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* *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* *Allowed: String from 'x', 'y', 'both', 'none'* *Default: 'none'*
widgetset	*The widget server to use* *Allowed: widgetserver tool* *Default: dws*

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* *Allowed: String from 'x', 'y', 'both', 'none'* *Default: 'none'*
widgetset	*The widget server to use* *Allowed: widgetserver tool* *Default: dws*

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'*

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

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'* *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* *Allowed: String from 'x', 'y', 'both', 'none'* *Default: 'none'*
widgetset	*The widget server to use* *Allowed: widgetserver tool* *Default: dws*

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 'min' and 'max'. This is used by minmaxhist, and is not intended for independant use.*
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

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=OJ287, 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* *Default: 'none'*
widgetset	*widgetserver to use to make label* *Allowed: widgetserver toolXS* *Default: dws*

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 window exits';

  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* *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* *Allowed: String from 'x', 'y', 'both', 'none'* *Default: 'none'*
widgetset	*The widget server to use* *Allowed: widgetserver tool* *Default: dws*

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');
```

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* *Allowed: String from 'c', 'n', 's', 'e', 'w', 'ne', 'nw', 'se', 'sw'* *Default: 'c'*
fill	*How to expand when resized* *Allowed: String from 'x', 'y', 'both', 'none'* *Default: 'none'*
widgetset	*The widget server to use* *Allowed: widgetserver tool* *Default: dws*

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’*
widgetset	*Widget set to use* *Allowed: A widgetserver closure tool* *Default: dws*

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* `widgetserver`*s 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;
```

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.