glish *symbol which is either a Table or Image* `tool`*. If the Table does not exists, a be one will be created. It will have no useful structure beyond that needed to store the regions.*
The regions are either provided in a record (`globalsave=F` *or should exist in global name space, whereupon you just provide their names.*
If `changenames=T`*, when you select regions, entry boxes which enable you to change the names of the regions on save are provided.*

Example

```
- include 'saveregions.g'
- r1 := drm.box([10,10],[20,30]);
- r2 := drm.quarter();
- sr1 := saveregions(table='hcn', regions="r1 r2", globalsave=T, changenames=F);
- r := [=];
- r[1] := r1;
```

```
- r[2] := r2;
- sr2 := saveregions(table='hcn', regions=r, globalsave=F);
```

In this example, two widgets are contructed with a Table called 'hcn'. The first shows how to save from global name space. The second saves from a record.

---

saveregions.done - Function

Package	**display**
Module	**widgets**
Tool	**saveregions**

Destroy ourselves

Synopsis
done()

Returns
Bool or fail

Description

This function destroys the widget.

Example

```
- r := saveregions();
- r
ref [*agent*=<agent>, done=<function>, gui=<function>,
regions=<function>, setnewtable=<function>]
- r.done();
T
- r
F
-
```

---

saveregions.gui - Function

Package	**display**
Module	**widgets**
Tool	**saveregions**

Restart the GUI

Synopsis
gui()

Returns
Bool

Description

When the widget is first constructed, it brings up a GUI. When you save regions, the GUI is dismissed. This function restarts the GUI; it will have the same state as when it was dismissed.

saveregions.setregions - Function

Package	**display**
Module	**widgets**
Tool	**saveregions**

Set new regions to save

Synopsis
setregions(regions)

Arguments

regions	*Regions or region names* *Allowed: Record of regions or vector of strings*

Returns
Bool or fail

Description

This function allows you to change the list of regions available for saving. If you constructed with `globalsave=T`*, then you pass in a vector of region names. Otherwise, you pass in a record of regions.*

Example

```
- include 'saveregions.g'
- r1 := drm.box([10,10],[20,30]);
- r2 := drm.quarter();
- sr1 := saveregions(globalsave=T);
- sr1.setregions("r1 r2");
- r[1] := r1;
- r[2] := r2;
- sr2 := saveregions(globalsave=F);
- sr2.setregions(r);
```

We show how to set regions from the global name space and from a record.

---

saveregions.settable - Function

Package	**display**
Module	**widgets**
Tool	**saveregions**

Set a (new) table

Synopsis
settable(table)

Arguments

table	*The table of interest* *Allowed: Table tool, image tool or string* *Default: ''*

Returns
Bool

Description

This function manually (i.e. not via the GUI interface) sets a (new) Table name. It is equivalent to doing it via the GUI. The Table can be specified in many ways; a Table tool, an Image) tool, the name of a Table (a string) which might be an AIPS++ image, or the name of a Glish *symbol which is either a Table or Image tool.*

Example

```
- include 'saveregions.g'
- sr := saveregions();
- im := image('hcn2');
- t := table('hcn2');
- sr.settable(im);
- sr.settable('hcn2');
- sr.settable(t);
- sr.settable('im');
- sr.settable('t');
```

In this example, we show all the ways you can specify the table.

1.4.33 scrolllistbox - Tool

Package display
Module widgets

A Tk listbox with optional scrollbars.
include "scrolllistbox.g"

Constructors

scrolllistbox	*Construct a scrolllistbox*

Functions

done	*Destroy ourselves*
hscrollbar	*Turn on or off the horizontal scrollbar*
listbox	*Return the underlying listbox agent*
seeoninsert	*See the end after each insert?*
vscrollbar	*Turn on or off the vertical scrollbar*

Description

The scrolllistbox widget is a listbox with optional associated horizontal and vertical scrollbars. It has very nearly the same interface as the standard listbox widget.
The scrolllistbox widget supports all listbox widget events.
All events which are emitted by the underlying listbox are forwarded and will be seen to be emitted by the scrolllistbox.

Direct access to the underlying listbox is provided by the listbox function. This is necessary in order to support popup help on scrolllistboxes. Popup help uses the tlead argument in constructing the outframe. The value of that argument needs to be one of the underlying widgets hence the need to return a reference to the listbox used in scrolllistbox. Care should be taken in using this as it allows the user to destroy the underlying listbox.
Note that most of the functions listed here are actually events. These include hscrollbar, scrolllistbox, seeonsert, and vscrollbar. In addition to these events, any valid listbox event can be sent to the scrolllistbox.

Example

```
include 'scrolllistbox.g'
f := frame()
slb := scrolllistbox(f)
slb->insert("one two three four five six seven eight nine")
# you decide you don't want the horizontal scrollbar
slb->hscrollbar(F)
# you decide you don't want the widget to always show the end
# after each insert event
slb->seeoninsert(F)
# you want to get the current selection(s), if any
s := slb->selection();
```

scrolllistbox - Constructor

Package	**display**
Module	**widgets**
Tool	**scrolllistbox**

Construct a scrolllistbox

Synopsis
scrolllistbox(parent, hscrollbar, vscrollbar, vscrollbarright, seeoninsert, width, height, mode, font, relief, borderwidth, foreground, background, exportselection, fill, widgetset)

Arguments

parent	*Parent of the scrolllistbox* *Allowed: Tk frame*
hscrollbar	*Show a horizontal scrollbar?* *Allowed: Boolean* *Default: T*
vscrollbar	*Show a vertical scrollbar?* *Allowed: Boolean* *Default: T*
vscrollbarright	*Show vertical scrollbar on right? (F=left)* *Allowed: Boolean* *Default: T*
seeoninsert	*See the end after each insert?* *Allowed: Boolean* *Default: T*
width	*Width in characters units* *Allowed: Integer* *Default: 20*
height	*Height in characters units* *Allowed: Integer* *Default: 6*
mode	*Listbox mode* *Allowed: String from 'browse', 'single', 'extended', 'multiple'* *Default: 'browse'*
font	*Font of text* *Allowed: X font* *Default: ''*
relief	*Edge relief* *Allowed: String from 'flat', 'ridge', 'raised', 'sunken', 'groove'* *Default: 'sunken'*
borderwidth	*Border width* *Allowed: Integer* *Default: 2*
foreground	*Color of text* *Allowed: X color* *Default: 'black'*
background	*Background color* *Allowed: X color* *Default: 'lightgrey'*
exportselection	*export to X clipboard?* *Allowed: Boolean* *Default: F*
fill	*how to expand when resized* *Allowed: String from 'x', 'y', 'both', 'none'* *Default: 'both'*
widgetset	*Widget set to use* *Allowed: A widgetserver closure tool* *Default: dws*

Returns
scrolllistbox widget

Description
Constructs a scrolllistbox according to the specified construction arguments, and returns a ref to the widget.

---

scrolllistbox.listbox - Function

Package	**display**
Module	**widgets**
Tool	**scrolllistbox**

Return the underlying listbox agent

Synopsis
listbox()

Returns
Tk listbox agent

Description

This function returns the underlying listbox agent. Be careful when using this as it is possible to delete the underlying listbox. Also, insertions made directly to the listbox will not trigger the automatic "see" of the end of the listbox which would normally occur when seeoninsert is set to T.

---

scrolllistbox.done - Function

Package	**display**
Module	**widgets**
Tool	**scrolllistbox**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This function destroys the widget.

---

scrolllistbox.hscrollbar - Function

Package	**display**
Module	**widgets**
Tool	**scrolllistbox**

Turn on or off the horizontal scrollbar

Synopsis
hscrollbar(torf)

Arguments

torf	*Show a horizontal scrollbar?* *Allowed: Boolean*

Description
This is an event which turns on or off the horizontal scrollbar.

Example

```
# slb is a previously created scrolllistbox
# turn off the horizontal scrollbar
slb->hscrollbar(F);
# turn it back on
slb->hscrollbar(T);
```

---

scrolllistbox.vscrollbar - Function

Package	**display**
Module	**widgets**
Tool	**scrolllistbox**

Turn on or off the vertical scrollbar

Synopsis
vscrollbar(torf)

Arguments

torf	*Show a vertical scrollbar?* *Allowed: Boolean*

Description
This is an event which turns on or off the vertical scrollbar. If turned on, the constructor argument 'vscrollbarright' determines if the scrollbar is placed on the left or right side of the listbox.

---

scrolllistbox.seeoninsert - Function

Package display
Module widgets
Tool scrolllistbox

See the end after each insert?

Synopsis
seeoninsert(torf)

Arguments

torf	*See the end after each insert?* *Allowed: Boolean*

Description

This event turns on or off the action which this widget does after each insert event. When on (T) a "see" event is sent to the listbox to show the end of the listbox after each inset event. When off (T) no such "see" event is sent.

Example

```
# slb is a previously created scrolllistbox
# don't see the end after each insert event
slb->seeoninsert(F);
# see the after each insert event
slb->seeoninsertT);
```

1.4.34 scrolltext - Tool

Package display
Module widgets

A Tk text widget with optional scrollbars.
include "scrolltext.g"

Constructors

scrolltext	*Construct a scrolltext*

Functions

done	*Destroy ourselves*
hscrollbar	*Turn on or off the horizontal scrollbar*
text	*Return the underlying text widget agent*
vscrollbar	*Turn on or off the vertical scrollbar*

Description

The scrolltext widget is a text widget with optional associated horizontal and vertical scrollbars. It has very nearly the same interface as the standard text widget.
The scrolltext widget supports all text widget events.
All events which are emitted by the underlying text are forwarded and will be seen to be emitted by the scrolltext.
Direct access to the underlying text is provided by the text function. This is necessary in order to support popup help on scrolltexts. Popuphelp uses the tlead argument in constructing the outframe. The value of that argument needs to be one of the underlying widgets hence the need to return a reference to the listbox used in scrolltext. Care should be taken in using this as it allows the user to destroy the underlying text widget.
Note that most of the functions listed here are actually events. These include hscrollbar, scrolltext, and vscrollbar. In addition to these events, any valid text

widget event can be sent to the scrolltext.

Example

```
include 'scrolltext.g'
f := frame()
slb := scrolltext(f)
slb->append('\none \ntwo \nthree')
# you decide you don't want the horizontal scrollbar
slb->hscrollbar(F)
# you decide you don't want the widget to always show the end
```

scrolltext - Constructor

Package	**display**
Module	**widgets**
Tool	**scrolltext**

Construct a scrolltext

Synopsis

scrolltext(parent, hscrollbar, vscrollbar, width, height, wrap, font, disabled, text, relief, borderwidth, foreground, background, fill, widgetset)

Arguments

parent	*Parent of the scrolltext* *Allowed: Tk frame*
hscrollbar	*Show a horizontal scrollbar?* *Allowed: Boolean* *Default: T*
vscrollbar	*Show a vertical scrollbar?* *Allowed: Boolean* *Default: T*
width	*Width in characters units* *Allowed: Integer* *Default: 30*
height	*Height in characters units* *Allowed: Integer* *Default: 8*
wrap	*Wrap line behaviour* *Allowed: String from 'none', 'word', 'char'* *Default: 'none'*
font	*Font of text* *Allowed: X font* *Default: ''*
disabled	*Is widget disabled ?* *Allowed: T or F* *Default: F*
text	*Initial text* *Allowed: String* *Default: None*
relief	*Edge relief* *Allowed: String from 'flat', 'ridge', 'raised', 'sunken', 'groove'* *Default: 'flat'*
borderwidth	*Border width* *Allowed: Integer* *Default: 2*
foreground	*Color of text* *Allowed: X color* *Default: 'black'*
background	*Background color* *Allowed: X color* *Default: 'lightgrey'*
fill	*how to expand when resized* *Allowed: String from 'x', 'y', 'both', 'none'* *Default: 'both'*
widgetset	*Widget set to use* *Allowed: A widgetserver tool* *Default: dws*

Returns
scrolltext widget

Description
Constructs a scrolltext widget according to the specified construction arguments, and returns a ref to the widget.

---

scrolltext.text - Function

Package	**display**
Module	**widgets**
Tool	**scrolltext**

Return the underlying text widget agent

Synopsis
text()

Returns
Tk text agent

Description

This function returns the underlying text agent. Be careful when using this as it is possible to delete the underlying text widget.

---

scrolltext.done - Function

Package	**display**
Module	**widgets**
Tool	**scrolltext**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This function destroys the widget.

---

scrolltext.hscrollbar - Function

Package	**display**
Module	**widgets**
Tool	**scrolltext**

Turn on or off the horizontal scrollbar

Synopsis
hscrollbar(torf)

Arguments

torf	*Show a horizontal scrollbar?* *Allowed: Boolean*

Description
This is an event which turns on or off the horizontal scrollbar.

Example

```
# slb is a previously created scrolltext
# turn off the horizontal scrollbar
slb->hscrollbar(F);
# turn it back on
slb->hscrollbar(T);
```

---

scrolltext.vscrollbar - Function

Package	**display**
Module	**widgets**
Tool	**scrolltext**

Turn on or off the vertical scrollbar

Synopsis
vscrollbar(torf)

Arguments

torf	*Show a vertical scrollbar?* *Allowed: Boolean*

Description
This is an event which turns on or off the vertical scrollbar.

1.4.35 selectablelist - Tool

Package display
Module widgets

Tk widget for selection from a menu or a listbox depending on list length include "selectablelist.g"

Constructors

selectablelist	*Construct a selectablelist*

Functions

disabled	*Disable or enable the widget*
done	*Destroy ourselves*
replace	*Replace the menu with the provided items*
setbackground	*Set the color of the background of the button*
setforeground	*Set the color of the foreground of the button*
setlabel	*Set the label of the button*
setwidth	*Set the width of the selection button*

Description

This Tk widget enables you to select from a list of items. Depending on the

length of the list, either a menu or a scrollable listbox is used for the selection. This is useful because for very long lists, you don't want a menu.
If a menu is created, then the button that this widget makes is just a normal menu button. If a listbox is made, then the button is just a plain button. When you press it, the listbox is created close to the button.
With the listbox, there is an associated entry box into which you can type, and the the list is searched and the match highlighted as you type. If you hit carriage return, then the selection is made. Alternatively you could click on the highlighted entry.
Whenever a selection is made, this widget emits an event called `select`. *T $value of this event is a record with two fields. Field "item" contains the name of the selected item. Field "index" conatain the index of the selected item from the constructor list.*
The widget accepts an event called `setwidth`. *It is the same as calling the setwidth tool function.*
An example of the operation of this widget can be viewed in the measures GUI (start with dm.gui()), and goto e.g. Frame -¿ Which direction -¿ Source.

selectablelist - Constructor

Package	**display**
Module	**widgets**
Tool	**selectablelist**

Construct a selectablelist

Synopsis

selectablelist(parent, lead, list, nbreak, label, updatelabel, casesensitive, hlp, padx, pady, width, height, justify, font, relief, borderwidth, foreground, background, disabled, anchor, fill, widgetset)

Arguments

parent	*Parent of the menu for the menu/action button* *Allowed: Widget*
lead	*Listbox will appear to the sw of this agent* *Allowed: Agent*
list	*The list of items* *Allowed: Vector of strings*
nbreak	*The break point for menu or listbox usage* *Allowed: Integer* *Default: 20*
label	*The label for the button* *Allowed: Strings* *Default: Label*
updatelabel	*Update the text on the button with the selection* *Allowed: T or F* *Default: F*
casesensitive	*Is the text searching case sensitive* *Allowed: T or F* *Default: F*
hlp	*Short popuphelp text* *Allowed: String* *Default: No help*
padx	*Horizontal padding around text* *Allowed: Integer* *Default: 7*
pady	*Vertical padding around text* *Allowed: Integer* *Default: 3*
width	*Width in character units* *Allowed: Integer* *Default: Auto-scale*
height	*Height in character units* *Allowed: Integer* *Default: 1*
justify	*Justification of text* *Allowed: String from 'center', 'left', 'right'* *Default: 'center'*
font	*Font of text* *Allowed: X font*
relief	*Edge relief* *Allowed: String from 'flat', 'ridge', 'raised', 'sunken', 'groove'* *Default: 'groove'*
borderwidth	*Border width* *Allowed: Integer* *Default: 2*
foreground	*Color of text* *Allowed: X color* *Default: 'black*
background	*Background color* *Allowed: X color* *Default: 'lightgrey'*
disabled	*Is inactivated ?* *Allowed: Boolean* *Default: F*
anchor	*Location of text*

Returns
Agent

Description

The `width` *argument can be used in three ways. If* `width=-1` *(the default), the button will be wide enough (and of fixed width) to accomodate all of the item labels in the menu. This means that the GUI won't jitter when you write different length labels into it. If* `width=0`*, then the button will change its width to accomodate whatever label is written into it. If* `width > 0`*, then the button is again of fixed width, with the width that you specify.*
Note that all of the construction arguments refer to the button (plain for listbox or menu), rather than the listbox. You have no control over the form of the listbox (apart from `.aipsrc` *control.*

Example

```
- include 'selectablelist.g'
- list := "km hm dam m dm cm mm";
- f := frame();
- ls := selectablelist(parent=f, lead=f, list=list, nbreak=5, label='Lengths',
                    updatelabel=T);
```

A button labelled "Lengths" will appear. When it is pressed a listbox will popup to the south-west of the parent. When a selection is made, the label of the button will change to the selection.

---

selectablelist.disabled - Function

Package	**display**
Module	**widgets**
Tool	**selectablelist**

Disable or enable the widget

Synopsis
disabled(disable)

Arguments

disable	*disable the widget* *Allowed: T or F* *Default: T*

Returns
Bool

Description

This `tool` *function disables or enables the widget.*

---

selectablelist.done - Function

Package	**display**
Module	**widgets**
Tool	**selectablelist**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This `tool` *function destroys the widget.*

---

selectablelist.replace - Function

Package	**display**
Module	**widgets**
Tool	**selectablelist**

Replace the menu with the provided items

Synopsis
replace(lead, list, label, updatelabel, casesensitive, width)

Arguments

lead	*Listbox will appear to the sw of this agent* *Allowed: Agent*
list	*The list of items* *Allowed: Vector of strings*
label	*The label for the button* *Allowed: Strings* *Default: Label*
updatelabel	*Update the text on the button with the selection* *Allowed: T or F* *Default: F*
casesensitive	*Is the text searching case sensitive* *Allowed: T or F* *Default: F*
width	*Width in character units* *Allowed: Integer* *Default: Auto-scale*

Returns
Bool

Description

This `tool` *function replaces the list. The arguments are as for the constructor.*

---

selectablelist.setbackground - Function

Package **display**
Module **widgets**
Tool **selectablelist**

Set the color of the background of the button

Synopsis
setbackground(color)

Arguments

color	*The color* *Allowed: Any X color encoded as a String*

Returns
Bool

Description

This `tool` *function sets the color of the background of the button.*

---

selectablelist.setforeground - Function

Package display
Module widgets
Tool selectablelist

Set the color of the foreground of the button

Synopsis
setforeground(color)

Arguments

color	*The color* *Allowed: Any X color encoded as a String*

Returns
Bool

Description

This `tool` *function sets the color of the foreground of the button.*

---

selectablelist.setlabel - Function

Package display
Module widgets
Tool selectablelist

Set the label of the button

Synopsis
setlabel(text)

Arguments

text	*The label text* *Allowed: String*

Returns
Bool

Description

This `tool` *function sets the label of the button.*

---

selectablelist.setwidth - Function

Package	**display**
Module	**widgets**
Tool	**selectablelist**

Set the width of the selection button

Synopsis
setwidth(width)

Arguments

width	*The width in character units* *Allowed: Integer*

Returns
Bool

Description

This tool function sets the width of the selection button in character units.

1.4.36 synclistboxes - Tool

Package display
Module widgets

A collection of synchronized scrolllistboxes.
include "synclistboxes.g"

Constructors

synclistboxes	*Construct a synclistboxes*

Functions

background	*Set the background color.*
bind	*bind an event to a glish event*
borderwidth	*change the borderwidth(s)*
clear	*clear the selection at the indicated items*
delete	*delete the indicated items*
done	*Destroy ourselves*
exportselection	*export the selection to the X clipboard?*
font	*change the text font*
foreground	*Set the foreground color.*
get	*request items, i.e. map indexes to items*
height	*Set the height*
hscrollbar	*Turn on or off the horizontal scrollbars*
insert	*insert values into the listboxes*
leadbox	*which listbox is the default returned by the listbox function*
listbox	*Return an underlying listbox agent*
mode	*change listbox mode*
nearest	*return item nearest the y offset param*
relief	*change the border relief*
see	*scroll listboxes to specified index is visible*
seeoninsert	*See the end after each insert?*
select	*select the indicate items*
selection	*get indexes of the current selection*
vscrollbar	*Turn on or off the vertical scrollbar*
width	*Set the width*

Description

The synclistboxes widget is a collection of scrolllistboxes which are synchronized so that a selection in one is mirrored in all of the other listboxes and the vertical scrollbar controls all of the listboxes in the widget.
The synclistboxes widget supports all scrolllistbox widget events. However, in order to allow for fine control over the individual listboxes, many of the events have an optional additional argument which specifies the listbox which will receive that event. Each event is described below as a function due to the current limitations of the documentation system. There is only one actual function associated with this widget - listbox.
Direct access to the underlying listboxes is provided by the listbox function. This is necessary in order to support popup help on synclistboxeses. Popup help uses the tlead argument in constructing the outframe. The value of that argument needs to be one of the underlying widgets hence the need to return a reference to a true listbox. Care should be taken in using this as it allows the user to destroy the underlying listboxes and get them momentarily out of sync.

Example

```
include 'synclistboxes.g'
f := frame()
# two syncronized listboxes filled from left to right inside f
# the packing order is determined by the optional "side" argument
# to the synclistboxes function.  The listbox labels will be
# "Numbers" and "Colors".
slb := synclistboxes(f,2,"Numbers Colors");
# insert some values into the listboxes.  An equal number of values
# must be inserted into each listbox with each use of insert.
somevals := array(' ',2,9);
somevals[1,] := "one two three four five six seven eight nine";
somevals[2,] := "red orange yellow lightgrey blue brown peach mauve gold";
slb->insert(somevals);
# you decide you don't want the horizontal scrollbars
slb->hscrollbar(F)
# and you would like that listbox to only be 8 characters wide
slb->width(8, 1)
# you decide you don't want the widget to always show the end
# after each insert event
slb->seeoninsert(F)
# you want to get the current selection(s), if any
s := slb->selection();
# and you want to get all of the values at that selection
g := slb->get(s);
# g[1] will have the value from the first listbox at index s
```

```
# g[2] will have the value from the second listbox at index s
```

synclistboxes - Constructor

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

Construct a synclistboxes

Synopsis

synclistboxes(parent, nboxes, labels, leadbox, side, hscrollbar, vscrollbar, seeoninsert, width, height, mode, font, relief, borderwidth, foreground, background, exportselection, fill, widgetset)

Arguments

parent	*Parent of the synclistboxes* *Allowed: Tk frame*
nboxes	*The number of listboxes this widget will contain* *Allowed: Integer* *Default: 1*
labels	*The label(s)* *Allowed: String* *Default: ''*
leadbox	*The default box returned by the listbox function* *Allowed: Integer* *Default: 1*
side	*The packing order* *Allowed: String from 'left', 'right', 'top', 'bottom'* *Default: 'left'*
hscrollbar	*Show a horizontal scrollbar in all listboxes?* *Allowed: Boolean* *Default: T*
vscrollbar	*Show a vertical scrollbar?* *Allowed: Boolean* *Default: T*
seeoninsert	*See the end after each insert?* *Allowed: Boolean* *Default: T*
width	*Width(s) in characters units* *Allowed: Integer* *Default: 20*
height	*Height in characters units* *Allowed: Integer* *Default: 6*
mode	*Listbox mode* *Allowed: String from 'browse', 'single', 'extended', 'multiple'* *Default: 'browse'*
font	*Font(s) of text* *Allowed: X font* *Default: ''*
relief	*Edge relief(s)* *Allowed: String from 'flat', 'ridge', 'raised', 'sunken', 'groove'* *Default: 'sunken'*
borderwidth	*Border width* *Allowed: Integer* *Default: 2*
foreground	*Color(s) of text* *Allowed: X color* *Default: 'black'*
background	*Background color(s)* *Allowed: X color* *Default: 'lightgrey'*
exportselection	*export to X clipboard?* *Allowed: Boolean* *Default: F*
fill	*how to expand when resized* *Allowed: String from 'x', 'y', 'both', 'none'* *Default: 'both'*

Returns
synclistboxes widget

Description
Constructs a synclistboxes according to the specified construction arguments, and returns a ref to the widget.
The arguments described with (s), e.g. width, may be either single values or multiple values. A single value argument applies to all of the listboxes. A multiple value argument must have the same number of elements as the number of listboxes in the widget and each argument is applied, in turn, to the listboxes in the order in which they are created.

---

synclistboxes.done - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This function destroys the widget.

---

synclistboxes.listbox - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

Return an underlying listbox agent

Synopsis
listbox(whichbox)

Arguments

whichbox	*Which listbox to return* *Allowed: Integer or unset* *Default: current leadbox*

Returns
Tk listbox agent

Description

This function returns an underlying listbox agent. Be careful when using this as it is possible to delete the underlying listbox. Also, insertions made directly to the listbox will not trigger the automatic "see" of the end of the listbox which would normally occur when seeoninsert is set to T. It is also possible to get the listboxes out of sync using this function. The default listbox to return, if the whichbox argument is not supplied or is unset, is the listbox indicated by the leadbox argument.

---

synclistboxes.background - Function

Package display
Module widgets
Tool synclistboxes

Set the background color.

Synopsis
background(background, whichbox)

Arguments

background	*the new background color* *Allowed: X color*
whichbox	*which listbox gets the new color* *Allowed: An optional integer*

Description
This is an event which sets the background color one or all listboxes. If there is only one argument, all listboxes are set to that background color. If there is a second argument, it specifies which listbox should be set to that background color. Listboxes are numbered started at 1 up to the total number of listboxes. They are numbered in the order in which they were created.

Example

```
# slb is a previously created synclistboxes having 4 listboxes
# set the background color of all listboxes to lightgrey
slb->background('lightgrey');
# set the background color of listbox 3 to white
slb->background('white',3);
```

---

synclistboxes.bind - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

bind an event to a glish event

Synopsis
bind(xevent, gevent, whichbox)

Arguments

xevent	*the x event to be bound* *Allowed: X event*
gevent	*the glish event to be generated* *Allowed: String*
whichbox	*which listbox should this be bound to* *Allowed: An optional integer*

Description
This event works just like other Tk bind events in glish. The only difference is that there is an optional third argument which specifies which listbox this binding will apply to. If omitted, the binding applies to all listboxes in the widget. The widget

will emit an event having the name gevent whenever xevent occurs within the listboxes which the binding applies to.

Example

```
# slb is a previously created synclistboxes having 4 listboxes
# bind mouse button 2 to all listboxes
slb->bind('<Button-2>','mb2');
# any button 2 events in any listbox will result in an "mb2" event
# being seen from slb
# bind mouse button 3 to the 2nd listbox
slb->bind('<Button-3>', 'mb3', 2);
# any button 3 events which occur within listbox 2 will result in an
# "mb3" event being seen from slb.
```

---

synclistboxes.borderwidth - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

change the borderwidth(s)

Synopsis
borderwidth(borderwidth, whichbox)

Arguments

borderwidth	*the new borderwidth* *Allowed: Integer*
whichbox	*which listbox gets the new borderwidth* *Allowed: An optional integer*

Description
This is an event which sets the borderwidth for one or all listboxes. If there is only one argument, all listboxes are set to that borderwidth. If there is a second argument, it specifies which listbox should be set to that borderwidth.

Example

```
# slb is a previously created synclistboxes having 4 listboxes
# set the borderwidth of lb 2 to be 3
slb->borderwidth(3,2);
```

---

synclistboxes.clear - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

clear the selection at the indicated items

Synopsis

clear(start, end)

Arguments

start	*the start of the range to clear* *Allowed: string*
end	*optional end of the range to clear* *Allowed: string*

Description

This event is just like the normal listbox clear event. It is applied to all listboxes within the widget.

Example

```
# slb is a previously created synclistboxes
# clear all selections
slb->clear('start','end')
```

---

synclistboxes.delete - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

delete the indicated items

Synopsis
delete(start, end)

Arguments

start	*the start of the range to delete* *Allowed: string*
end	*optional end of the range to delete* *Allowed: string*

Description
This event is just like the normal listbox delete event. It is applied to all listboxes within the widget.

Example

```
# slb is a previously created synclistboxes
# delete everything
slb->delete('start','end')
```

---

synclistboxes.exportselection - Function

Package display
Module widgets
Tool synclistboxes

export the selection to the X clipboard?

Synopsis
exportselection(torf)

Arguments

torf	*export the selection to the X clipboard?* *Allowed: Boolean*

Description
This event is just like the normal listbox exportselection event. It is applied to all listboxes within the widget.

---

synclistboxes.foreground - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

Set the foreground color.

Synopsis
foreground(foreground, whichbox)

Arguments

foreground	*the new foreground color* *Allowed: X color*
whichbox	*which listbox gets the new color* *Allowed: An optional integer*

Description
This is an event which sets the foreground color in one or all listboxes. If there is only one argument, all listboxes are set to that foreground color. If there is a second argument, it specifies which listbox should be set to that foreground color. Listboxes are numbered started at 1 up to the total number of listboxes. They are numbered in the order in which they were created. See the description for the background event for a related example.

---

synclistboxes.font - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

change the text font

Synopsis

font(font, whichbox)

Arguments

font	*The new text font* *Allowed: X font*
whichbox	*which listbox gets the new font* *Allowed: An optional integer*

Description
This is an event which sets the text font in one or all listboxes. If there is only one argument, all listboxes are set to that text font. If there is a second argument, it specifies which listbox should be set to that text font.

---

synclistboxes.get - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

request items, i.e. map indexes to items

Synopsis
get(indexes)

Arguments

indexes	*the requested indexes* *Allowed: Integer(s)*

Returns
glish record of items

Description
All of the values in each of the listboxes at the requested indexes are returned by using this event. The returned value is a record which can be indexed by listbox number, i.e. if the return value is result, then result[1] contains the requested items for listbox 1, result[2] the items for listbox 2, etc.

Example

```
# slb is a previously created synclistboxes having 4 listboxes
# get the items at indexes [1,2,4]
g := slb->get([1,2,4]);
# g[1] has the values for listbox 1, g[2] for listbox 2, etc.
```

---

synclistboxes.height - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

Set the height

Synopsis
height(height)

Arguments

height	*the new height* *Allowed: Integer*

Description
This event is just like the normal listbox height event. It is applied to all listboxes within the widget.

---

synclistboxes.hscrollbar - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

Turn on or off the horizontal scrollbars

Synopsis
hscrollbar(torf)

Arguments

torf	*Show a horizontal scrollbar?* *Allowed: Boolean*

Description
This is an event which turns on or off all of the horizontal scrollbars. They must either be all on or all off to preserve syncronization.

Example

```
# slb is a previously created synclistboxes
# turn off the horizontal scrollbar
slb->hscrollbar(F);
# turn it back on
slb->hscrollbar(T);
```

---

synclistboxes.insert - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

insert values into the listboxes

Synopsis
insert(values)

Arguments

values	*insert these values* *Allowed: A vector or matrix of strings*

Description
In order to remain in sync, the listboxes must always each have the same number of items as the other listboxes. This is enforced by making the delete event apply to all listboxes and requiring that the insert event value have a integer multiple of nboxes elements. For a single vector there must be exactly nboxes elements. Each

element is then placed in each listbox in order. To place multiple elements in each listbox using a single insert event, the value of that event should have a shape equal to [nboxes, nelements]. In other words, to place 7 elements into each of 3 listboxes the insert value must have a shape of [3, 7].
Insertions are currently all done at the end of the listboxes.

Example

```
# slb is a previously created synclistboxes having 2 listboxes
# insert 9 elements.
somevals := array(' ',2,9);
somevals[1,] := "one two three four five six seven eight nine";
somevals[2,] := "red orange yellow lightgrey blue brown peach mauve gold";
slb->insert(somevals);
```

---

synclistboxes.leadbox - Function

Package display
Module widgets
Tool synclistboxes

which listbox is the default returned by the listbox function

Synopsis
leadbox(leadbox)

Arguments

leadbox	*which listbox takes the lead* *Allowed: Integer*

Description
As explain under the listbox function, popuphelp requires access to a true widget to act as the lead widget for positioning popuphelp. It knows to look for a listbox() function to supply that widget. The leadbox value determines which listbox is to act as the default lead widget when listbox is called with no arguments.

synclistboxes.mode - Function

Package display

Module widgets
Tool synclistboxes

change listbox mode

Synopsis
mode(mode)

Arguments

mode	*change listbox mode* *Allowed: String from 'browse', 'single', 'multiple', or 'extended'*

Description
This event is just like the normal listbox mode event. It is applied to all listboxes within the widget.

---

synclistboxes.nearest - Function

Package display
Module widgets
Tool synclistboxes

return item nearest the y offset param

Synopsis
nearest(nearest)

Arguments

nearest	*y offset param* *Allowed: Integer*

Returns
integer

Description
This event is just like the normal listbox nearest event. It is applied to first listbox (all listboxes would return the same value assuming they are all in sync).

---

synclistboxes.relief - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

change the border relief

Synopsis
relief(relief, whichbox)

Arguments

relief	*The new border relief* *Allowed: Standard relief string*
whichbox	*which listbox gets the new relief* *Allowed: An optional integer*

Description
This is an event which sets the border relieft in one or all listboxes. If there is only one argument, all listboxes are set to that relief. If there is a second argument, it specifies which listbox should be set to that text relief.

---

synclistboxes.see - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

scroll listboxes to specified index is visible

Synopsis
see(index)

Arguments

index	*the index to scroll to* *Allowed: String*

Description
This event is just like the normal listbox see event. It is applied to all listboxes within the widget.

---

synclistboxes.seeoninsert - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

See the end after each insert?

Synopsis
seeoninsert(torf)

Arguments

torf	*See the end after each insert?* *Allowed: Boolean*

Description

This event turns on or off the action which this widget does after each insert event. When on (T) a "see" event is sent to the listboxes to show the end of the listboxes after each inset event. When off (T) no such "see" event is sent.

Example

```
# slb is a previously created synclistboxes
# don't see the end after each insert event
slb->seeoninsert(F);
# see the after each insert event
slb->seeoninsertT);
```

---

synclistboxes.select - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

select the indicate items

Synopsis
select(indexex)

Arguments

indexex	*the indexes to select* *Allowed: String*

Description
This event is just like the normal listbox select event. It is applied to all listboxes within the widget.
This widget also emits a select event whenever the user makes a selection graphically.

---

synclistboxes.selection - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

get indexes of the current selection

Synopsis
selection()

Returns
integer

Description
This event is just like the normal listbox selection event. It is applied to the first

listbox in the widget (all the listboxes should return the same result if they are still in sync).
This widget also emits a select event whenever the user makes a selection graphically.

---

synclistboxes.vscrollbar - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

Turn on or off the vertical scrollbar

Synopsis
vscrollbar(torf)

Arguments

torf	*Show a vertical scrollbar?* *Allowed: Boolean*

Description
This is an event which turns on or off the verticall scrollbar. See the description of hscrollbar for a similar example.

---

synclistboxes.width - Function

Package	**display**
Module	**widgets**
Tool	**synclistboxes**

Set the width

Synopsis
width(width, whichbox)

Arguments

width	*The new width* *Allowed: Integer*
whichbox	*which listbox gets the new width* *Allowed: An optional integer*

Description
This is an event which sets the width of one or all listboxes. If there is only one argument, all listboxes are set to that width. If there is a second argument, it specifies which listbox should be set to that width.

1.4.37 tabdialog - Tool

Package display
Module widgets

Tk widget where the menu label changes with selection
include "tabdialog.g"

Constructors

tabdialog	*Construct an tabdialog*

Functions

add	*Add a new Tab*
available	*Is this Tab available*
delete	*Delete an existing Tab*
deleteall	*Delete all of the Tabs*
dialogframe	*Get the internal dialog frame*
done	*Destroy ourselves*
front	*Bring this Tab to the front*
list	*List Tabs*
replace	*Replace a new Tab*
which	*Which Tab is visible*

Description

Glish/Tk does not yet come with a Tab widget. When we are bound to a richer widget set such as Tix, such a widget will be available.
In the mean time, this widget simulates a Tab widget with a button bar.
The widget provides you a frame into which you place your Tabs. For each Tab

there is a button; pushing it maps the Tab to the foreground and the previous one to the background. The active tab is denoted by a sunken relief for the button. Otherwise, buttons are shown with raised relief.

tabdialog - Constructor

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Construct an tabdialog

Synopsis
tabdialog(parent, colmax, title, hlthickness, widgetset)

Arguments

parent	*Parent of the tabdialog* *Allowed: Agent*
colmax	*Maximum number of Tab buttons per row* *Allowed: Integer* *Default: 5*
title	*Title place above button row* *Allowed: String* *Default: 'Select'*
hlthickness	*Thickness of tab border* *Allowed: Integer* *Default: 5*
widgetset	*The widget server to use* *Allowed: widgetserver tool* *Default: dws*

Returns
Agent

Description

This is the constructor. If you assign `title=unset` *then no title will appear. As many rows of buttons as needed will be created as you add TABs.*

Example

```
include 'tabdialog.g'
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1', 'Press for tab1')
t.add(frames['2'], 'TAB2', 'Press for yet another tab')
```

This will make one row of buttons with two Tabs.

---

tabdialog.add - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Add a new Tab

Synopsis
add(tab, tabname, hlp)

Arguments

tab	*The Tab* *Allowed: Agent*
tabname	*Name of the tab for the tab button* *Allowed: String*
hlp	*Popup help text for tab button* *Allowed: String*

Returns
Bool

Description

This function adds the specified Tab and adds a button with the given name.
Note that adding a Tab does not make it the visible one unless it's the first Tab.

Example

```
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
```

This will make one row of buttons with two Tabs. The first Tab will be the visible one.

---

tabdialog.available - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Is this Tab available

Synopsis

available(tabname)

Arguments

tabname	*Name of the tab for the tab button* *Allowed: String*

Returns
Bool

Description

This function indicates whether the specified Tab name has been added or not.

Example

```
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
#
t.available('TAB1')
T
t.available('TAB10')
F
```

---

tabdialog.delete - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Delete an existing Tab

Synopsis
delete(tabname)

Arguments

tabname	*Name of the tab for the tab button* *Allowed: String*

Returns
Bool

Description

This function deletes the specified Tab. Subsequent additions will first fill this empty location.

Example

```
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
t.delete('TAB1')
```

This will make one row of buttons with two Tabs. The first Tab will be the visible one. The first tab is then deleted.

---

tabdialog.deleteall - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Delete all of the Tabs

Synopsis
deleteall()

Returns
Bool

Description

This function deletes all of the Tabs.

Example

```
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
#
t.deleteall()
```

---

tabdialog.dialogframe - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Get the internal dialog frame

Synopsis
dialogframe()

Returns
Agent

Description

This function recovers the internal Tab dialog frame into which you should place your Tabs.

Example

```
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
```

---

tabdialog.done - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Destroy ourselves

Synopsis
done()

Returns
Bool

Description

This function destroys the widget.

Example

```
- include 'tabdialog.g'
- f := dws.frame();
- t := tabdialog(f)
- t.done()
```

---

tabdialog.front - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Bring this Tab to the front

Synopsis

front(tabname)

Arguments

tabname	*Name of the tab* *Allowed: String*

Returns

Bool

Description

This function brings the specified Tab to the front.

Example

```
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
#
t.front('TAB2')
```

---

tabdialog.list - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

List Tabs

Synopsis
list()

Returns
Vector of strings

Description

This function returns the list of Tabs.

Example

```
f := dws.frame();
t := tabdialog(f)
```

```
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
#
t.list()
TAB1 TAB2
```

---

tabdialog.replace - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Replace a new Tab

Synopsis
replace(tab, tabname)

Arguments

tab	*The Tab* *Allowed: Agent*
tabname	*Name of the tab for the tab button* *Allowed: String*

Returns
Bool

Description

This function replaces the specified pre-existing Tab.

Example

```
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
#
frames['3'] := dws.frame(df)
frames['3']->unmap()
frames['3'].b := dws.button(frames['3'], 'b3')
#
t.replace(frames['3'], 'TAB2')
```

---

tabdialog.which - Function

Package	**display**
Module	**widgets**
Tool	**tabdialog**

Which Tab is visible

Synopsis
which()

Returns
Record

Description

This function returns a record containing the name of the Tab which is currently visible and its index (1 relative).

Example

```
f := dws.frame();
t := tabdialog(f)
df := t.dialogframe()
#
frames := [=]
frames['1'] := dws.frame(df)
frames['1']->unmap()
frames['1'].b := dws.button(frames['1'], 'b1')
frames['2'] := dws.frame(df)
frames['2']->unmap()
frames['2'].b := dws.button(frames['2'], 'b2')
#
t.add(frames['1'], 'TAB1')
t.add(frames['2'], 'TAB2')
#
t.front('TAB2')
t.which()
[name=TAB2, index=2]
```

1.4.38 tabdialogtest - Function

Package display
Module widgets

Test tabdialog widget

Synopsis
tabdialogtest()

Returns
Agent

Description

This function tests the tabdialog widget. It makes a GUI with some buttons you can press.

Example

```
- include 'tabdialog.g'
- tt := tabdialog()
```

1.4.39 tapedeck - Tool

Package **display**
Module **widgets**

Glish/Tk widget providing tapedeck-like control buttons
include "tapedeck.g"

Constructors

tapedeck	*Construct a tapedeck*

Functions

disable	*Disable the tapedeck controls*
dismiss	*Disable and dismiss the tapedeck*
done	*Disable and dismiss the tapedeck*
enable	*Enable the tapedeck controls*

Description

This Glish/Tk widget is simply a collection of upto seven buttons which can be used wherever operations like "play", "rewind" and "stop" are required. The constructor offers flexibility in which buttons are made available, and what colors they will be. An event is emitted whenever a button is pressed, and is one of: `tostart, reversestep, reverseplay, stop, forwardplay, forwardstep, toend`.

tapedeck - Constructor

Package **display**
Module **widgets**
Tool **tapedeck**

Construct a tapedeck

Synopsis

tapedeck(parent, background, hasstop, hasplay, hasstep, hasto, hasforward, hasreverse, stopcolor, playcolor, stepcolor, tocolor, orient, widgetset)

Arguments

parent	*Parent of the menu* *Allowed: Tk frame*
background	*Background color* *Allowed: X color* *Default: 'lightgrey'*
hasstop	*Whether to provide 'stop' button* *Allowed: Boolean* *Default: T*
hasplay	*Whether to provide 'play' button/s* *Allowed: Boolean* *Default: T*
hasstep	*Whether to provide 'step' button/s* *Allowed: Boolean* *Default: T*
hasto	*Whether to provide 'to' button/s (ie. to end, to start)* *Allowed: Boolean* *Default: T*
hasforward	*Whether to provide forward-direction buttons* *Allowed: Boolean* *Default: T*
hasreverse	*Whether to provide reverse-direction buttons* *Allowed: Boolean* *Default: T*
stopcolor	*Color of the 'stop' button* *Allowed: X color* *Default: 'black'*
playcolor	*Color of the 'play' button/s* *Allowed: X color* *Default: 'black'*
stepcolor	*Color of the 'step' button/s* *Allowed: X color* *Default: 'black'*
tocolor	*Color of the 'to' (cue) button/s* *Allowed: X color* *Default: 'black'*
orient	*Orientation of the tapedeck* *Allowed: 'horizontal' 'vertical'* *Default: 'horizontal'*
widgetset	*Widget set to use* *Allowed: A widgetserver closure tool* *Default: dws*

Returns
tapedeck widget

Description
Constructs a tapedeck according to the specified construction arguments, and returns a ref to the widget.

Example

```
include 'tapedeck.g'
f := frame();
td := tapedeck(f, background='black', hasto=F);
```

---

tapedeck.dismiss - Function

Package	**display**
Module	**widgets**
Tool	**tapedeck**

Disable and dismiss the tapedeck

Synopsis
dismiss()

Returns
None

Description
This provides a tidy way to remove a tapedeck once it is finished with. This is now deprecated - use 'done' instead.

Example

```
f := frame();
td := tapedeck(f);
# ... fill in whenevers ...
await td->stop;
td.dismiss();
```

---

tapedeck.done - Function

Package	**display**
Module	**widgets**
Tool	**tapedeck**

Disable and dismiss the tapedeck

Synopsis
done()

Returns
None

Description
This provides a tidy way to remove a tapedeck once it is finished with.

Example

```
f := frame();
td := tapedeck(f);
# ... fill in whenevers ...
await td->stop;
td.done();
```

---

tapedeck.enable - Function

Package	**display**
Module	**widgets**
Tool	**tapedeck**

Enable the tapedeck controls

Synopsis
enable()

Returns

None

Description
This will enable the tapedeck buttons, provided it is balanced by an equal number of calls to disable.

Example

```
f := frame();
td := tapedeck(f);
td.disable();
# ... fill in some whenevers
td.enable();
```

---

tapedeck.disable - Function

Package	**display**
Module	**widgets**
Tool	**tapedeck**

Disable the tapedeck controls

Synopsis
disable()

Returns
None

Description
This will disable the tapedeck buttons, provided it is balanced by an equal number of calls to enable.

Example

```
f := frame();
td := tapedeck(f);
```

```
td.disable();
# ... fill in some whenevers
td.enable();
```

1.4.40 taqlwidget - Tool

Package display
Module widgets

Widget to form a TaQL command
include "taqlwidget.g"

Description

The Table Query Language (TaQL) is an SQL-like language for defining commands to do selections and/or sorts on an AIPS++ table. It offers a lot of power and flexibility, but the syntax is not always clear. Furthermore, the names and data types of the columns in the table might not be known. This makes it sometimes hard to create a query command.
The TaQL widget is a Query-By-Example like widget to assist the user in creating a TaQL command. Such a command consists of four optional parts. Constructor arguments control if the user can specify these parts by enabling the appropriate parts of the widget.

- *The select-list tells which columns to select from the table.*
  A check box for each column name lets the user define the select-list. Pressing a button means that the column is added to the select-list in the entry box at the bottom. Pressing again removes it. It is also possible to edit the select-list directly.
  Note that an empty select-list means that all columns are selected.

- *The where-expression tells which rows to select from the table.*
  This part of the widget is always active and is meant to ease the process of creating the where-expression. The available columns and, if applicable, lists of possible values are shown in a form. The popuphelp shows the data type, etc.. The user can fill in the form to create (part of) the where-expression. Special buttons are available to create more complex where-expressions with parentheses and AND, OR, and NOT operators by combining the current contents of the form with the where-expression already present in the entry box at the bottom. It is also possible to edit the where-expression in the entry box directly.
  The expand button is only giving information. Pressing it shows how the current contents of the form convert to a where-expression

- *The sort-list tells if and how the resulting table is sorted.*
  A listbox is shown for each column to let the user add or remove the column to the sort-list in the entry box at the bottom. The user can choose No Sort, Ascending, or Descending. The No Sort choice removes the column from the sort-list. It is also possible to edit the sort-list directly.

- *The giving entry box lets the user specify the name of the resulting table. No name means that the result is transient.*

All these parts can be filled in manually, but it is easier to use the form at the top. It shows all available fields and if appropriate also a check box to tell if the field has to added to select list. It can also have a sort menu to add the field to the sort list.
Each field is followed by an operator menu and an entry box which can be used to fill in values. Multiple values are possible separated by commas. When a field can only have a limited set of values, the entry box is preceeded by a menu from which the user can select values. Fields of type string have some extra operators to compare the field with a regular expression (or filename-like pattern). It also has an option menu to make tell if case has to be ignored (default no) and if the string has to be enclosed in quotes (default yes). Note that in a TaQL command an unquoted string is treated as the name of another field.
To make the widget appear less overwhelming, the more advanced features (like the OR button and the resulting entry boxes) are hidden in a rollup. The rollup also has an expand button to show what the TaQL string looks like for the current contents of the form.
Constructing the widget is driven by the description of the table columns as given by the `table` function getdesc().
However, use of the widget is not limited to tables. As long as a correct description is given, the widget can be used to create query commands for other purposes as well.

taqlwidget - Constructor

Package	**display**
Module	**widgets**
Tool	**taqlwidget**

Construct TaQL widget

Synopsis
taqlwidget(cdesc, title, canselect, cansort, cangiving, heigth, giving, widgetset)

Description
This constructor creates the widget from the given description. The widget runs

asynchronously as a subsequence. When the user presses the `Go` *or* `Cancel` *button a* `returns` *event is generated which gives a record containing the fields* `select`*,* `where`*,* `orderby`*, and* `giving`*. These fields represent the four parts of a full TaQL command. All fields are empty if the* `Cancel` *button was pressed.*
The constructor fails if no GUI is active in the system.
The argument `cdesc` *determines the columns to be shown. It is a record in which each field describes a column. The field name is the column name; the field value is a record from which the following fields are used (if present):*

- `valueType` *defines the type. Possible values are string, integer, float, double, complex, dcomplex, and boolean. Its main use is in building the help info.*
  If the type is string a value is enclosed in quotes if necessary.

- `istime` *defines if the field contains a date/time (defaults to F). If so, some extra operations will be added automatically to handle the date/time correctly. If the valueType is string, the DATETIME function is applied to the field to convert the string to a date/time. Otherwise the field is a numeric date/time in seconds and is divided by 24*3600 to convert it to days. The MJD function is applied to the value to convert it to a double value.*

- `ndim` *defines the dimensionality of an array where 0 or -1 indicates any dimensionality. If the field is not present, the column is a scalar. It is only used in building the help info.*

- `shape` *defines the fixed shape of arrays in a column. It is only used in building the help info.*

- `comment` *can be used to give some extra information about the column. It is only used in building the help info.*

- `labels` *defines the possible values for the column if it is known that the column can only have a limited number of values. Those values are shown in an optionmenu where the user can select from. It should be a vector of type* `valueType`*.*

- `labelnames` *gives symbolic names for the possible values defined in* `labels`*. They are also shown in the optionmenu. They are only used if* `labels` *is given and if both vectors have the same length.*

The table description returned by table.getdesc() is a record with the fields shown above. Only the fields `istime`*,* `labels`*, and* `labelnames` *are not part of it. It means that a table description can directly be used in the* `taqlwidget` *constructor.*
The description can contain as many fields as one likes. When more fields than the widget heigth (argument `height`*) are given, a vertical scrollbar is added which can be used to scroll through all fields.*

The arguments canSomething determine if the user is allowed to build a select string, sort string, and giving string. If so, the widget is extended with the appropriate entry boxes, etc. to make it possible.

Arguments

cdesc	*in*	*The column descriptions* *Allowed: Record*
title	*in*	*Title string* *Allowed: String* *Default: Query Dialogue*
canselect	*in*	*Possible to build select string?* *Allowed: Bool* *Default: F*
cansort	*in*	*Possible to build sort string?* *Allowed: Bool* *Default: F*
cangiving	*in*	*Possible to build giving string?* *Allowed: Bool* *Default: F*
heigth	*in*	*Height of the field list* *Allowed: integer* *Default: 12*
giving	*in*	*default giving string* *Allowed: string* *Default: ''*
widgetset	*in*	*widgetset to be used* *Allowed: widgetset tool* *Default: dws*

Returns

subsequence

Example

```
include 'table.g'
include 'taqlwidget.g'
t := table('3C273XC1.MS');
q := taqlwidget(t.getdesc(), cansort=T);
await q->returns;
rec := $value;
t1 := t.query (rec.where, rec.giving, rec.orderby, rec.select);
```

Please note that the table.query() function supports `taqlwidget` *directly (in synchronous mode). The above example could have been written in a much easier way as:*

```
include 'table.g'
t := table('3C273XC1.MS');
t1 := t.query (unset);
```

1.4.41 twodactionoptionmenu - Tool

Package **display**
Module **widgets**

Pop-up, 'two-dimensional' menu
include "twodactionoptionmenu.g"

Description

This `tool` *is designed for use when a large number of menu items are required. It allows menu options (either icons or text) to be positioned in columns and rows. These menu options appear when a user clicks the button, which is represented by the default choice. The widget is similar to the actionoptionmenu widget, however it allows for columns of options, as opposed to a one dimensional list. It was designed for use with bitmaps in mind, however text buttons will also work, although they are not currently "justified". Currently, this widget is a little 'light on' for functionality - e.g. functions such as 'setvalue' etc (as opposed to the current method of setting by index) would be nice.*

twodactionoptionmenu - Constructor

Package **display**
Module **widgets**
Tool **twodactionoptionmenu**

Construct an optionmenu

Synopsis

twodactionoptionmenu(parent, images, names, values, ncolumn, hlp, hlp2, padx, pady, borderwidth, widgetset)

Arguments

parent	*Parent of the menu* *Allowed: Widget*
images	*Filenames of bitmaps to use* *Allowed: Vector of strings* *Default: Empty menu*
names	*The menu item names* *Allowed: Vector of strings* *Default: Empty menu*
values	*The menu item values* *Allowed: Vector of anything* *Default: Empty menu*
ncolumn	*Number of columns of buttons* *Allowed: Integer* *Default: 3*
hlp	*Short popuphelp text* *Allowed: String* *Default: No help*
hlp2	*Long popuphelp text* *Allowed: String* *Default: No help*
padx	*Horizontal padding around text* *Allowed: Integer* *Default: 2*
pady	*Vertical padding around text* *Allowed: Integer* *Default: 2*
borderwidth	*Border width* *Allowed: Integer* *Default: 2*
widgetset	*The widget server to use* *Allowed: widgetserver tool* *Default: dws*

Returns
Agent

Description
This is the constructor. The menu either uses icons (`images`*), or text labels (*`names`*). The array provided with the most elements is used. The number of elements in the array used must equal the number of elements in* `labels`*. The menu is built by construction the specified number of columns * required rows in order to fit all menu options on the pop up frame.*

Example

```
- include 'twodactionoptionmenu.g'
- names := ['Absolute', 'Relative to ref. pixel', 'Relative to center'];
- values := [1,2,3];
- f := frame();
- hlp := 'Select type of coordinate';
- m := twodactionoptionmenu(parent = f, names = names,
values = values, ncolumn = 2, hlp = hlp);
```

Creates a two dimensional option menu with 2 columns and required rows (in this case 2). If bitmaps were required instead:

```
- include 'twodactionoptionmenu.g'
- images := ['abs.xbm', 'rel.xbm', 'relcent.xbm'];
- values := [1,2,3];
- f := frame();
- hlp := 'Select type of coordinate';
- m := twodactionoptionmenu(parent = f, names = names,
values = values, ncolumn = 2, hlp = hlp);
```

---

twodactionoptionmenu.currentvalue - Function

Package	**display**
Module	**widgets**
Tool	**twodactionoptionmenu**

Return the currently selected value

Synopsis
currentvalue()

Arguments

Returns
Whatever is supplied in the value array on construction

Description

This function returns the `value` *of the currently selected option.*

---

twodactionoptionmenu.newselection - Function

Package **display**
Module **widgets**
Tool **twodactionoptionmenu**

Select a new option based on index

Synopsis
newselection(index)

Arguments

index	*The index of the new option* *Allowed: Integer*

Returns
Bool

Description

This sets the current selection to that specified. You must use the index, and not the value or names field to select a new option.

1.4.42 widgetserver - Tool

Package **display**
Module **widgets**

Tool to provide Tk widgets and AIPS++ megawidgets
include "widgetserver.g"

Constructors

widgetserver	*Construct a widgetserver*

Functions

actionoptionmenu	*Make an action optionmenu megawidget*
addpopuphelp	*add popup help to a GUI*
blinklabel	*Make a blinklabel megawidget*
busy	*Set the busy cursor and optionally disable a frame*
button	*Make a button widget*
canvas	*Make a canvas widget*
checkmenu	*Make a checkmenu megawidget*
combobox	*Make a combobox megawidget*
deleteregions	*Make a deleteregions megawidget*
dialogbox	*Make a dialogbox megawidget*
entry	*Make a entry widget*
extendoptionmenu	*Make an extendoptionmenu megawidget*
fontchooser	*Make a fontchooser megawidget*
frame	*Make a frame widget*
guientry	*Make a guientry megawidget*
have_gui	*Query whether GUIs can be built*
helpmenu	*Add a Help menu which contains the standard AIPS++ Help menu items*
label	*Make a label widget*
listbox	*Make a listbox widget*
menuframes	*Add a widget to hold named embedded frames*
message	*Make a message widget*
messageline	*Make a messageline megawidget*
multiscale	*Make a multiscale megawidget*
notbusy	*Set the normal cursor and optionally enable a frame*
optionmenu	*Make an optionmenu megawidget*
pgplot	*Make a pgplot widget*
popuphelp	*add popup help to a GUI*
popupmenu	*add a popup type selection help menu to a GUI*
popupremove	*remove popup help from a GUI*
popupselectmenu	*Make a popupselectmenu megawidget*
recordbrowser	*Make a recordbrowser megawidget*
resources	*Query what resources are used by various widgets*
restoreregions	*Make a restoreregions megawidget*
rollup	*Make a rollup megawidget*
saveregions	*Make a saveregions megawidget*
scale	*Make a scale widget*
scrollbar	*Make a scrollbar widget*
scrolllistbox	*Make a scrolllistbox megawidget*
selectablelist	*Make a selectablelist megawidget*
setmode	*Set the context/mode*
synclistboxes	*Make a synclistboxes megawidget*
tabdialog	*Make a tabdialog megawidget*
tapedeck	*Make a tapedeck megawidget*
text	*Make a text widget*

tk_hold	*Hold Glish/Tk widget events*
tk_iconpath	*???*
tk_release	*Release Glish/Tk widget events*
type	*Return the type of this tool*
unsetmode	*Unset (pop) the context/mode*

Description

The basic purpose of this tool is to provide wrappers for the Glish/Tk widgets and the AIPS++ megawidgets. The wrappers implement a new, context-sensitive set of defaults for the `relief, font, background` *and* `foreground` *arguments to widget constructors. In one context, the* `widgetserver` *will yield defaults which are identical to the standard Glish/Tk defaults; in another context, the* `widgetserver` *returns widgets which by default conform to the AIPS++ Graphical User Interface Guidelines (see AIPS++ Note 2??). As a programmer, you will almost always interact with* `defaultwidgetserver`*, which is provided upon inclusion of* `widgetserver.g`*, and is also known as* `dws` *for brevity.*
The `widgetserver` *now provides the capability for supporting multiple instances of the Glish/Tk client. If the construction argument* `widgetset` *is over-ridden from its default, then the constructed* `widgetserver` *will use the supplied Glish/Tk client instead of the default client. The default Glish/Tk client is stored in the global Glish record* `dgtk`*. A new Glish/Tk client can be made using the* `init_glishtk` *function.*
The `widgetserver` *has two modes:*

Mode 'glish': *widgets constructed by the* `widgetserver` *will have construction argument defaults that are identical to the "unwrapped" Glish/Tk constructors. For example, the code:*

```
include 'widgetserver.g';
dws.setmode('glish');
f1 := frame();
b1 := button(f1, 'Dismiss');
f2 := dws.frame();
b2 := dws.button(f2, 'Dismiss');
```

produces two identical *frames with* identical *"Dismiss" buttons in them.*

Mode 'app': *widgets constructed by the* `widgetserver` *will have construction argument defaults set such that the* `relief, font, background` *and* `foreground` *correspond to the AIPS++ GUI Guidelines for that particular widget. The AIPS++ GUI Guidelines specify, for example, that all "Dismiss" buttons should have orange backgrounds. Thus, the code:*

```
include 'widgetserver.g';
dws.setmode('app');
f1 := frame();
b1 := button(f1, 'Dismiss');
f2 := dws.frame();
b2 := dws.button(f2, 'Dismiss');
```

produces two identical *frames, but with* different *"Dismiss" buttons: the second one will have an orange background.*

The user is, however, able to control the `relief, font, background` *and* `foreground` *defaults for the* `'app'` *mode using their* `.aipsrc` *file. In the example above, they could make all "Dismiss" buttons created using a* `widgetserver` *in* `'app'` *mode have a yellow (rather than orange) background by adding the following line to their* `.aipsrc` *file:*

```
gui.prefs.dismissbutton.background: yellow
```

Any context-sensitive default implemented by a `widgetserver` *is always over-written (ie. ignored) if the programmer explicitly specifies that construction argument. For example, the following code:*

```
include 'widgetserver.g';
dws.setmode('glish');
f1 := dws.frame();
b1 := dws.button(f1, 'Dismiss', background='green');
dws.setmode('app');
f2 := dws.frame();
b2 := dws.button(f2, 'Dismiss', background='green');
```

produces two identical *frames, with* identical *green "Dismiss" buttons. In both cases, regardless of the mode of* `dws`*, the explicit* `background` *argument over-writes the defaults.*
Control of the various defaults for `'app'` *mode Glish/Tk widgets and AIPS++ megawidgets is provided in the* `.aipsrc` *file. All resources related to GUI* `relief, font, background` *and* `foreground` *defaults are stored in* `gui.prefs`.

Start-up mode: *The default start-up mode for newly constructed* `widgetservers` *can be set in the user's* `.aipsrc` *file as follows:*

```
gui.prefs.mode: glish
```

or

```
gui.prefs.mode: app
```

Fonts: *Four different fonts can be specified for use by the* `widgetserver`*:* `small, medium, large` *and* `bold`*. They can be specified using the following syntax in the* `.aipsrc` *file:*

```
gui.prefs.fonts.small: -*-helvetica-medium-r-normal-*-11-*
gui.prefs.fonts.medium: -*-helvetica-medium-r-normal-*-13-*
gui.prefs.fonts.large: -*-helvetica-medium-r-normal-*-18-*
gui.prefs.fonts.bold: -*-helvetica-bold-r-normal-*-13-*
```

The strings given must be valid X fonts. You might find it useful to use the X program **xfontsel** *to create X font strings. Glish/Tk does not deal particularly well with the case of ill-formed X font strings, or of missing fonts, so modify with these settings with caution. When using a* `widgetserver`*, fonts for particular widgets (eg. buttons, listboxes, etc.) can be specified using ordinary X font strings (examples above) or more simply the strings* `'small'`*,* `'medium'`*,* `'large'` *or* `'bold'`*. The* `widgetserver` *will detect what you have given, and map it accordingly.*

Frames: *The first place to start modifying your* `.aipsrc` *is probably the* `frame` *section. Any* `background`*,* `foreground` *and* `font` *settings applied to Glish/Tk frames are automatically forwarded on to* all *Glish/Tk widgets and AIPS++ megawidgets, where they can of course be over-written by specific default settings. The defaults for frames in the* `'app'` *mode can be set as follows:*

```
gui.prefs.frame.background: #c0d0e0
gui.prefs.frame.foreground: white
gui.prefs.frame.font: medium
gui.prefs.frame.relief: flat
```

Note that colors can be given as a string or as a hexadecimal RGB triple, preceded by a '#' character. The user is free to put any of the above lines in their `.aipsrc` *file and view the difference in GUIs built using the default* `widgetserver`*. The user need not give all the settings; just those they wish to modify.*

To indicate the behaviour of "cascading defaults," the following code:

```
dws.setmode('app');
f := dws.frame();
b := dws.button(f, 'Orangutan');
```

when the above four `gui.prefs.frame.*` *lines are installed in the user's* `.aipsrc` *file, would produce a pale blue button with white text. Since the user has not specified default* `font`*,* `foreground` *and* `background` *parameters for buttons, either in their* `.aipsrc` *file or explicitly in the Glish code, they will get the appropriate defaults from their* `.aipsrc` *settings for frames.*

Note that the `relief` *default for frames is not cascaded to lower level widgets, since relief is the primary—and often the* only*—visual distinction between the various Glish/Tk widgets.*

Buttons: *Three different types of buttons are known to the* `widgetserver`*: check, radio and plain buttons. The user can set* `.aipsrc` *defaults for each type, eg.:*

```
gui.prefs.plainbutton.background: #f0f0d0
gui.prefs.checkbutton.background: #f0f0d0
gui.prefs.checkbutton.relief: ridge
gui.prefs.radiobutton.background: #f0f0d0
gui.prefs.radiobutton.relief: ridge
```

Those preferences that are not specified default first to the `.aipsrc` *settings for frames (for* `font`, `foreground` *and* `background`*) if given, otherwise to the AIPS++ GUI Guidelines specifications.*

Special buttons: *In addition to the four standard button types (*`plain`, `check`, `radio` *and* `menu`*), the* `widgetserver` *knows three additional types:* `action`, `halt` *and* `dismiss`*. When asked to create a button of type* `halt`*, for example, the* `widgetserver` *will make the button with foreground, background and font according to preferences set in the user's* `.aipsrc` *file. These preferences can be set, for example, as follows:*

```
gui.prefs.dismissbutton.background: yellow
gui.prefs.actionbutton.font: bold
gui.prefs.haltbutton.relief: groove
```

The defaults, which apply when the user doesn't overwrite them in their `.aipsrc` *file, are for* `action` *buttons to be bold text on green background,* `halt` *buttons to be bold white text on a red background and* `dismiss` *buttons to be bold text on an orange background.*

Other Glish/Tk widgets: *Most remaining Glish/Tk widgets (scale, text, scrollbar, label, entry, message, listbox and canvas) can have preferences specified in the* `.aipsrc` *file in just the same way as frames and buttons. Note that for the canvas widget, only* `background` *and* `relief` *are meaningful, and for the scrollbar widget, only* `background` *and* `foreground` *preferences are useful. A summary of the syntax for the Glish/Tk widget preference settings in the* `.aipsrc` *file is:*

```
gui.prefs.widget.resource: value
```

where `widget` *is one of* `frame`, `plainbutton`, `radiobutton`, `checkbutton`, `gobutton`, `abortbutton`, `dismissbutton`, `scale`, `text`, `scrollbar`, `label`, `entry`, `message`, `listbox`, `canvas`*;* `resource` *is one of* `font`, `relief`, `foreground`, `background`*; and value is: one of* `small`, `medium`, `large`, `bold` *or an X font string for the* `font` *resource; one of* `flat`, `raised`, `groove`, `ridge`, `sunken` *for the* `relief` *resource; and a valid X color for the* `foreground` *and* `background` *resources.*

The Pgplot widget: *The pgplot widget is supplied and supported by the* `widgetserver`*, but at present default preferences are not supported.*

AIPS++ megawidgets: *Most AIPS++ megawidgets (rollup, optionmenu, ...), like the Glish/Tk widgets, are supported by the* `widgetserver`*, and have context-sensitive defaults. The user can specify them in an identical fashion to the previously listed "Other Glish/Tk widgets" (see above), where* `widget` *can also be one of* `rollup`, `optionmenu`, ... *The remaining AIPS++ megawidgets (dialogbox, ...) are supported by the* `widgetserver`*, but do not have context-sensitive defaults: they derive their defaults from the* `widgetserver`*-provided base Glish/Tk widgets.*

widgetserver - Constructor

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Construct a widgetserver

Synopsis
widgetserver(whichgtk)

Arguments

whichgtk	*Record holding interface to Glish/Tk client* *Allowed: Record of functions* *Default: dgtk*

Returns
widgetserver tool

Description
This constructs a widgetserver tool which can then be used to build AIPS++ GUIs. The only construction argument indicates which Glish/Tk client should be used to supply widgets, and should only be specified in exceptional circumstances.

Example

```
include 'widgetserver.g';
mywidgets := widgetserver();
```

widgetserver.frame - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a frame widget

Synopsis
frame(frame arguments)

Arguments

frame arguments	*standard Glish/Tk frame arguments*

Returns
Glish/Tk frame

Description
This makes a Glish/Tk frame widget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame(side='left');
```

---

widgetserver.button - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a button widget

Synopsis
button(button arguments)

Arguments

button arguments	*standard Glish/Tk button arguments*

Returns
Glish/Tk button

Description
This makes a Glish/Tk button widget, applying context-sensitive defaults. As well as the standard Glish/Tk button types (`plain, check, radio`* and *`menu`*), the three additional types* `action, halt` *and* `dismiss` *are provided—see the top-level widgetserver documentation.*

Example

```
f := dws.frame();
b := dws.button(f, 'Howdy');
```

---

widgetserver.scale - Function

Package display
Module widgets
Tool widgetserver

Make a scale widget

Synopsis
scale(scale arguments)

Arguments

scale arguments	*standard Glish/Tk scale arguments*

Returns
Glish/Tk scale

Description

This makes a Glish/Tk scale widget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame();
s := dws.scale(f, orient='vertical');
```

---

widgetserver.text - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a text widget

Synopsis
text(text arguments)

Arguments

text arguments	*standard Glish/Tk text arguments*

Returns
Glish/Tk text

Description
This makes a Glish/Tk text widget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame();
t := dws.text(f, text='Howdy');
```

---

widgetserver.scrollbar - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a scrollbar widget

Synopsis
scrollbar(scrollbar arguments)

Arguments

scrollbar arguments	*standard Glish/Tk scrollbar arguments*

Returns
Glish/Tk scrollbar

Description
This makes a Glish/Tk scrollbar widget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame();
s := dws.scrollbar(f, orient='horizontal');
```

---

widgetserver.label - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a label widget

Synopsis
label(label arguments)

Arguments

label arguments	*standard Glish/Tk label arguments*

Returns
Glish/Tk label

Description
This makes a Glish/Tk label widget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame();
l := dws.label(f, 'Howdy');
```

---

widgetserver.entry - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a entry widget

Synopsis
entry(entry arguments)

Arguments

entry arguments	*standard Glish/Tk entry arguments*

Returns
Glish/Tk entry

Description
This makes a Glish/Tk entry widget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame();
en := dws.entry(f, width=20);
```

---

widgetserver.message - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a message widget

Synopsis
message(message arguments)

Arguments

message arguments	*standard Glish/Tk message arguments*

Returns
Glish/Tk message

Description
This makes a Glish/Tk message widget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame();
m := dws.message(f, 'Howdy');
```

---

widgetserver.listbox - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a listbox widget

Synopsis
listbox(listbox arguments)

Arguments

listbox arguments	*standard Glish/Tk listbox arguments*

Returns
Glish/Tk listbox

Description
This makes a Glish/Tk listbox widget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame();
lb := dws.listbox(f, width=60);
```

---

widgetserver.canvas - Function

Package display
Module widgets
Tool widgetserver

Make a canvas widget

Synopsis
canvas(canvas arguments)

Arguments

canvas arguments	*standard Glish/Tk canvas arguments*

Returns
Glish/Tk canvas

Description
This makes a Glish/Tk canvas widget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame();
c := dws.canvas(f, 400, 300);
```

---

widgetserver.pgplot - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a pgplot widget

Synopsis
pgplot(pgplot arguments)

Arguments

pgplot arguments	*standard Glish/Tk pgplot arguments*

Returns
Glish/Tk pgplot

Description
This makes a Glish/Tk pgplot widget, without *applying context-sensitive defaults described in the top-level widgetserver documentation.*

Example

```
f := dws.frame();
pg := dws.pgplot(f);
```

---

widgetserver.have_gui - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Query whether GUIs can be built

Synopsis
have_gui()

Returns
Boolean indicated that GUIs can (T) or cannot (F) be built

Description
This is a wrapper for the bare have_gui function. Before building a GUI, it is advisable to check the return value of this function. See the Glish manual.

widgetserver.tk_hold - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Hold Glish/Tk widget events

Synopsis
tk_hold()

Description
This is a wrapper for the bare tk_hold function. Use this function to "pause" visible results of Glish/Tk widget construction, resizing and mapping/unmapping. See the Glish manual.

widgetserver.tk_release - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Release Glish/Tk widget events

Synopsis
tk_release()

Description
This is a wrapper for the bare tk_release function. Use this function to "unpause" visible results of Glish/Tk widget construction, resizing and mapping/unmapping. See the Glish manual.

---

widgetserver.tk_iconpath - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

???

Synopsis
tk_iconpath()

Description
??? See the Glish manual.

---

widgetserver.rollup - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a rollup megawidget

Synopsis
rollup(rollup megawidget arguments)

Arguments

rollup megawidget arguments	*standard AIPS++ rollup megawidget arguments*

Returns
AIPS++ rollup megawidget

Description
This makes an AIPS++ rollup megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame();
rup := dws.rollup(f);
```

---

widgetserver.optionmenu - Function

Package display
Module widgets
Tool widgetserver

Make an optionmenu megawidget

Synopsis
optionmenu(optionmenu megawidget arguments)

Arguments

optionmenu megawidget arguments	*standard AIPS++ optionmenu megawidget arguments*

Returns
AIPS++ optionmenu megawidget

Description
This makes an AIPS++ optionmenu megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame();
m := dws.optionmenu(f, labels="fish cats dogs", values=[0,1,2]);
```

---

widgetserver.extendoptionmenu - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make an extendoptionmenu megawidget

Synopsis
extendoptionmenu(extendoptionmenu megawidget arguments)

Arguments

extendoptionmenu megawidget arguments	*standard AIPS++ extendoptionmenu megawidget arguments*

Returns
AIPS++ extendoptionmenu megawidget

Description
This makes an AIPS++ extendoptionmenu megawidget, applying context-sensitive defaults. It differes from the optionmenu megawidget in that the menu can be interactively extended. In addition, you can only specify `labels`, `names` *and* `values` *are not allowed.*

Example

```
f := dws.frame();
m := dws.optionmenu(f, labels="fish cats dogs");
```

---

widgetserver.actionoptionmenu - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make an action optionmenu megawidget

Synopsis
actionoptionmenu(actionoptionmenu megawidget arguments)

Arguments

actionoptionmenu megawidget arguments	*standard AIPS++ actionoptionmenu megawidget arguments*

Returns
AIPS++ actionoptionmenu megawidget

Description

This makes an AIPS++ actionoptionmenu megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation. It is identical to the optionmenu megawidget. The `actionoptionmenu` *exists only so you can use it for "action" menus, where the relief (usually raised) and colors will in general be different from that of a "selection" option menu (relief is usually groove).*
The `actionoptionmenu` *is implemented on top of the* `optionmenu` *in the widgetserver tool. This means that it is only available via the* `widgetserver`.

Example

```
- include 'widgetserver.g'
- names := ['Delete all files', 'Delete images', 'Delete measurement sets'];
- labels := "DelAll DelIm DelMS";
- f := dws.frame();
- hlp := 'Delete selected files';
- m := dws.actionoptionmenu(f, labels, names, hlp=hlp);
```

---

widgetserver.checkmenu - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a checkmenu megawidget

Synopsis
checkmenu(checkmenu megawidget arguments)

Arguments

checkmenu megawidget arguments	*standard AIPS++ checkmenu megawidget arguments*

Returns
AIPS++ actionoptionmenu megawidget

Description
This makes an AIPS++ checkmenu megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

---

widgetserver.dialogbox - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a dialogbox megawidget

Synopsis
dialogbox(megawidget arguments)

Arguments

megawidget arguments	*standard AIPS++ dialogbox megawidget arguments*

Returns
AIPS++ dialogbox megawidget

Description

This makes an AIPS++ dialogbox megawidget, without *applying context-sensitive defaults described in the top-level widgetserver description. Nevertheless, the dialogbox will be produced in the current mode of the widgetserver.*

Example

```
db := dws.dialogbox('Enter your name');
```

---

widgetserver.guientry - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a guientry megawidget

Synopsis
guientry(guientry megawidget arguments)

Arguments

guientry megawidget arguments	*standard AIPS++ guientry megawidget arguments*

Returns
AIPS++ guientry megawidget

Description
This makes an AIPS++ guientry megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

---

widgetserver.messageline - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a messageline megawidget

Synopsis
messageline(messageline megawidget arguments)

Arguments

messageline megawidget arguments	*standard AIPS++ messageline megawidget arguments*

Returns
AIPS++ messageline megawidget

Description
This makes an AIPS++ messageline megawidget, applying context-sensitive defaults as described in the top-level widgetserver description as described in the top-level widgetserver description.

Example

```
f := dws.frame();
ml := dws.messageline(f);
ml->post('This is an informative message which is also logged.');
```

---

widgetserver.scrolllistbox - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a scrolllistbox megawidget

Synopsis
scrolllistbox(scrolllistbox megawidget arguments)

Arguments

scrolllistbox megawidget arguments	*standard AIPS++ scrolllistbox megawidget arguments*

Returns
AIPS++ scrolllistbox megawidget

Description
This makes an AIPS++ scrolllistbox megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame();
slb := dws.scrolllistbox(f);
slb->insert("one two three four five six seven eight nine");
```

---

widgetserver.synclistboxes - Function

Package display
Module widgets
Tool widgetserver

Make a synclistboxes megawidget

Synopsis
synclistboxes(synclistboxes megawidget arguments)

Arguments

synclistboxes megawidget arguments	*standard AIPS++ synclistboxes megawidget arguments*

Returns
AIPS++ synclistboxes megawidget

Description
This makes an AIPS++ synclistboxes megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame();
slb := dws.synclistboxes(f,2,"Numbers Colors");
somevals := array(' ',2,9);
somevals[1,] := "one two three four five six seven eight nine";
somevals[2,] := "red orange yellow lightgrey blue brown peach mauve gold";
slb->insert(somevals);
```

---

widgetserver.combobox - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a combobox megawidget

Synopsis
combobox(combobox megawidget arguments)

Arguments

combobox megawidget arguments	*standard AIPS++ combobox megawidget arguments*

Returns
AIPS++ combobox megawidget

Description
This makes an AIPS++ combobox megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame();
cb := dws.combobox(f);
cb.insert("one two three four five six seven eight nine");
whenever cb.agent()->select do {
   print 'combobox select event: new selection, item ', $value;
   print '   value at selection is : ', cb.get($value);
}
```

---

widgetserver.blinklabel - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a blinklabel megawidget

Synopsis
blinklabel(blinklabel megawidget arguments)

Arguments

blinklabel megawidget arguments	*standard AIPS++ blinklabel megawidget arguments*

Returns
AIPS++ blinklabel megawidget

Description
This makes an AIPS++ blinklabel megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame();
fw := dws.blinklabel(f, 'Conditions changed');
```

---

widgetserver.fontchooser - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a fontchooser megawidget

Synopsis
fontchooser(fontchooser megawidget arguments)

Arguments

fontchooser megawidget arguments	*standard AIPS++ fontchooser megawidget arguments*

Returns
AIPS++ fontchooser megawidget

Description
This makes an AIPS++ fontchooser megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

---

widgetserver.multiscale - Function

Package **display**
Module **widgets**
Tool **widgetserver**

Make a multiscale megawidget

Synopsis
multiscale(multiscale megawidget arguments)

Arguments

multiscale megawidget arguments	*standard AIPS++ multiscale megawidget arguments*

Returns
AIPS++ multiscale megawidget

Description
This makes an AIPS++ multiscale megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame();
fw := dws.multiscale(f, start=-30.0, end=30.0, resolution=0.2);
```

---

widgetserver.popupselectmenu - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a popupselectmenu megawidget

Synopsis
popupselectmenu(popupselectmenu megawidget arguments)

Arguments

popupselectmenu megawidget arguments	*standard AIPS++ popupselectmenu megawidget arguments*

Returns
AIPS++ popupselectmenu megawidget

Description
This makes an AIPS++ popupselectmenu megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

---

widgetserver.recordbrowser - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a recordbrowser megawidget

Synopsis

recordbrowser(recordbrowser megawidget arguments)

Arguments

recordbrowser megawidget arguments	*standard AIPS++ recordbrowser megawidget arguments*

Returns
AIPS++ recordbrowser megawidget

Description
This makes an AIPS++ recordbrowser megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

---

widgetserver.restoreregions - Function

Package display
Module widgets
Tool widgetserver

Make a restoreregions megawidget

Synopsis
restoreregions(restoreregions megawidget arguments)

Arguments

restoreregions megawidget arguments	*standard AIPS++ restoreregions megawidget arguments*

Returns
AIPS++ restoreregions megawidget

Description
This makes an AIPS++ restoreregions megawidget, applying context-sensitive defaultsas described in the top-level widgetserver description .

---

widgetserver.deleteregions - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a deleteregions megawidget

Synopsis
deleteregions(deleteregions megawidget arguments)

Arguments

deleteregions megawidget arguments	*standard AIPS++ deleteregions megawidget arguments*

Returns
AIPS++ deleteregions megawidget

Description
This makes an AIPS++ deleteregions megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

---

widgetserver.saveregions - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a saveregions megawidget

Synopsis
saveregions(saveregions megawidget arguments)

Arguments

saveregions megawidget arguments	*standard AIPS++ saveregions megawidget arguments*

Returns
AIPS++ saveregions megawidget

Description
This makes an AIPS++ saveregions megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

---

widgetserver.selectablelist - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a selectablelist megawidget

Synopsis
selectablelist(selectablelist megawidget arguments)

Arguments

selectablelist megawidget arguments	*standard AIPS++ selectablelist megawidget arguments*

Returns
AIPS++ selectablelist megawidget

Description
This makes an AIPS++ selectablelist megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

---

widgetserver.tapedeck - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a tapedeck megawidget

Synopsis
tapedeck(tapedeck megawidget arguments)

Arguments

tapedeck megawidget arguments	*standard AIPS++ tapedeck megawidget arguments*

Returns
AIPS++ tapedeck megawidget

Description
This makes an AIPS++ tapedeck megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

---

widgetserver.tabdialog - Function

Package **display**
Module **widgets**
Tool **widgetserver**

Make a tabdialog megawidget

Synopsis
tabdialog(tabdialog megawidget arguments)

Arguments

tabdialog megawidget arguments	*standard AIPS++ tabdialog megawidget arguments*

Returns
AIPS++ tabdialog megawidget

Description
This makes an AIPS++ tabdialog megawidget, applying context-sensitive defaults as described in the top-level widgetserver documentation.

Example

```
f := dws.frame();
t := dws.tabdialog(f);
```

---

widgetserver.guientry - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Make a guientry megawidget

Synopsis
guientry(guientry megawidget arguments)

Arguments

guientry megawidget arguments	*standard AIPS++ guientry megawidget arguments*

Returns
AIPS++ guientry megawidget

Description
This makes an AIPS++ guientry megawidget server, applying context-sensitive defaults as described in the top-level widgetserver description. Note that this is not a widget but itself serves widgets, as indicated in the example.

Example

```
f := dws.frame();
ge := dws.guientry();
s := ge.string(f, 'Initial string')
```

---

widgetserver.setmode - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Set the context/mode

Synopsis
setmode(mode)

Arguments

mode	*Which mode to use* *Allowed: String from 'app', 'glish'*

Description
This function is used to set the mode of a widgetserver to either `'app'` *or* `'glish'`*. Before building a GUI, you should set what mode you would like the widgetserver to be in, then build your GUI. For example:*

```
dws.setmode('glish');
f := dws.frame();
# ...
```

would produce a GUI with standard Glish/Tk default colors, font and relief for the various widgets. The widgetserver stores its previous mode/s, so it is recommended that after building your GUI, you restore the previous mode with the `unsetmode` *function. Normally, AIPS++ GUIs should be written with the mode set to* `'app'`*.*

---

widgetserver.unsetmode - Function

Package display
Module widgets
Tool widgetserver

Unset (pop) the context/mode

Synopsis
unsetmode()

Description
This function should be used to restore the previous context or mode that was in effect just prior to the most recent `setmode` *call. For example, if for some reason, you need to create a non-AIPS++ conformant GUI, you might do:*

```
dws.setmode('glish');
f := dws.frame();
# build GUI ...
# ...
dws.unsetmode();
```

---

widgetserver.resources - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Query what resources are used by various widgets

Synopsis

resources(widget, type)

Arguments

widget	*Which widget to query* *Allowed: String from 'button', 'optionmenu', etc*
type	*Subtype of button widgets* *Allowed: String from 'plain', 'check', 'radio', 'menu', 'action', 'halt', 'dismiss'* *Default: F*

Description

This function is provided so that programmers can find out what relief, font and foreground and background colors are used by the various widgets that the widgetserver delivers. Generally, just the name of the widget needs to be supplied as a string argument. In the special case of 'button' widgets, an additional argument is given so that the type of the button can be specified. The function returns a record whose fields (upto four of them) contain the resources used by the widget.

Example

```
rec := dws.resources('button', 'action');
print rec.background;
```

---

widgetserver.helpmenu - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Add a Help menu which contains the standard AIPS++ Help menu items

Synopsis
helpmenu(helpmenu megawidget arguments)

Arguments

helpmenu megawidget arguments	*standard AIPS++ helpmenu megawidget arguments*

Returns
AIPS++ helpmenu megawidget

Description
This makes an AIPS++ helpmenu megawidget. The default font, color, etc., are those for the button widget.

Example

```
f := dws.frame();
# f would generally be a menubar with appropriate padding such that the
# helpmenu frame is always all the way on the right side
helpmenuframe := dws.frame(f,borderwidth=0,height=0,width=0,side='right');
hm := dws.helpmenu(helpmenuframe, "Imager", "Refman:imager");
```

---

widgetserver.menuframes - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Add a widget to hold named embedded frames

Synopsis
menuframes(menuframes megawidget arguments)

Arguments

menuframes megawidget arguments	*standard AIPS++ menuframes megawidget arguments*

Returns
AIPS++ menuframes megawidget

Description
This makes an AIPS++ menuframes megawidget.

Example

```
f := dws.frame();
f1 := dws.frame();
mb := dws.button(f1, 'Windows', type='menu')
r := dws.menuframes(f, mb, exclusive=F);
```

---

widgetserver.busy - Function

Package display
Module widgets
Tool widgetserver

Set the busy cursor and optionally disable a frame

Synopsis
busy(busyframe, disable, busycursor)

Arguments

busyframe	*The frame which will appear busy* *Allowed: Frame*
disable	*Should a disable event be sent to this frame?* *Allowed: T or F* *Default: T*
busycursor	*The new cursor* *Allowed: String* *Default: 'watch'*

Returns
T

Description
This function changes the cursor of `busyframe` *to the* `busycursor`*, which defaults to "watch", to indicate that that frame is busy. If* `disable` *is* `T`*, then a disable event is also sent to that frame, disabling all widgets within* `busyframe`*. When* `disable` *is* `T` *it must be matched either by a call to* `notbusy` *with* `enable` *being* `T` *or an enable event must be explicitly sent to that frame.*

Example

```
- f := dws.frame()
- # some additional widgetserver calls to build the contents of f
- # to set the busy cursor in f and disable the widgets in f
- dws.busy(f)
- # to set the normal cursor in f and enable its widgets
- dws.notbusy(f)
```

---

widgetserver.notbusy - Function

Package **display**
Module **widgets**
Tool **widgetserver**

Set the normal cursor and optionally enable a frame

Synopsis
notbusy(busyframe, enable, normalcursor)

Arguments

busyframe	*The frame which will appear to not be busy* *Allowed: Frame*
enable	*Should an enable event be sent to this frame?* *Allowed: T or F* *Default: T*
normalcursor	*The new cursor* *Allowed: String* *Default: ''*

Returns
T

Description
This function changes the cursor of `busyframe` *to the* `normalcursor` *to indicate that that frame is not busy. The default value of* `normalcursor` *will set the cursor of* `busyframe` *to its normal (default) value. If* `enable` *is* `T`*, then an enable event is also sent to that frame, enabling all widgets within* `busyframe`*. When* `enable` *is* `T` *it should be matched either by a previous call to* `busy` *with* `disable` *being* `T` *or a disable event should have been explicitly sent to that frame prior to using* `enable`*.*

Example

```
- f := dws.frame()
- # some additional widgetserver calls to build the contents of f
- # to set the busy cursor in f and disable the widgets in f
- dws.busy(f)
- # to set the normal cursor in f and enable its widgets
- dws.notbusy(f)
```

---

widgetserver.popuphelp - Function

Package display
Module widgets
Tool widgetserver

add popup help to a GUI

Synopsis
popuphelp()

Description
This function forwards all its arguments to the popuphelp function in the guiutils module. See the documention in that module for a full description.

---

widgetserver.addpopuphelp - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

add popup help to a GUI

Synopsis
addpopuphelp()

Description
This function forwards all its arguments to the addpopuphelp function in the guiutils module. See the documention in that module for a full description.

---

widgetserver.popupmenu - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

add a popup type selection help menu to a GUI

Synopsis
popupmenu()

Description
This function forwards all its arguments to the popupmenu function in the guiutils module. See the documention in that module for a full description.

---

widgetserver.popupremove - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

remove popup help from a GUI

Synopsis
popupremove()

Description
This function forwards all its arguments to the popupremove function in the guiutils module. See the documention in that module for a full description.

---

widgetserver.type - Function

Package	**display**
Module	**widgets**
Tool	**widgetserver**

Return the type of this tool

Synopsis
type()

Returns
'widgetserver'

Description
This function returns in a string the type of the tool; 'widgetserver'.

1.4.43 is_widgetserver - Function

Package	**display**
Module	**widgets**

Is the argument a widgetserver tool?

Synopsis
is_widgetserver(tool)

Description
This global function can be used to determine if the supplied argument is a widgetserver tool. If so it returns True, otherwise it returns False.

Arguments

tool	*The variable that you wish to test* *Allowed: Any glish variable* *Default: no default*

Returns
T or F, indicating if the argument is a widgetserver tool or not.

Example

```
include 'widgetserver.g'
if (!is_widgetserver(dws)) {
  fail 'Default widgetserver not available!';
}
```

This example shows how you can use this function to determine if the supplied variable is a widgetserver.

A note on colors

At present, the Aipsview and Viewer `tools` *will only work on X window screens which can provide a "PseudoColor visual." Most present-day Sun workstations provide an 8-bit PseudoColor visual by default; most present-day SGI workstations provide a 10-bit PseudoColor visual by default; and most Linux workstations and X-terminals offer a 16-bit TrueColor visual by default, but commonly offer instead a 15-bit TrueColor visual or a 24-bit TrueColor visual. Some TrueColor X servers (notably the Sun servers, and a few commercial servers running under Linux on selected hardware) are actually able to provide PseudoColor visuals within selected windows.*

Without a PseudoColor visual, the Aipsview and Viewer `tools` **will fail. This is because additional work is needed for these** `tools` **to support TrueColor (and DirectColor) visuals correctly.**

However, it is important to note that for standard false color images, PseudoColor visuals are generally better than TrueColor visuals. This is because PseudoColor visuals have "hardware colormaps," into which selected colors can be placed for use. A false color image is normally displayed by first mapping a set of colors (eg. a rainbow or greyscale) into part of the hardware colormap, and then drawing the pixels to the screen, specifying for each pixel with which entry in the hardware colormap it should color itself. Once drawn, the colors can be changed in the hardware colormap, and the display is updated instantly. For example, the rainbow colors could be replaced with topographic colors. Or as is more common, the colors within the hardware colormap can be shifted lengthwise and compressed or stretched to give more range to the signal or noise regimes of an image: this is the standard "colormap fiddling" technique.

TrueColor visuals, on the other hand, do not have a hardware colormap, and so for each pixel the exact color must be specified by red, green and blue components at the time of drawing. *Once drawn, the only way to modify the coloring of the image is to completely redraw the image. Thus, the ability to interactive fiddle the colormap of an image is lost.*

Furthermore, while a screen having an 8-bit PsuedoColor visual can display at maximum 256 colors on the screen at once, and a 15-bit TrueColor visual can display upto 32768 colors, in the second case, the individual color components (red, green and blue) can be specified at only 5-bit resolution. For the former, 8-bit resolution is available for the color components. Thus, smooth color shadings are easier to obtain on an 8 or 10-bit PseudoColor visual than on the typical 15/16-bit TrueColor visuals used on Linux machines.

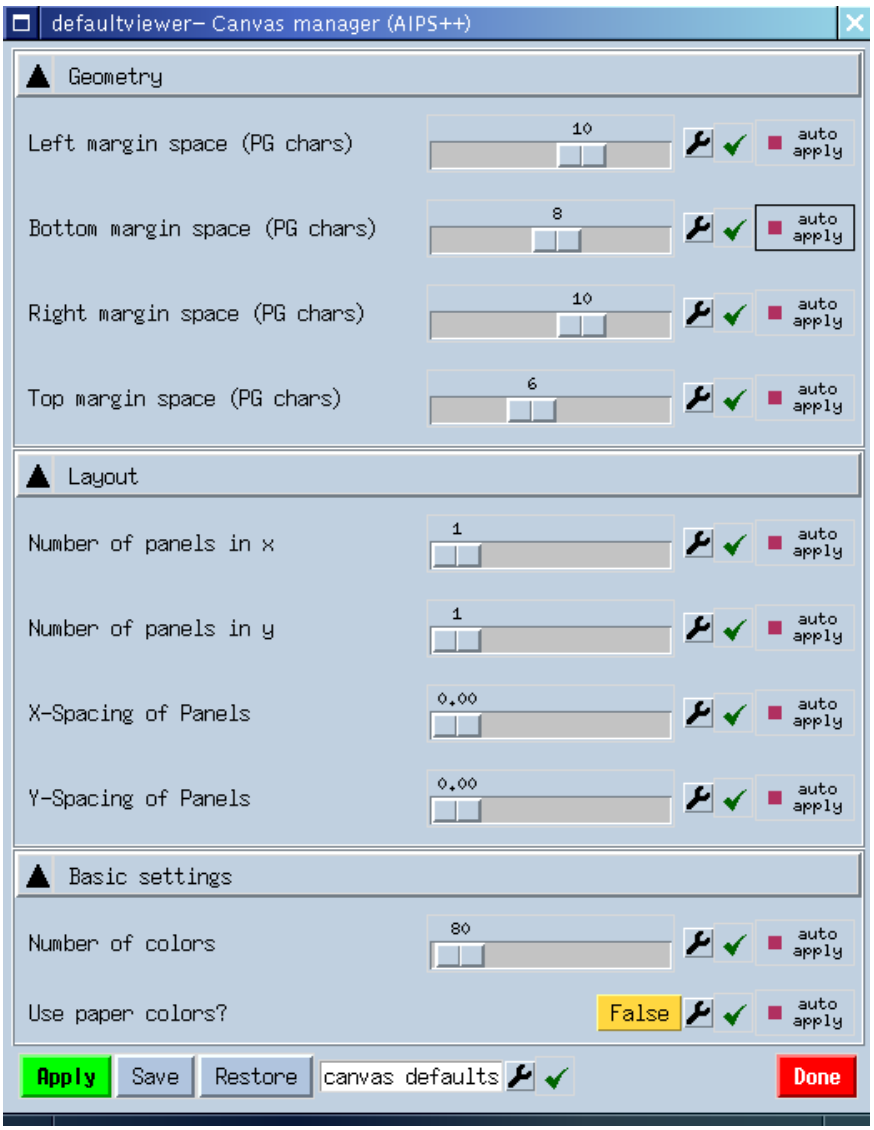

Figure 1.1: The Viewercanvasmanager `tool` graphical user interface

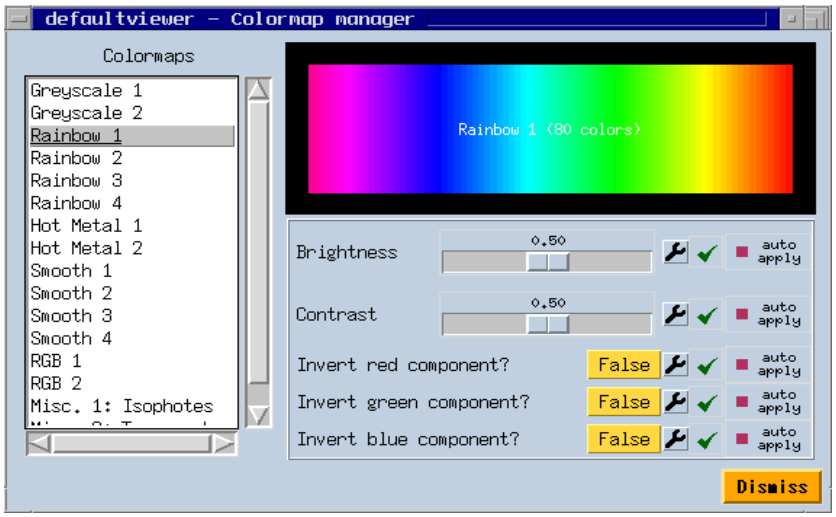

Figure 1.2: The Viewercolormapmanager `tool` graphical user interface

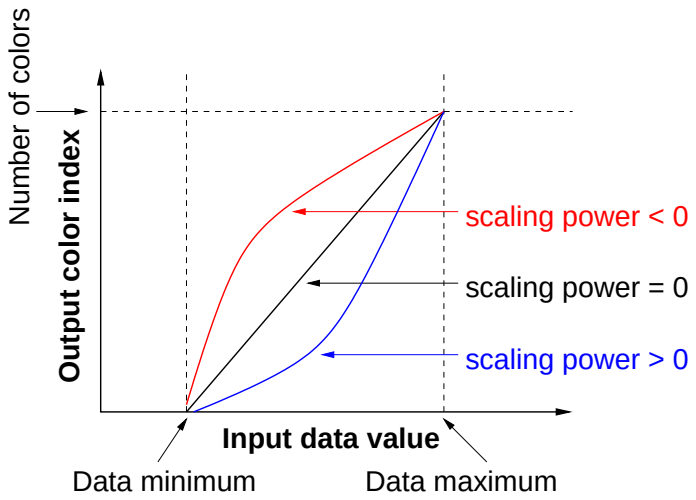

Figure 1.3: Example curves for scaling power cycles.

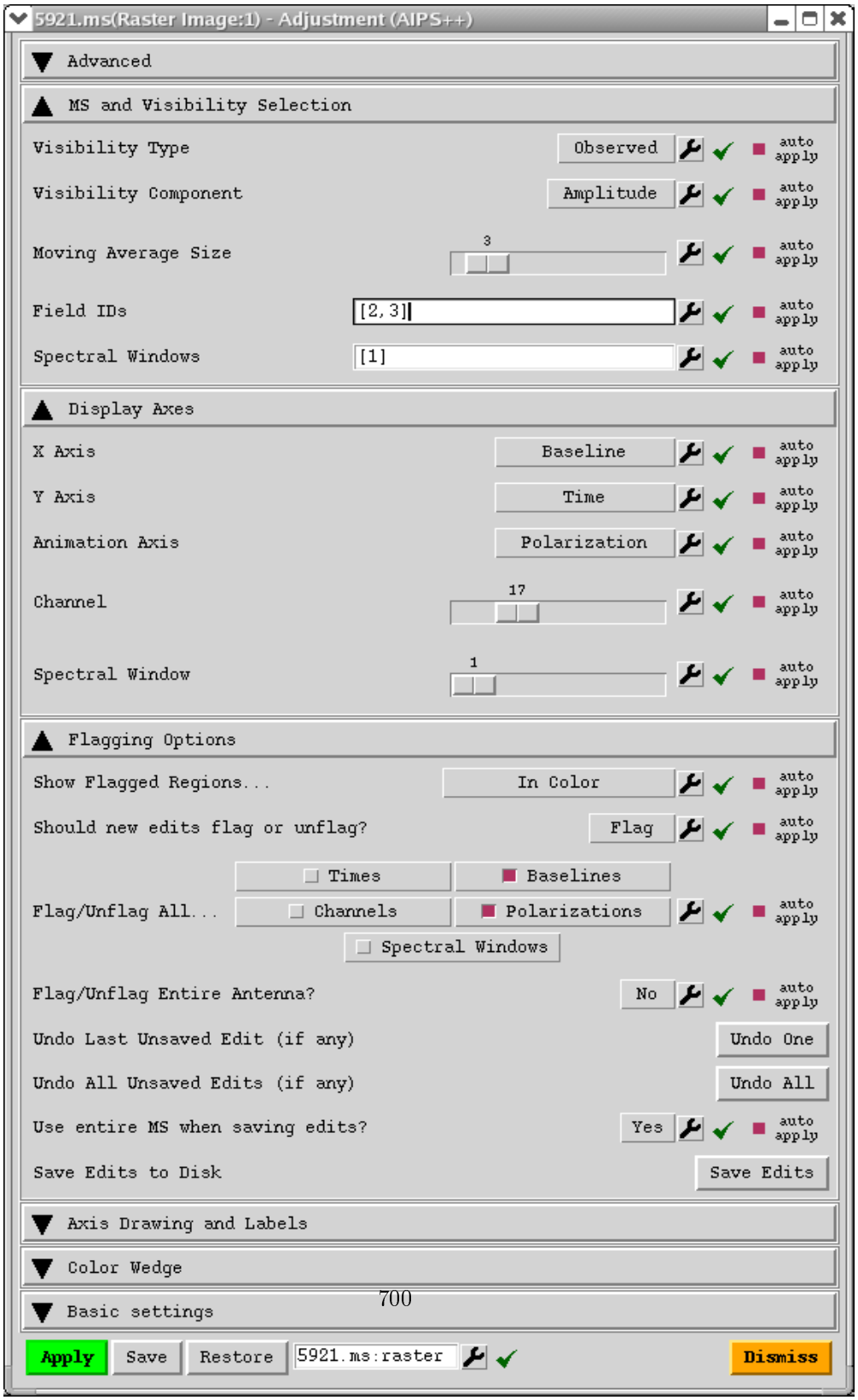

Figure 1.4: Options Adjustment Panel for Measurement Sets

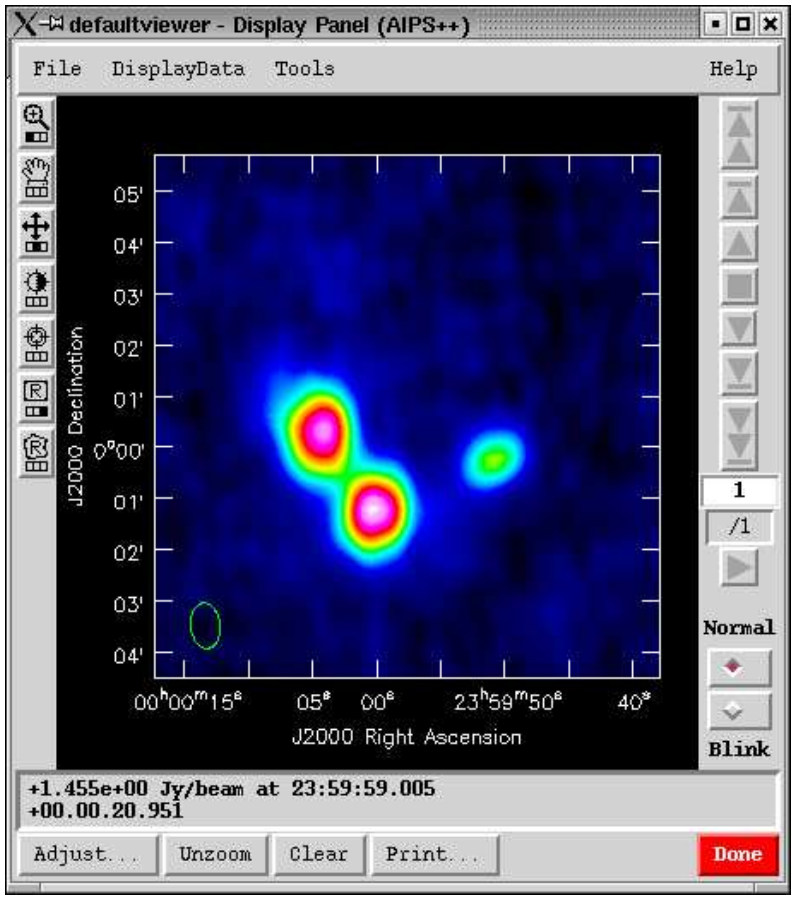

Figure 1.5: The Viewerdisplaypanel `tool` graphical user interface

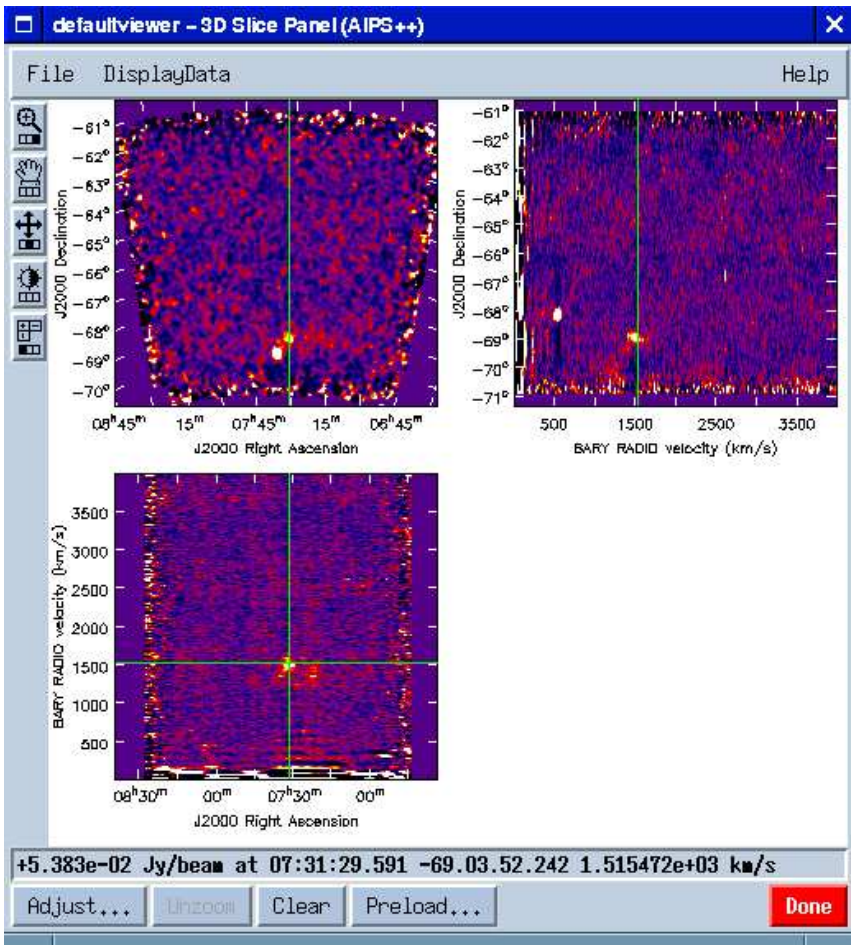

Figure 1.6: The Viewerslicepanel `tool` graphical user interface

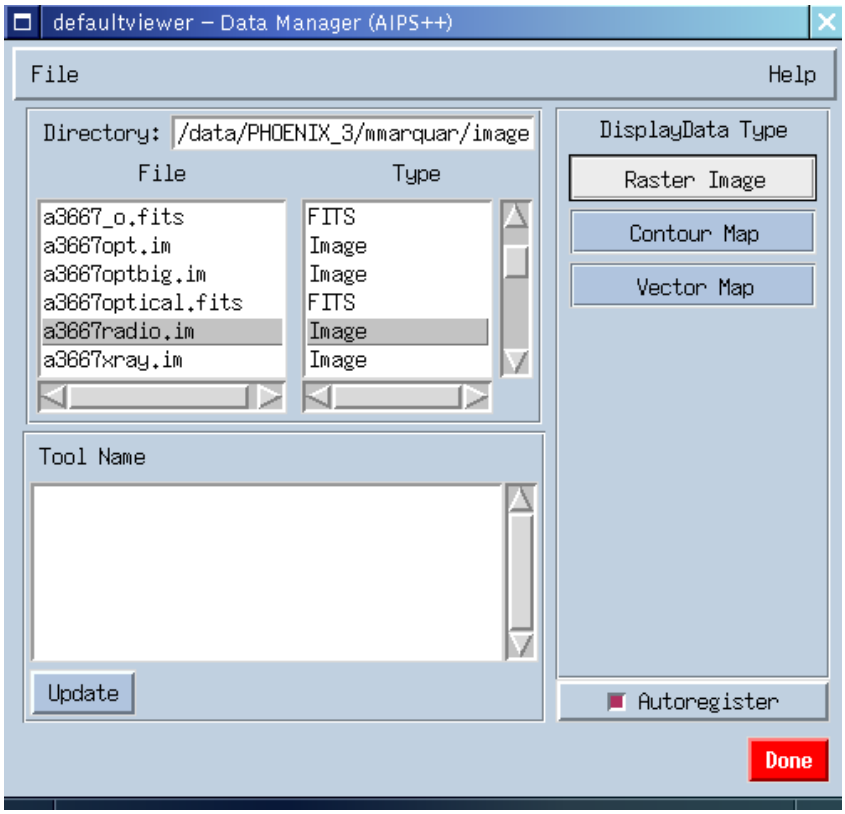

Figure 1.7: The Viewer Data Manager

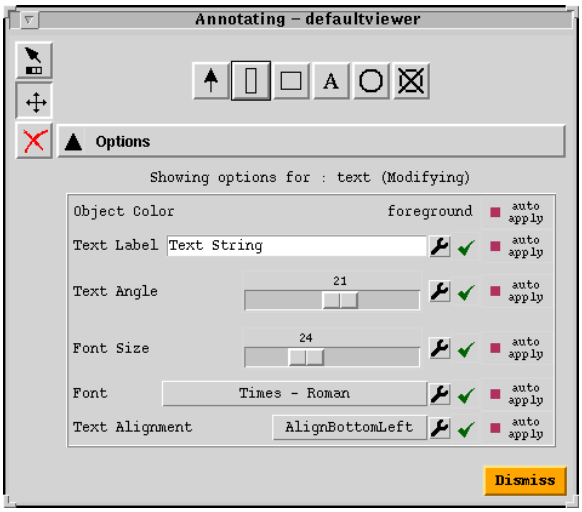

Figure 1.8: The Vieweranno tations `tool`tool graphical user interface

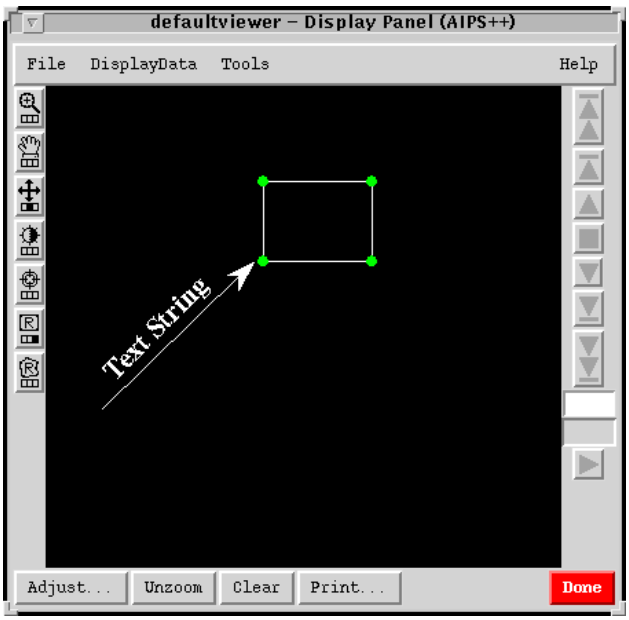

Figure 1.9: Manipulating annotations objects

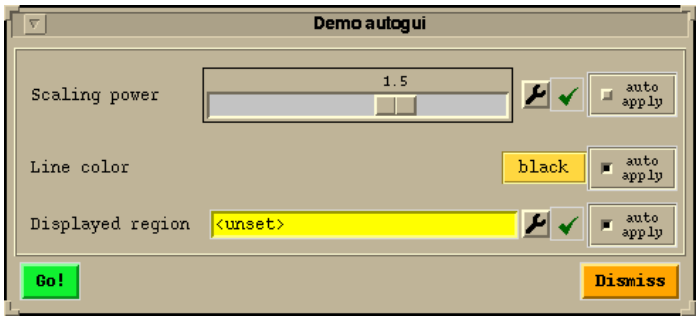

Figure 1.10: A simple autogui `tool`

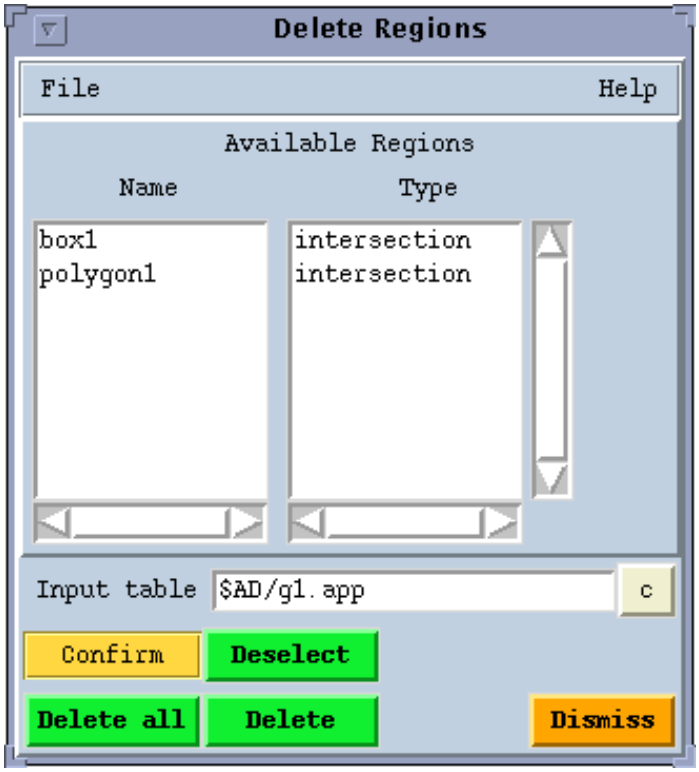

Figure 1.11: The deleteregions GUI

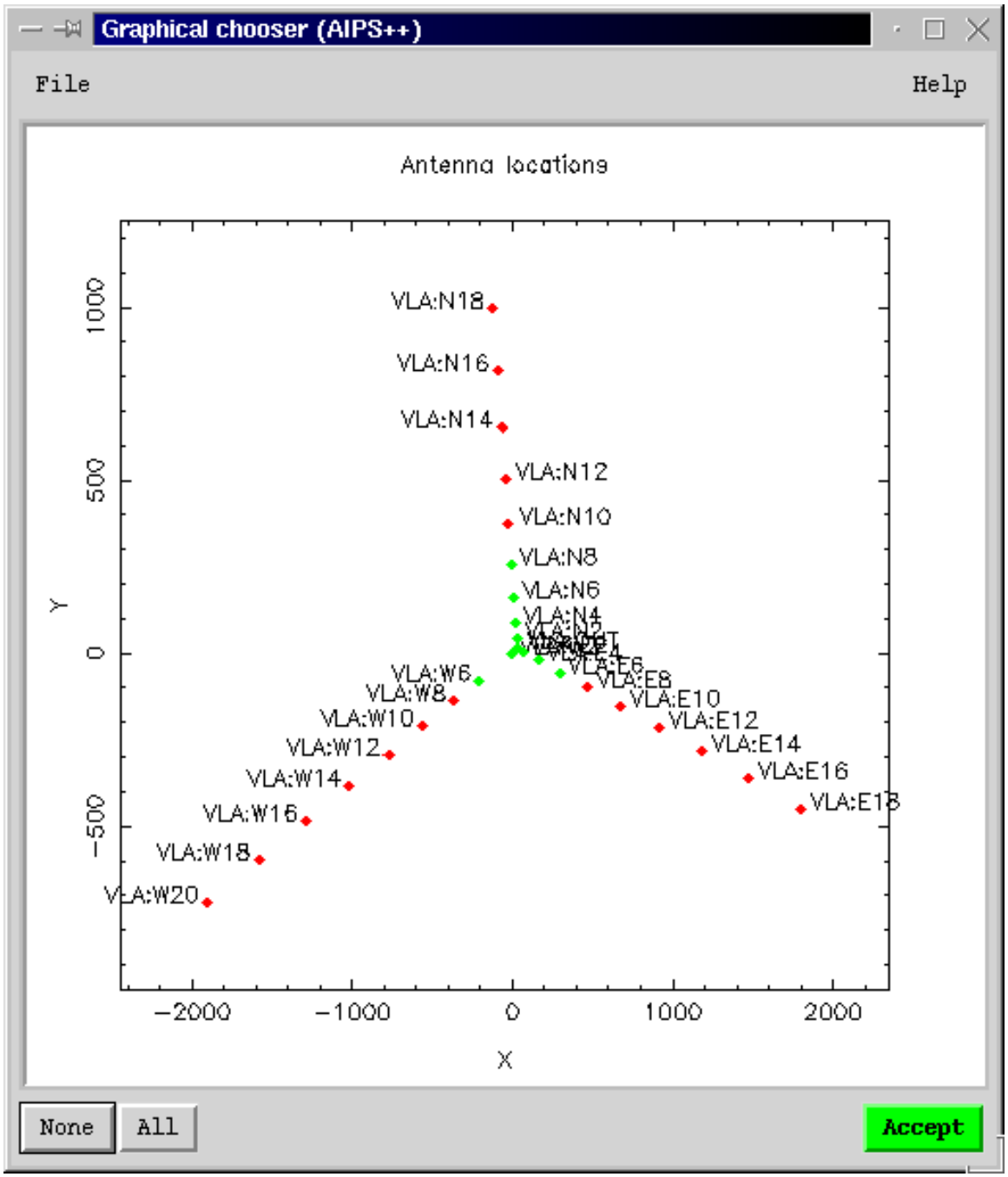

Figure 1.12: An example of a graphical chooser. Selected items are in green.

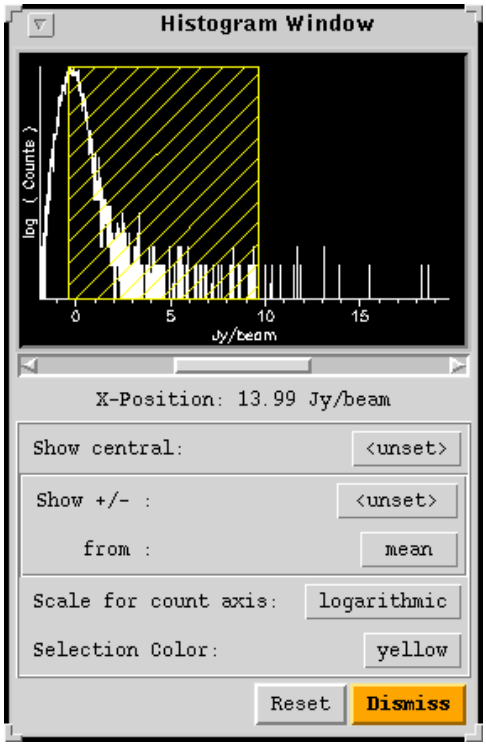

Figure 1.13: The histogramgui Window

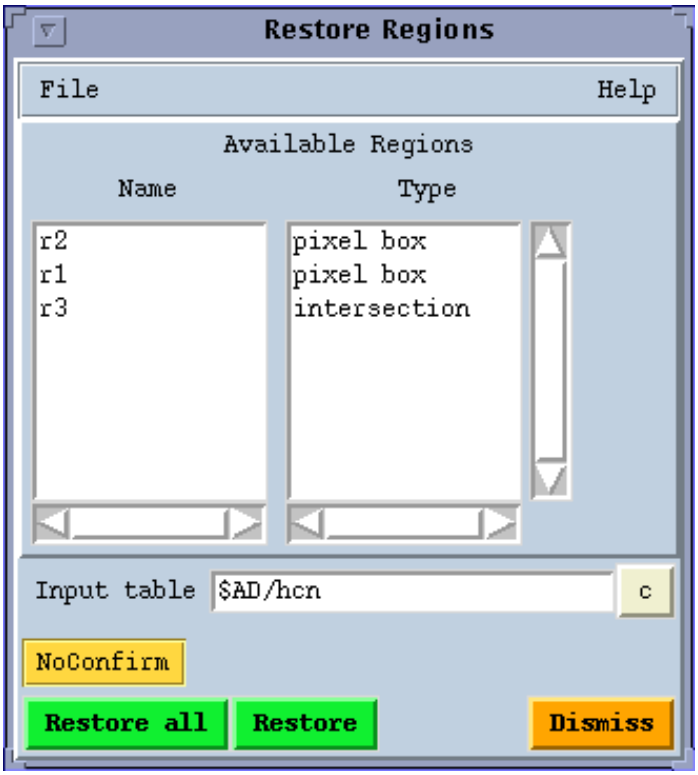

Figure 1.14: The restoreregions GUI

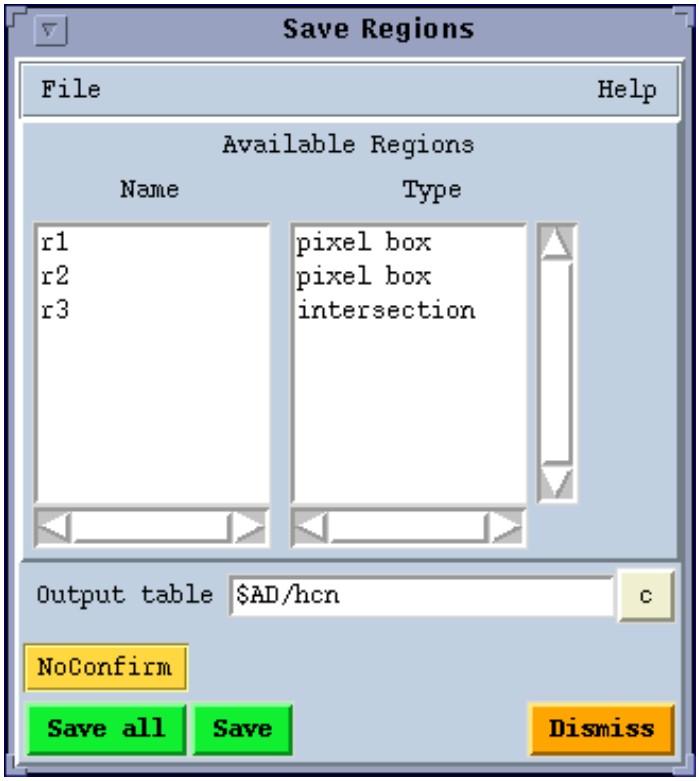

Figure 1.15: The saveregions GUI

Index

actionoptionmenu, 320, 321, 671
activate, 48
activatecallback, 119
add, 50, 506, 628
addactionhandler, 27
addactions, 25
addautogui, 493
addgui, 259
addhelp, 24
addlockedtocurrent, 307
addmenus, 24
addonreturn, 373
addpopuphelp, 67, 692
addredrawfunction, 103, 130
addtool, 138
addtouserchoicelist, 213
addunit, 311
adjust, 260, 470
agent, 370
alldisplaydatas, 167
alldisplaypanels, 168
animator, 261
annotationdd, 262
annotations, 296
annotator, 268
append, 33, 43
apply, 56
array, 434
arrowbackground, 389
arrowbutton, 389
arrowforeground, 390
autocli, 323
autogui, 324, 326, 331
autoguitest, 338
autoinserorder, 380
autoregister, 262
available, 629
availableshape, 304

background, 611
bar chart, 68
barchart, 68, 69
bind, 381, 612
blink, 342
blinklabel, 339, 340, 677
boolean, 436
booleanarray, 435
borderwidth, 375, 613
brightness, 200
busy, 22, 140, 689
button, 659
buttonbar, 333

canclearpopup, 374
canplay, 94, 121
canvas, 185, 192, 245, 286, 290, 665
canvasmanager, 263
canvasprintmanager, 263
catalog, 295
chart, 70
check, 439
checkmenu, 343, 345, 672
choice, 6, 437
choicewindow, 74
chooser, 13
classtype, 214
cleanup, 28
clear, 33, 40, 43, 52, 102, 127, 485, 515, 614
clearclipboard, 53
cleardrawlist, 53
colormap, 166, 200, 205
colormapmanager, 169
colormapnames, 204

colors, 185, 290
combobox, 363, 365, 676
commandgui, 138
compact, 56
contents, 40
contour, 166, 211
contrast, 200
copy, 55
createshape, 303
cube, 286
currentvalue, 651
curs, 130
cursor, 129, 381
cut, 49, 54

datachooser, 11
datamanager, 295
ddd, 215
ddproxy, 215
deactivate, 49
deactivatecallback, 118
defaultactions, 23
defaulthelp, 23
defaultmenus, 23
Delete, 507
delete, 34, 36, 38, 41, 44, 376, 485, 614, 630
deleteall, 169, 631
deletedata, 170
deleteregions, 391, 392, 681
deleteshape, 301
demo, 141
dialogbox, 398, 672
dialogframe, 632
direction, 449
disable, 171, 294, 476, 498, 509, 550, 554, 577, 644
disableallbutton, 544
disablebutton, 559
disablecontrols, 274
disabled, 362, 376, 416, 540, 544, 558, 601
dismiss, 22, 293, 299, 324, 476, 502, 642
display, 166, 309
displaydata, 166, 211
displaylist, 49, 50, 91, 115
displaylisttest, 62
displaypanel, 245
displaytype, 216
done, 62, 74, 98, 131, 141, 171, 188, 196, 203, 216, 265, 292, 299, 312, 336, 343, 394, 417, 460, 469, 476, 481, 510, 515, 541, 545, 555, 560, 568, 579, 584, 590, 596, 602, 610, 633, 642
donechoice, 7
doppler, 446
down, 576
dying, 265

editgui, 138
emittoolkitlist, 172
enable, 173, 295, 477, 498, 510, 551, 555, 578, 643
enableallbutton, 545
enablebutton, 560
enablecontrols, 275
entry, 663
entrybackground, 388
entrydisabled, 386
entryfont, 387
entryforeground, 388
entryjustify, 388
entryrelief, 387
entrywidth, 386
epoch, 448
evalwindow, 38, 39
exitchoice, 8
exportselection, 375, 615
extend, 347, 404, 520
extendoptionmenu, 399, 401, 670

fiddle, 200
fiddling, 200
file, 440
file browser, 295
filechooser, 9
filename, 217
fillgui, 337, 493
findlabel, 411, 532

findname, 357, 533
findvalue, 358, 534
font, 616
fontchooser, 677
fontchooserwidget, 417, 418
foreground, 616
frame, 574, 658
frequency, 444
fromascii, 312
fromcomponentlist, 313
fromforeign, 483
fromimage, 425
fromms, 426
front, 634

gchooser, 419, 421
gchoosertest, 423
get, 51, 326, 334, 371, 486, 617
getagent, 88
getalloptions, 305
getbrightness, 206
getcallbacks, 119
getcontrast, 207
getentry, 373
geteventmask, 412, 535
getindex, 408, 526
getinvertflags, 208
getlabel, 352, 410, 528
getlabels, 410, 528
getname, 529
getnames, 353, 530
getoffstates, 357
getonstates, 356
getoptions, 189, 217, 272
getpreviousindex, 409, 527
getradiovalue, 546
getshapeoptions, 306
getslidervalues, 546
getstate, 354
getstates, 355
getvalue, 531, 556, 561
getvalues, 354, 499, 532
getworkframe, 25
gopher, 423, 424
gui, 6, 8, 9, 28, 31, 46, 49, 58, 68, 73, 142, 174, 188, 196, 203, 294, 394, 473, 568, 584
guientry, 427, 431, 673, 684
guientrytest, 465
guiframework, 14, 20
guiutils, 5
guiutils.addpopuphelp, 67
guiutils.barchart, 68, 69
guiutils.barchart.chart, 70
guiutils.barchart.done, 74
guiutils.barchart.gui, 73
guiutils.barchart.large, 73
guiutils.barchart.nogui, 72
guiutils.barchart.remove, 71
guiutils.barchart.values, 72
guiutils.choice, 6
guiutils.choicewindow, 74
guiutils.chooser, 13
guiutils.datachooser, 11
guiutils.displaylist, 49, 50
guiutils.displaylist.add, 50
guiutils.displaylist.apply, 56
guiutils.displaylist.clear, 52
guiutils.displaylist.clearclipboard, 53
guiutils.displaylist.cleardrawlist, 53
guiutils.displaylist.compact, 56
guiutils.displaylist.copy, 55
guiutils.displaylist.cut, 54
guiutils.displaylist.done, 62
guiutils.displaylist.get, 51
guiutils.displaylist.gui, 58
guiutils.displaylist.lastchange, 57
guiutils.displaylist.nclipboard, 54
guiutils.displaylist.ndrawlist, 53
guiutils.displaylist.nogui, 58
guiutils.displaylist.paste, 56
guiutils.displaylist.restore, 59
guiutils.displaylist.save, 59
guiutils.displaylist.set, 52
guiutils.displaylist.setchangecallback, 61
guiutils.displaylist.setrefreshcallback, 61
guiutils.displaylist.setselectcallback, 60
guiutils.displaylisttest, 62

guiutils.donechoice, 7
guiutils.evalwindow, 38, 39
guiutils.evalwindow.clear, 40
guiutils.evalwindow.contents, 40
guiutils.evalwindow.delete, 41
guiutils.exitchoice, 8
guiutils.filechooser, 9
guiutils.gui, 28, 31
guiutils.gui.inputform, 29
guiutils.gui.tabform, 30
guiutils.guiframework, 14, 20
guiutils.guiframework.addactionhandler, 27
guiutils.guiframework.addactions, 25
guiutils.guiframework.addhelp, 24
guiutils.guiframework.addmenus, 24
guiutils.guiframework.busy, 22
guiutils.guiframework.cleanup, 28
guiutils.guiframework.defaultactions, 23
guiutils.guiframework.defaulthelp, 23
guiutils.guiframework.defaultmenus, 23
guiutils.guiframework.dismiss, 22
guiutils.guiframework.getworkframe, 25
guiutils.guiframework.updatestatus, 26
guiutils.guiframework.updatetitle, 26
guiutils.help, 63, 65–67
guiutils.information dialog, 74, 76
guiutils.infowindow, 75
guiutils.messagebox, 34, 35
guiutils.messagebox.delete, 36
guiutils.messagebox.text, 35
guiutils.popup help, 63, 65–67
guiutils.popuphelp, 63
guiutils.popupmenu, 66
guiutils.popupremove, 64
guiutils.progress, 46
guiutils.progress.activate, 48
guiutils.progress.deactivate, 49
guiutils.progress.update, 48
guiutils.single_entry_dialog_box, 36, 37
guiutils.single_entry_dialog_box.delete, 38
guiutils.status_line, 31, 32
guiutils.status_line.append, 33
guiutils.status_line.clear, 33
guiutils.status_line.delete, 34
guiutils.status_line.show, 32
guiutils.tablechooser, 12
guiutils.tool tips, 63, 65–67
guiutils.tracewindow, 41, 42
guiutils.tracewindow.append, 43
guiutils.tracewindow.clear, 43
guiutils.tracewindow.delete, 44
guiutils.tracewindow.view_named_function, 44
guiutils:bar chart, 68
guiutils:barchart, 68
guiutils:cut, 49
guiutils:gui, 6, 8, 9, 46, 49, 68
guiutils:paste, 49

hasbeam, 220
hasitem, 484
have_gui, 667
height, 618
help, 63, 65–67, 326
helpmenu, 466, 467, 687
hidden, 579
histogram, 470
histogramgui, 470, 472
hold, 173, 273
holdsdata, 269
hscrollbar, 379, 481, 591, 597, 618

information dialog, 74, 76
infowindow, 75
inputform, 29
insert, 377, 487, 619
insertentry, 378
is_viewer, 317
is_widgetserver, 693, 694
isenabled, 362, 540
ismapped, 292
isregistered, 281
itemnames, 484

kill, 552

label, 662
labelanchor, 385
labelbackground, 385
labelfont, 383
labelforeground, 384
labeljustify, 384
labelrelief, 384
labeltext, 382
labelwidth, 383
large, 73
lastchange, 57, 103, 129
leadbox, 620
leftframe, 267
list, 458, 635
listbox, 477, 587, 590, 593, 606, 610, 664
listboxcontainer, 477, 479
listboxheight, 390
loaddata, 175
loop, 325
lowerframe, 266

map, 290, 335
margins, 185, 290
maskline, 100, 113
measurecodes, 441
menuframes, 503, 505, 688
mergecolumns, 314
message, 127, 664
messagebox, 34, 35
messageline, 512, 513, 674
minmaxhist, 463
mode, 620
modifygui, 338
multiautogui, 489, 491
multiautoguitest, 494
multiscale, 495, 496, 678

name, 221, 315
nclipboard, 54
ndrawlist, 53
nearest, 621
newcanvasmanagergui, 275
newcanvasprintmanagergui, 276
newcmap, 269
newcolormapmanagergui, 176
newdata, 474
newdatamanager, 177
newdisplaydatagui, 221
newdisplaypanel, 178
newselection, 652
newshape, 300
nmaps, 205
nogui, 58, 72
notbusy, 690

optionmenu, 516, 518, 669

palette, 200
paper, 185, 290
paste, 49, 56
pgplot, 666
pgplotmanager, 77, 83
pgplotps, 86
pgplotter, 132, 136
pgplottertest, 158
pgplotwidget, 104, 110
pixelheight, 371
pixelrange, 542
pixeltype, 223
pixelwidth, 370
play, 95, 122
plot, 77, 105, 132
plotfile, 96, 124
plotter, 76
plotter.pgplotmanager, 77, 83
plotter.pgplotmanager.addredrawfunction, 103
plotter.pgplotmanager.canplay, 94
plotter.pgplotmanager.clear, 102
plotter.pgplotmanager.displaylist, 91
plotter.pgplotmanager.done, 98
plotter.pgplotmanager.getagent, 88
plotter.pgplotmanager.lastchange, 103
plotter.pgplotmanager.maskline, 100
plotter.pgplotmanager.play, 95
plotter.pgplotmanager.plotfile, 96
plotter.pgplotmanager.plotxy, 99
plotter.pgplotmanager.record, 93
plotter.pgplotmanager.recording, 94
plotter.pgplotmanager.refresh, 97

plotter.pgplotmanager.resetplotnumber, 91
plotter.pgplotmanager.restore, 98
plotter.pgplotmanager.setagent, 89
plotter.pgplotmanager.settings, 102
plotter.pgplotps, 86
plotter.pgplotter, 132, 136
plotter.pgplotter.addtool, 138
plotter.pgplotter.busy, 140
plotter.pgplotter.commandgui, 138
plotter.pgplotter.demo, 141
plotter.pgplotter.done, 141
plotter.pgplotter.editgui, 138
plotter.pgplotter.gui, 142
plotter.pgplotter.plotxy1, 145
plotter.pgplotter.plotxy2, 148
plotter.pgplotter.ploty, 144
plotter.pgplotter.ploty2, 146
plotter.pgplotter.psprint, 147
plotter.pgplotter.psprinttofile, 148
plotter.pgplotter.screen, 142
plotter.pgplotter.setcolor, 156
plotter.pgplotter.setplottitle, 156
plotter.pgplotter.setxaxisgrid, 154
plotter.pgplotter.setxaxislabel, 157
plotter.pgplotter.setxscale, 152
plotter.pgplotter.sety2axisgrid, 155
plotter.pgplotter.sety2scale, 153
plotter.pgplotter.setyaxisgrid, 155
plotter.pgplotter.setyaxislabel, 157
plotter.pgplotter.setyscale, 153
plotter.pgplotter.timey, 150
plotter.pgplotter.timey2, 151
plotter.pgplotter.title, 143
plotter.pgplotter.tool, 140
plotter.pgplotter.type, 152
plotter.pgplotter.userframe, 143
plotter.pgplottertest, 158
plotter.pgplotwidget, 104, 110
plotter.pgplotwidget.activatecallback, 119
plotter.pgplotwidget.addredrawfunction, 130
plotter.pgplotwidget.canplay, 121
plotter.pgplotwidget.clear, 127
plotter.pgplotwidget.curs, 130
plotter.pgplotwidget.cursor, 129
plotter.pgplotwidget.deactivatecallback, 118
plotter.pgplotwidget.displaylist, 115
plotter.pgplotwidget.done, 131
plotter.pgplotwidget.getcallbacks, 119
plotter.pgplotwidget.lastchange, 129
plotter.pgplotwidget.maskline, 113
plotter.pgplotwidget.message, 127
plotter.pgplotwidget.play, 122
plotter.pgplotwidget.plotfile, 124
plotter.pgplotwidget.plotxy, 111
plotter.pgplotwidget.postscript, 123
plotter.pgplotwidget.record, 120
plotter.pgplotwidget.recording, 121
plotter.pgplotwidget.refresh, 125
plotter.pgplotwidget.resetplotnumber, 126
plotter.pgplotwidget.restore, 124
plotter.pgplotwidget.setcallback, 116
plotter.pgplotwidget.settings, 128
plotter.pgplotwidget.shortnametofullname, 115
plotter.pgplotwidget.size, 126
plotter.plot, 77, 105, 132
plotter.plotting, 77, 105, 132
plotter.postscript, 77, 105, 132
plotter.TV, 77, 105, 132
plotting, 77, 105, 132
plotxy, 99, 111
plotxy1, 145
plotxy2, 148
ploty, 144
ploty2, 146
popup help, 63, 65–67
popupcursor, 382
popuphelp, 63, 691
popupmenu, 66, 692
popupremove, 64, 692
popupselectmenu, 548, 549, 679
position, 447
post, 515
postnoforward, 516
postscript, 77, 105, 123, 132
preload, 289

print, 192, 300
printing, 192
profilewidthentry, 552, 553
progress, 46
psprint, 147
psprinttofile, 148

quantity, 450

radialvelocity, 445
radiobuttons, 556, 557
range, 452
raster, 166, 211
record, 93, 120, 453
recordbrowser, 679
recordbrowserwidget, 563
recording, 94, 121
refresh, 97, 125, 395, 569
region, 454
regions, 569
register, 280
registercolormap, 279
registerednames, 282
registrationflags, 282
release, 181, 274
relief, 622
remove, 71
removelockedfromcurrent, 308
renamecolumn, 315
replace, 348, 405, 521, 602, 636
reset, 210, 360, 468, 561
resetall, 210
resetplotnumber, 91, 126
resources, 686
restore, 59, 98, 124
restoreoptions, 191, 223
restoreregions, 565, 566, 680
rightframe, 267
rollup, 571, 572, 668

save, 59
saveoptions, 191, 224
saveregions, 581, 582, 681
scalar, 433
scalarmeasure, 443
scale, 660
screen, 142
scrollbar, 661
scrolllistbox, 477, 587, 588, 606, 674
scrolltext, 593, 594
see, 488, 622
seeoninsert, 483, 592, 623
select, 372, 488, 623
selectablelist, 598, 599, 682
selectindex, 349, 406, 523
selection, 372, 489, 624
selectlabel, 407, 523
selectname, 350, 524
selectvalue, 351, 525
set, 52
setagent, 89
setalloptions, 305
setbackground, 361, 415, 538, 603
setbrightness, 206
setcallback, 116
setcallbacks, 325
setchangecallback, 61
setcolor, 156
setcontrast, 208
seteventmask, 413, 536
setforeground, 361, 414, 538, 604
setinvertflags, 209
setkey, 304
setlabel, 359, 537, 604
setmode, 684
setoptions, 190, 225, 273, 289
setplottitle, 156
setpopuphelp, 580
setradiovalue, 548
setrange, 500, 547
setrefreshcallback, 61
setregions, 395, 585
setresolution, 501
setselectcallback, 60
setselection, 475
setshapeoptions, 307
setstate, 562
setstats, 474
settable, 396, 570, 586
settings, 102, 128
setvalues, 501
setwidth, 415, 539, 605

setxaxisgrid, 154
setxaxislabel, 157
setxscale, 152
sety2axisgrid, 155
sety2scale, 153
setyaxisgrid, 155
setyaxislabel, 157
setyscale, 153
setzoom, 284
shortnametofullname, 115
show, 32, 492
shownone, 492
single_entry_dialog_box, 36, 37
size, 126
skycat, 309
skycatalog, 309, 310
slicedp, 286
state, 511
status, 270
status_line, 31, 32
string, 455
switch, 508, 575
synclistboxes, 606, 608, 675

tabdialog, 626, 627, 683
tabdialogtest, 638
tabform, 30
table, 309, 316, 645
tablechooser, 12
tapedeck, 639, 682
TaQL, 645
taqlwidget, 645, 646
testpattern, 277
text, 35, 596, 661
timey, 150
timey2, 151
title, 143, 182
Tk, 320, 339, 344, 363, 391, 398, 400, 417, 419, 423, 428, 466, 477, 495, 503, 512, 516, 542, 549, 553, 557, 563, 565, 571, 581, 587, 593, 598, 606, 626, 639
tk_hold, 667
tk_iconpath, 668
tk_release, 667
tool, 140
tool tips, 63, 65–67
toolkit, 182
tracewindow, 41, 42
TV, 77, 105, 132
twodactionoptionmenu, 649
twoentry, 461
type, 152, 183, 187, 195, 201, 226, 278, 316, 461, 693

unmap, 291
unregister, 281
unregisterall, 283
unregistercolormap, 279
unsetmode, 685
untyped, 456
unzoom, 284
up, 577
update, 48
updatedatasets, 183
updatestatus, 26
updatetitle, 26
upperframe, 266
userframe, 143

values, 72
view, 166
view_named_function, 44
viewer, 160, 166, 167, 187, 195, 202, 227, 264, 309
viewer.annotations, 296
viewer.brightness, 200
viewer.canvas, 185, 192, 290
viewer.colormap, 166, 200
viewer.colors, 185, 290
viewer.contour, 166, 211
viewer.contrast, 200
viewer.datamanager, 295
viewer.display, 166, 309
viewer.displaydata, 166, 211
viewer.displaypanel, 245
viewer.fiddle, 200
viewer.fiddling, 200
viewer.is_viewer, 317
viewer.margins, 185, 290
viewer.palette, 200

viewer.paper, 185, 290
viewer.print, 192
viewer.printing, 192
viewer.raster, 166, 211
viewer.skycat, 309
viewer.skycatalog, 309, 310
viewer.skycatalog.addunit, 311
viewer.skycatalog.done, 312
viewer.skycatalog.fromascii, 312
viewer.skycatalog.fromcomponentlist, 313
viewer.skycatalog.mergecolumns, 314
viewer.skycatalog.name, 315
viewer.skycatalog.renamecolumn, 315
viewer.skycatalog.table, 316
viewer.skycatalog.type, 316
viewer.slicedp, 286
viewer.table, 309
viewer.view, 166
viewer.viewer, 166, 167, 309
viewer.viewer.alldisplaydatas, 167
viewer.viewer.alldisplaypanels, 168
viewer.viewer.colormapmanager, 169
viewer.viewer.deleteall, 169
viewer.viewer.deletedata, 170
viewer.viewer.disable, 171
viewer.viewer.done, 171
viewer.viewer.emittoolkitlist, 172
viewer.viewer.enable, 173
viewer.viewer.gui, 174
viewer.viewer.hold, 173
viewer.viewer.loaddata, 175
viewer.viewer.newcolormapmanagergui, 176
viewer.viewer.newdatamanager, 177
viewer.viewer.newdisplaypanel, 178
viewer.viewer.release, 181
viewer.viewer.title, 182
viewer.viewer.toolkit, 182
viewer.viewer.type, 183
viewer.viewer.updatedatasets, 183
viewer.viewer.widgetset, 184
viewer.viewerannotations, 296, 298
viewer.viewerannotations.addlockedtocurrent, 307
viewer.viewerannotations.availableshape, 304
viewer.viewerannotations.createshape, 303
viewer.viewerannotations.deleteshape, 301
viewer.viewerannotations.dismiss, 299
viewer.viewerannotations.done, 299
viewer.viewerannotations.getalloptions, 305
viewer.viewerannotations.getshapeoptions, 306
viewer.viewerannotations.newshape, 300
viewer.viewerannotations.print, 300
viewer.viewerannotations.removelockedfromcurrent, 308
viewer.viewerannotations.setalloptions, 305
viewer.viewerannotations.setkey, 304
viewer.viewerannotations.setshapeoptions, 307
viewer.viewerannotations.whichshape, 302
viewer.viewercanvasmanager, 184, 186
viewer.viewercanvasmanager.done, 188
viewer.viewercanvasmanager.getoptions, 189
viewer.viewercanvasmanager.gui, 188
viewer.viewercanvasmanager.restoreoptions, 191
viewer.viewercanvasmanager.saveoptions, 191
viewer.viewercanvasmanager.setoptions, 190
viewer.viewercanvasmanager.type, 187
viewer.viewercanvasmanager.viewer, 187
viewer.viewercanvasprintmanager, 192, 194
viewer.viewercanvasprintmanager.done, 196

viewer.viewercanvasprintmanager.gui, 196
viewer.viewercanvasprintmanager.type, 195
viewer.viewercanvasprintmanager.viewer, 195
viewer.viewercanvasprintmanager.writeps, 198
viewer.viewercanvasprintmanager.writexpm, 199
viewer.viewercolormapmanager, 199, 201
viewer.viewercolormapmanager.colormap, 205
viewer.viewercolormapmanager.colormapnames, 204
viewer.viewercolormapmanager.done, 203
viewer.viewercolormapmanager.getbrightness, 206
viewer.viewercolormapmanager.getcontrast, 207
viewer.viewercolormapmanager.getinvertflags, 208
viewer.viewercolormapmanager.gui, 203
viewer.viewercolormapmanager.nmaps, 205
viewer.viewercolormapmanager.reset, 210
viewer.viewercolormapmanager.resetall, 210
viewer.viewercolormapmanager.setbrightness, 206
viewer.viewercolormapmanager.setcontrast, 208
viewer.viewercolormapmanager.setinvertflags, 209
viewer.viewercolormapmanager.type, 201
viewer.viewercolormapmanager.viewer, 202
viewer.viewerdatamanager, 295
viewer.viewerdisplaydata, 211, 212
viewer.viewerdisplaydata.addtouserchoicelist, 213
viewer.viewerdisplaydata.classtype, 214
viewer.viewerdisplaydata.ddd, 215
viewer.viewerdisplaydata.ddproxy, 215
viewer.viewerdisplaydata.displaytype, 216
viewer.viewerdisplaydata.done, 216
viewer.viewerdisplaydata.filename, 217
viewer.viewerdisplaydata.getoptions, 217
viewer.viewerdisplaydata.hasbeam, 220
viewer.viewerdisplaydata.name, 221
viewer.viewerdisplaydata.newdisplaydatagui, 221
viewer.viewerdisplaydata.pixeltype, 223
viewer.viewerdisplaydata.restoreoptions, 223
viewer.viewerdisplaydata.saveoptions, 224
viewer.viewerdisplaydata.setoptions, 225
viewer.viewerdisplaydata.type, 226
viewer.viewerdisplaydata.viewer, 227
viewer.viewerdisplaydata.wedgedd, 226
viewer.viewerdisplaydata.zlength, 227
viewer.viewerdisplaypanel, 245, 257
viewer.viewerdisplaypanel.addgui, 259
viewer.viewerdisplaypanel.adjust, 260
viewer.viewerdisplaypanel.animator, 261
viewer.viewerdisplaypanel.annotationdd, 262
viewer.viewerdisplaypanel.annotator, 268
viewer.viewerdisplaypanel.autoregister, 262
viewer.viewerdisplaypanel.canvasmanager, 263
viewer.viewerdisplaypanel.canvasprintmanager, 263

viewer.viewerdisplaypanel.disablecontrols, 274
viewer.viewerdisplaypanel.done, 265
viewer.viewerdisplaypanel.dying, 265
viewer.viewerdisplaypanel.enablecontrols, 275
viewer.viewerdisplaypanel.getoptions, 272
viewer.viewerdisplaypanel.hold, 273
viewer.viewerdisplaypanel.holdsdata, 269
viewer.viewerdisplaypanel.isregistered, 281
viewer.viewerdisplaypanel.leftframe, 267
viewer.viewerdisplaypanel.lowerframe, 266
viewer.viewerdisplaypanel.newcanvasmanagergui, 275
viewer.viewerdisplaypanel.newcanvasprintmanagergui, 276
viewer.viewerdisplaypanel.newcmap, 269
viewer.viewerdisplaypanel.register, 280
viewer.viewerdisplaypanel.registercolormap, 279
viewer.viewerdisplaypanel.registerednames, 282
viewer.viewerdisplaypanel.registrationflags, 282
viewer.viewerdisplaypanel.release, 274
viewer.viewerdisplaypanel.rightframe, 267
viewer.viewerdisplaypanel.setoptions, 273
viewer.viewerdisplaypanel.setzoom, 284
viewer.viewerdisplaypanel.status, 270
viewer.viewerdisplaypanel.testpattern, 277
viewer.viewerdisplaypanel.type, 278
viewer.viewerdisplaypanel.unregister, 281
viewer.viewerdisplaypanel.unregisterall, 283
viewer.viewerdisplaypanel.unregistercolormap, 279
viewer.viewerdisplaypanel.unzoom, 284
viewer.viewerdisplaypanel.upperframe, 266
viewer.viewerdisplaypanel.viewer, 264
viewer.viewerdisplaypanel.writexpm, 268
viewer.viewerdisplaypanel.zoom, 285
viewer.viewerslicedp, 286, 287
viewer.viewerslicedp.preload, 289
viewer.viewerslicedp.setoptions, 289
viewer.viewerstandardgui, 290
viewer.viewerstandardgui.disable, 294
viewer.viewerstandardgui.dismiss, 293
viewer.viewerstandardgui.done, 292
viewer.viewerstandardgui.enable, 295
viewer.viewerstandardgui.gui, 294
viewer.viewerstandardgui.ismapped, 292
viewer.viewerstandardgui.map, 290
viewer.viewerstandardgui.unmap, 291
viewer.visualisation, 166
viewer:canvas, 245, 286
viewer:catalog, 295
viewer:cube, 286
viewer:file browser, 295
viewerannotations, 296, 298
viewercanvasmanager, 184, 186
viewercanvasprintmanager, 192, 194
viewercolormapmanager, 199, 201
viewerdatamanager, 295
viewerdisplaydata, 211, 212
viewerdisplaypanel, 245, 257
viewerslicedp, 286, 287
viewerstandardgui, 290
visualisation, 166
vscrollbar, 379, 482, 591, 597, 625

wedgedd, 226
which, 637
whichshape, 302
widget, 645

widgets, 319, 320, 323, 327, 339, 344, 363, 391, 398, 400, 417, 419, 423, 428, 466, 477, 489, 495, 503, 512, 516, 542, 549, 553, 557, 563, 565, 571, 581, 587, 593, 598, 606, 626, 639, 649, 652
widgets.actionoptionmenu, 320, 321
widgets.adjust, 470
widgets.autocli, 323
widgets.autocli.dismiss, 324
widgets.autocli.get, 326
widgets.autocli.help, 326
widgets.autocli.loop, 325
widgets.autocli.setcallbacks, 325
widgets.autogui, 324, 326, 331
widgets.autogui.buttonbar, 333
widgets.autogui.done, 336
widgets.autogui.fillgui, 337
widgets.autogui.get, 334
widgets.autogui.map, 335
widgets.autogui.modifygui, 338
widgets.autoguitest, 338
widgets.blinklabel, 339, 340
widgets.blinklabel.blink, 342
widgets.blinklabel.done, 343
widgets.checkmenu, 343, 345
widgets.checkmenu.disabled, 362
widgets.checkmenu.extend, 347
widgets.checkmenu.findname, 357
widgets.checkmenu.findvalue, 358
widgets.checkmenu.getlabel, 352
widgets.checkmenu.getnames, 353
widgets.checkmenu.getoffstates, 357
widgets.checkmenu.getonstates, 356
widgets.checkmenu.getstate, 354
widgets.checkmenu.getstates, 355
widgets.checkmenu.getvalues, 354
widgets.checkmenu.isenabled, 362
widgets.checkmenu.replace, 348
widgets.checkmenu.reset, 360
widgets.checkmenu.selectindex, 349
widgets.checkmenu.selectname, 350
widgets.checkmenu.selectvalue, 351
widgets.checkmenu.setbackground, 361
widgets.checkmenu.setforeground, 361
widgets.checkmenu.setlabel, 359
widgets.combobox, 363, 365
widgets.combobox.addonreturn, 373
widgets.combobox.agent, 370
widgets.combobox.arrowbackground, 389
widgets.combobox.arrowbutton, 389
widgets.combobox.arrowforeground, 390
widgets.combobox.autoinserorder, 380
widgets.combobox.bind, 381
widgets.combobox.borderwidth, 375
widgets.combobox.canclearpopup, 374
widgets.combobox.cursor, 381
widgets.combobox.delete, 376
widgets.combobox.disabled, 376
widgets.combobox.entrybackground, 388
widgets.combobox.entrydisabled, 386
widgets.combobox.entryfont, 387
widgets.combobox.entryforeground, 388
widgets.combobox.entryjustify, 388
widgets.combobox.entryrelief, 387
widgets.combobox.entrywidth, 386
widgets.combobox.exportselection, 375
widgets.combobox.get, 371
widgets.combobox.getentry, 373
widgets.combobox.hscrollbar, 379
widgets.combobox.insert, 377
widgets.combobox.insertentry, 378
widgets.combobox.labelanchor, 385
widgets.combobox.labelbackground, 385
widgets.combobox.labelfont, 383
widgets.combobox.labelforeground, 384
widgets.combobox.labeljustify, 384
widgets.combobox.labelrelief, 384

widgets.combobox.labeltext, 382
widgets.combobox.labelwidth, 383
widgets.combobox.listboxheight, 390
widgets.combobox.pixelheight, 371
widgets.combobox.pixelwidth, 370
widgets.combobox.popupcursor, 382
widgets.combobox.select, 372
widgets.combobox.selection, 372
widgets.combobox.vscrollbar, 379
widgets.deleteregions, 391, 392
widgets.deleteregions.done, 394
widgets.deleteregions.gui, 394
widgets.deleteregions.refresh, 395
widgets.deleteregions.setregions, 395
widgets.deleteregions.settable, 396
widgets.dialogbox, 398
widgets.extendoptionmenu, 399, 401
widgets.extendoptionmenu.disabled, 416
widgets.extendoptionmenu.done, 417
widgets.extendoptionmenu.extend, 404
widgets.extendoptionmenu.findlabel, 411
widgets.extendoptionmenu.geteventmask, 412
widgets.extendoptionmenu.getindex, 408
widgets.extendoptionmenu.getlabel, 410
widgets.extendoptionmenu.getlabels, 410
widgets.extendoptionmenu.getpreviousindex, 409
widgets.extendoptionmenu.replace, 405
widgets.extendoptionmenu.selectindex, 406
widgets.extendoptionmenu.selectlabel, 407
widgets.extendoptionmenu.setbackground, 415
widgets.extendoptionmenu.seteventmask, 413
widgets.extendoptionmenu.setforeground, 414
widgets.extendoptionmenu.setwidth, 415
widgets.fontchooserwidget, 417, 418
widgets.gchooser, 419, 421
widgets.gchoosertest, 423
widgets.gopher, 423, 424
widgets.gopher.fromimage, 425
widgets.gopher.fromms, 426
widgets.guientry, 427, 431
widgets.guientry.array, 434
widgets.guientry.boolean, 436
widgets.guientry.booleanarray, 435
widgets.guientry.check, 439
widgets.guientry.choice, 437
widgets.guientry.direction, 449
widgets.guientry.done, 460
widgets.guientry.doppler, 446
widgets.guientry.epoch, 448
widgets.guientry.file, 440
widgets.guientry.frequency, 444
widgets.guientry.list, 458
widgets.guientry.measurecodes, 441
widgets.guientry.minmaxhist, 463
widgets.guientry.position, 447
widgets.guientry.quantity, 450
widgets.guientry.radialvelocity, 445
widgets.guientry.range, 452
widgets.guientry.record, 453
widgets.guientry.region, 454
widgets.guientry.scalar, 433
widgets.guientry.scalarmeasure, 443
widgets.guientry.string, 455
widgets.guientry.twoentry, 461
widgets.guientry.type, 461
widgets.guientry.untyped, 456
widgets.guientrytest, 465
widgets.helpmenu, 466, 467
widgets.helpmenu.done, 469
widgets.helpmenu.reset, 468
widgets.histogramgui, 470, 472
widgets.histogramgui.disable, 476
widgets.histogramgui.dismiss, 476
widgets.histogramgui.done, 476
widgets.histogramgui.enable, 477
widgets.histogramgui.gui, 473
widgets.histogramgui.newdata, 474

widgets.histogramgui.setselection, 475
widgets.histogramgui.setstats, 474
widgets.is_widgetserver, 693, 694
widgets.listbox, 477, 587, 593, 606
widgets.listboxcontainer, 477, 479
widgets.listboxcontainer.clear, 485
widgets.listboxcontainer.delete, 485
widgets.listboxcontainer.done, 481
widgets.listboxcontainer.fromforeign, 483
widgets.listboxcontainer.get, 486
widgets.listboxcontainer.hasitem, 484
widgets.listboxcontainer.hscrollbar, 481
widgets.listboxcontainer.insert, 487
widgets.listboxcontainer.itemnames, 484
widgets.listboxcontainer.see, 488
widgets.listboxcontainer.seeoninsert, 483
widgets.listboxcontainer.select, 488
widgets.listboxcontainer.selection, 489
widgets.listboxcontainer.vscrollbar, 482
widgets.menuframes, 503, 505
widgets.menuframes.add, 506
widgets.menuframes.Delete, 507
widgets.menuframes.disable, 509
widgets.menuframes.done, 510
widgets.menuframes.enable, 510
widgets.menuframes.state, 511
widgets.menuframes.switch, 508
widgets.messageline, 512, 513
widgets.messageline.clear, 515
widgets.messageline.done, 515
widgets.messageline.post, 515
widgets.messageline.postnoforward, 516
widgets.multiautogui, 489, 491
widgets.multiautogui.addautogui, 493
widgets.multiautogui.fillgui, 493
widgets.multiautogui.multiautoguitest, 494
widgets.multiautogui.show, 492
widgets.multiautogui.shownone, 492
widgets.multiscale, 495, 496
widgets.multiscale.disable, 498
widgets.multiscale.dismiss, 502
widgets.multiscale.enable, 498
widgets.multiscale.getvalues, 499
widgets.multiscale.setrange, 500
widgets.multiscale.setresolution, 501
widgets.multiscale.setvalues, 501
widgets.optionmenu, 516, 518
widgets.optionmenu.disabled, 540
widgets.optionmenu.done, 541
widgets.optionmenu.extend, 520
widgets.optionmenu.findlabel, 532
widgets.optionmenu.findname, 533
widgets.optionmenu.findvalue, 534
widgets.optionmenu.geteventmask, 535
widgets.optionmenu.getindex, 526
widgets.optionmenu.getlabel, 528
widgets.optionmenu.getlabels, 528
widgets.optionmenu.getname, 529
widgets.optionmenu.getnames, 530
widgets.optionmenu.getpreviousindex, 527
widgets.optionmenu.getvalue, 531
widgets.optionmenu.getvalues, 532
widgets.optionmenu.isenabled, 540
widgets.optionmenu.replace, 521
widgets.optionmenu.selectindex, 523
widgets.optionmenu.selectlabel, 523
widgets.optionmenu.selectname, 524
widgets.optionmenu.selectvalue, 525
widgets.optionmenu.setbackground, 538
widgets.optionmenu.seteventmask, 536
widgets.optionmenu.setforeground, 538
widgets.optionmenu.setlabel, 537
widgets.optionmenu.setwidth, 539
widgets.pixelrange, 542

widgets.pixelrange.disableallbutton, 544
widgets.pixelrange.disabled, 544
widgets.pixelrange.done, 545
widgets.pixelrange.enableallbutton, 545
widgets.pixelrange.getradiovalue, 546
widgets.pixelrange.getslidervalues, 546
widgets.pixelrange.setradiovalue, 548
widgets.pixelrange.setrange, 547
widgets.popupselectmenu, 548, 549
widgets.popupselectmenu.disable, 550
widgets.popupselectmenu.enable, 551
widgets.popupselectmenu.kill, 552
widgets.profilewidthentry, 552, 553
widgets.profilewidthentry.disable, 554
widgets.profilewidthentry.done, 555
widgets.profilewidthentry.enable, 555
widgets.profilewidthentry.getvalue, 556
widgets.radiobuttons, 556, 557
widgets.radiobuttons.disablebutton, 559
widgets.radiobuttons.disabled, 558
widgets.radiobuttons.done, 560
widgets.radiobuttons.enablebutton, 560
widgets.radiobuttons.getvalue, 561
widgets.radiobuttons.reset, 561
widgets.radiobuttons.setstate, 562
widgets.recordbrowserwidget, 563
widgets.restoreregions, 565, 566
widgets.restoreregions.done, 568
widgets.restoreregions.gui, 568
widgets.restoreregions.refresh, 569
widgets.restoreregions.regions, 569
widgets.restoreregions.settable, 570
widgets.rollup, 571, 572
widgets.rollup.disable, 577
widgets.rollup.done, 579
widgets.rollup.down, 576
widgets.rollup.enable, 578
widgets.rollup.frame, 574
widgets.rollup.hidden, 579
widgets.rollup.setpopuphelp, 580
widgets.rollup.switch, 575
widgets.rollup.up, 577
widgets.saveregions, 581, 582
widgets.saveregions.done, 584
widgets.saveregions.gui, 584
widgets.saveregions.setregions, 585
widgets.saveregions.settable, 586
widgets.scrolllistbox, 477, 587, 588, 606
widgets.scrolllistbox.done, 590
widgets.scrolllistbox.hscrollbar, 591
widgets.scrolllistbox.listbox, 590
widgets.scrolllistbox.seeoninsert, 592
widgets.scrolllistbox.vscrollbar, 591
widgets.scrolltext, 593, 594
widgets.scrolltext.done, 596
widgets.scrolltext.hscrollbar, 597
widgets.scrolltext.text, 596
widgets.scrolltext.vscrollbar, 597
widgets.selectablelist, 598, 599
widgets.selectablelist.disabled, 601
widgets.selectablelist.done, 602
widgets.selectablelist.replace, 602
widgets.selectablelist.setbackground, 603
widgets.selectablelist.setforeground, 604
widgets.selectablelist.setlabel, 604
widgets.selectablelist.setwidth, 605
widgets.synclistboxes, 606, 608
widgets.synclistboxes.background, 611
widgets.synclistboxes.bind, 612
widgets.synclistboxes.borderwidth, 613
widgets.synclistboxes.clear, 614
widgets.synclistboxes.delete, 614
widgets.synclistboxes.done, 610
widgets.synclistboxes.exportselection, 615
widgets.synclistboxes.font, 616
widgets.synclistboxes.foreground, 616
widgets.synclistboxes.get, 617

widgets.synclistboxes.height, 618
widgets.synclistboxes.hscrollbar, 618
widgets.synclistboxes.insert, 619
widgets.synclistboxes.leadbox, 620
widgets.synclistboxes.listbox, 610
widgets.synclistboxes.mode, 620
widgets.synclistboxes.nearest, 621
widgets.synclistboxes.relief, 622
widgets.synclistboxes.see, 622
widgets.synclistboxes.seeoninsert, 623
widgets.synclistboxes.select, 623
widgets.synclistboxes.selection, 624
widgets.synclistboxes.vscrollbar, 625
widgets.synclistboxes.width, 625
widgets.tabdialog, 626, 627
widgets.tabdialog.add, 628
widgets.tabdialog.available, 629
widgets.tabdialog.delete, 630
widgets.tabdialog.deleteall, 631
widgets.tabdialog.dialogframe, 632
widgets.tabdialog.done, 633
widgets.tabdialog.front, 634
widgets.tabdialog.list, 635
widgets.tabdialog.replace, 636
widgets.tabdialog.which, 637
widgets.tabdialogtest, 638
widgets.table, 645
widgets.tapedeck, 639
widgets.tapedeck.disable, 644
widgets.tapedeck.dismiss, 642
widgets.tapedeck.done, 642
widgets.tapedeck.enable, 643
widgets.TaQL, 645
widgets.taqlwidget, 645, 646
widgets.Tk, 320, 339, 344, 363, 391, 398, 400, 417, 419, 423, 428, 466, 477, 495, 503, 512, 516, 542, 549, 553, 557, 563, 565, 571, 581, 587, 593, 598, 606, 626, 639
widgets.twodactionoptionmenu, 649
widgets.twodactionoptionmenu.currentvalue, 651
widgets.twodactionoptionmenu.newselection, 652
widgets.widget, 645
widgets.widgets, 320, 323, 327, 339, 344, 363, 391, 398, 400, 417, 419, 423, 428, 466, 477, 489, 495, 503, 512, 516, 542, 549, 553, 557, 563, 565, 571, 581, 587, 593, 598, 606, 626, 639, 649, 652
widgets.widgetserver, 652, 658
widgets.widgetserver.actionoptionmenu, 671
widgets.widgetserver.addpopuphelp, 692
widgets.widgetserver.blinklabel, 677
widgets.widgetserver.busy, 689
widgets.widgetserver.button, 659
widgets.widgetserver.canvas, 665
widgets.widgetserver.checkmenu, 672
widgets.widgetserver.combobox, 676
widgets.widgetserver.deleteregions, 681
widgets.widgetserver.dialogbox, 672
widgets.widgetserver.entry, 663
widgets.widgetserver.extendoptionmenu, 670
widgets.widgetserver.fontchooser, 677
widgets.widgetserver.frame, 658
widgets.widgetserver.guientry, 673, 684
widgets.widgetserver.have_gui, 667
widgets.widgetserver.helpmenu, 687
widgets.widgetserver.label, 662
widgets.widgetserver.listbox, 664
widgets.widgetserver.menuframes, 688
widgets.widgetserver.message, 664
widgets.widgetserver.messageline, 674
widgets.widgetserver.multiscale, 678
widgets.widgetserver.notbusy, 690
widgets.widgetserver.optionmenu,

669
widgets.widgetserver.pgplot, 666
widgets.widgetserver.popuphelp, 691
widgets.widgetserver.popupmenu, 692
widgets.widgetserver.popupremove, 692
widgets.widgetserver.popupselectmenu, 679
widgets.widgetserver.recordbrowser, 679
widgets.widgetserver.resources, 686
widgets.widgetserver.restoreregions, 680
widgets.widgetserver.rollup, 668
widgets.widgetserver.saveregions, 681
widgets.widgetserver.scale, 660
widgets.widgetserver.scrollbar, 661
widgets.widgetserver.scrolllistbox, 674
widgets.widgetserver.selectablelist, 682
widgets.widgetserver.setmode, 684
widgets.widgetserver.synclistboxes, 675
widgets.widgetserver.tabdialog, 683
widgets.widgetserver.tapedeck, 682
widgets.widgetserver.text, 661
widgets.widgetserver.tk_hold, 667
widgets.widgetserver.tk_iconpath, 668
widgets.widgetserver.tk_release, 667
widgets.widgetserver.type, 693
widgets.widgetserver.unsetmode, 685
widgets:histogram, 470
widgetserver, 652, 658
widgetset, 184
width, 625
writeps, 198
writexpm, 199, 268

zlength, 227
zoom, 285