

Package ‘gWidgets’

September 9, 2009

Version 0.0-37

Title gWidgets API for building toolkit-independent, interactive GUIs

Author John Verzani. Based on the iwidgets code of Simon Urbanek and suggestions by Simon Urbanek, Philippe Grosjean and Michael Lawrence

Maintainer John Verzani <gwidgetsrgtk@gmail.com>

Depends methods, utils

Suggests gWidgetsRGtk2, gWidgetstcltk

Description gWidgets provides a toolkit-independent API for building interactive GUIs. At least one of the gWidgetsXXX packages, such as gWidgetstcltk, needs to be installed. Some icons are on loan from the scigraphica project <http://scigraphica.sourceforge.net>.

License GPL (>= 2)

URL <http://www.math.csi.cuny.edu/pmg>

LazyLoad yes

Repository CRAN

Date/Publication 2009-09-09 12:39:49

R topics documented:

| | |
|----------------------------|----|
| gWidgets-package | 2 |
| gbutton | 4 |
| gcheckbox | 5 |
| gcheckboxgroup | 6 |
| gcombobox | 8 |
| gcommandline | 9 |
| gdf | 10 |
| gedit | 12 |
| gfile | 13 |
| gformlayout | 14 |

| | |
|---------------------------------|-----------|
| ggenericwidget | 18 |
| ggraphics | 20 |
| ggroup | 21 |
| ghelp | 24 |
| ghtml | 25 |
| gimage | 26 |
| glabel | 27 |
| glayout | 28 |
| gmenu | 29 |
| gnotebook | 32 |
| gpanedgroup | 33 |
| gradio | 34 |
| gseparator | 36 |
| gslider | 36 |
| gstatusbar | 38 |
| gtable | 39 |
| gtext | 41 |
| gtree | 43 |
| guiToolkit | 45 |
| guiWidget-class | 46 |
| gvarbrowser | 48 |
| gWidgets-dialogs | 49 |
| gWidgets-dnd | 51 |
| gWidgets-handlers | 53 |
| gWidgets-icons | 56 |
| gWidgets-methods | 57 |
| gWidgets-undocumented | 59 |
| gwindow | 60 |
| Index | 62 |

gWidgets-package *gWidgets API for building toolkit-independent, interactive GUIs*

Description

gWidgets provides a toolkit-independent API for building interactive GUIs

Details

The gWidgets package creates an API for interacting with GUI toolkits. This package is toolkit neutral. The interactions is provided by a further package, such as gWidgetsRGtk2, or gWidgetsr-Java.

Some details on the installation of toolkits is provided by `installing_gWidgets_toolkits`.

Index:

| | |
|--------------------------|--|
| gWidgets-dialogs | Basic dialog constructors |
| gWidgets-dnd | Functions to add drag and drop ability to widgets |
| gWidgets-handlers | Methods to add event handlers to objects |
| gWidgets-icons | Functions for adding icons |
| gWidgets-methods | Methods for gWidgets instances |
| gWidgets-undocumented.Rd | Undocumented, but exported, functions |
| gaction | Reusable objects for buttons, menus, toolbars |
| gcheckbox | Constructor of widget to indicate whether a value is desired or not |
| gcommandline | A command line interface |
| gdf | Constructor of widget to edit a data frame |
| gedit | Constructors for widgets to handle text input |
| gfile | Dialogs for file and date selection |
| ggenericwidget | A constructor to create widgets for evaluating functions |
| ggraphics | Constructor for a toolkit specific plot device and a notebook to wrap plots in |
| ggroup | Containers for packing in subsequent widgets |
| ghelp | Widget to interface with help pages |
| ghtml | widget to show html. (not implemented) |
| glabel | Constructors for widgets that show text or images |
| glayout | A container for aligning widgets in a table |
| gmenu | Constructors to make menubar or toolbars |
| gnotebook | constructor for notebook widget |
| gpanedgroup | A paned group holds two widgets with a handle between them to adjust the amount of space allocated to each |
| gradio | Widgets to allow selection from a vector of items |
| gseparator | Widget to draw separator line |
| gslider | Constructors for widgets to select a value from a sequence. |
| gstatusbar | Constructor of status bar widget |
| gtable | Constructor for widget to display tabular data |
| gtree | Constructor for widget to display heirarchical data |
| guiToolkit | Function to select the GUI toolkit used by gWidgets |
| guiWidget-class | Class "guiWidget" ~~~ |
| gvarbrowser | Widget for browsing environment |
| gwindow | Constructor for base container |

Author(s)

Philippe Grosjean, Michael Lawrence, Simon Urbanek, John Verzani

Maintainer: John Verzani <gwidgetsrgtk@gmail.com>

gbutton

Button constructors

Description

A button widget is used to present a widget that a user can press to initiate some action.

Buttons show text and/or images in a clickable object whose shading indicates that the button is to be clicked on.

Usage

```
gbutton(text = "", border=TRUE, handler = NULL, action = NULL, container = NULL,
        ..., toolkit = guiToolkit())
```

Arguments

| | |
|-----------|---|
| text | Text to show in the button. For buttons, if this text matches a stock icon name, an icon is shown as well. |
| border | If TRUE a border is drawn to make a button look like a button. If FALSE, the no border so the button looks like a label. |
| handler | Handler called on a click event |
| action | Either a <code>gaction</code> instance in which case, the <code>text</code> and <code>handler</code> arguments are not used, or is an R object to be passed to the specified handler. |
| container | Optional container to attach widget to. |
| ... | Passed to <code>add</code> method of container |
| toolkit | Which GUI toolkit to use |

Details

As buttons are intended to show the user how to initiate some action, they are often labeled as commands. Additionally, if the action is not currently possible given the state of the GUI, a button is typically disabled. This can be done through the `enabled` method.

The `svalue()` method returns the value of the widget. For a button, this is the text as a single string (which may not include a "\n" for newlines if not supported by the toolkit).

The `svalue<-` method can be used to set the text of the widget. For buttons, values with length greater than one are pasted together collapsed with "\n".

The `addHandlerChanged` method is aliased to the `addHandlerClicked` method which can be used to set a handler to respond to click events.

When the `action` argument is a `gaction` instance, then the button label and handler will be derived from the `gaction` instance. The `enabled<-` method of the `gaction` instance should be used to set the sensitivity to user input, not the `enabled<-` method of the `gbutton` instance.

Examples

```
## Not run:
## button group example
w <- gwindow("Button examples")
g <- ggroup(cont = w)
addSpring(g) ## push to right of widget
gbutton("help", cont = g)
addSpace(g, 20) ## some breathing room
gbutton("cancel", cont = g)
gbutton("ok", cont = g, handler = function(h, ...) cat("do it\n"))
## End(Not run)
```

gcheckbox

*Constructor of widget to indicate whether a value is desired or not***Description**

A checkbox shows a value and a box to check indicating if the value is desired or not.

Usage

```
gcheckbox(text, checked = FALSE, handler = NULL, action = NULL,
        container = NULL, ..., toolkit = guiToolkit())
```

Arguments

| | |
|-----------|---|
| text | Text to show by box |
| checked | Logical indicating initial state of box |
| handler | Called when box is toggled. |
| action | Passed to handler |
| container | Optional container to attach widget to. |
| ... | |
| toolkit | Which toolkit to use? |

Details

The value of the widget is either `TRUE` or `FALSE`.

The `svalue` method returns a logical indicating `TRUE` if the box is checked.

The `svalue<-` method can be used to set the value using a logical.

The `"["` method returns the label on the box.

The `"[<-"` method can be used to change the label on the box.

The default handler is set by the `addHandlerClicked` method. This is called when the button is toggled. If one wishes to have the handler called only when checked to indicate `TRUE`, say, one should check the state of the widget in the handler.

Author(s)

John Verzani

See AlsoMethods for gComponent objects are detailed in [gWidgets-methods](#).Event Handlers are detailed in [gWidgets-handlers](#).**Examples**

```
## Not run:
gcheckbox("checked", container=TRUE, handler=function(h,...) {
  cat("The widget is checked?", svalue(h$obj), "\n")
})
## End(Not run)
```

gcheckboxgroup

*Widget to allow multiple selection from a vector of items***Description**

Widgets to select one (or several) from a given vector of items. These are a radio group where all values are shown at once, but only one may be selected; a checkbox group where more than one may be selected; and a combo box (or droplist) where initially only a single value is shown, and the others are a mouse click away,

Usage

```
gcheckboxgroup(items, checked = FALSE, horizontal = FALSE,
  handler = NULL, action = NULL, container = NULL, ..., toolkit = guiToolkit())
```

Arguments

| | |
|------------|---|
| items | Vector of values to select from |
| checked | A logical vector indicating initial values. |
| horizontal | A logical specifying the layout for gradio and gcheckboxgroup |
| handler | Called when selection is changed |
| action | Passed to handler when called. |
| container | Optional container to attach widget to |
| ... | Passed to add method of container |
| toolkit | Which GUI toolkit to use |

Details

The `svalue` method returns the selected values by name. If the extra argument `index=TRUE` is specified, the indices of the selected values is given.

The `svalue<-` method can be used to set the selected value. This widget is a cross between a checkbox and a radio button group. As such, there are different ways to specify the state. As with a checkbox, the argument can be a logical vector indicating which checkboxes are to be checked (recycling is done). As with a radio button group, the value can be a character vector indicating by label which checkboxes are to be checked; or if the `index=TRUE` argument is given, a vector of indices for those checkboxes to be checked.

The `"["` method refers to the vector defining the items.

The `"[<-"` method can be used to change the vector defining the items. The length should be the same as the original, although in some toolkits this isn't necessary.

The `"length"` method returns the number of items.

See Also

A checkboxgroup is one of several ways to select a value of a set of items. See also

[gcheckbox](#), [gradio](#), [gcombobox](#), and [gtable](#).

Methods for gComponent objects are detailed in [gWidgets-methods](#).

Event Handlers are detailed in [gWidgets-handlers](#).

Examples

```
## Not run:
flavors <- c("vanilla", "chocolate", "strawberry")

f <- function(h,...) print(
  paste("Yum",
    paste(svalue(h$obj),collapse=" and "),
    sep = " "))

w <- gwindow("checkbox example")
gp <- ggroup(container=w)
glabel("Favorite flavors:",cont=gp)
cbg <- gcheckboxgroup(flavors, cont=gp, handler=f)

svalue(cbg) <- c(TRUE, FALSE, TRUE)
svalue(cbg)
svalue(cbg) <- "vanilla"
svalue(cbg, index=TRUE) <- 1:2
cbg[3] <- "raspberry"

## End(Not run)
```

gcombobox

*Widgets to allow selection from a vector of items***Description**

A combobox allows selection of a value from a list of items using a popup menu. Additionally, the widget can combine a text entry widget for user input outside of the pre-set list of items. A combobox is useful for selection from a moderate size of items (2-30, say). For smaller sets of items, a radio button group is a possibility, for larger sets of items, a scrollable (and perhaps searchable) table may be preferred.

Usage

```
gcombobox(items, selected = 1, editable = FALSE, coerce.with=NULL, handler = NULL,
          action = NULL, container = NULL, ..., toolkit = guiToolkit())
```

```
gdroplist(items, selected = 1, editable = FALSE, coerce.with=NULL, handler = NULL,
          action = NULL, container = NULL, ..., toolkit = guiToolkit())
```

Arguments

| | |
|--------------------------|---|
| <code>items</code> | Vector of values to select from. This may also be a data frame, in which case the first column is the vector of values, the second (if present) indicates a stock icon to display with the item (not all toolkits), and the third (if present) will indicate a tooltip to display (not all toolkits). |
| <code>selected</code> | For gradio the initial selected value (as an index) For a drop list, the first selected value. Use 0 to leave blank |
| <code>editable</code> | A logical indicating if the user can add an entry to the list of available answers |
| <code>coerce.with</code> | Apply this function to selected value before returning |
| <code>handler</code> | Called when selection is changed |
| <code>action</code> | Passed to handler when called. |
| <code>container</code> | Optional container to attach widget to |
| <code>...</code> | Passed to add method of container |
| <code>toolkit</code> | Which GUI toolkit to use |

Details

The initial `items` can be a vector or a data frame. Not all tool kits do something with the extra columns in the data frame.

The `svalue` method returns the selected value by name. Assume the value is a character vector. Use the `coerce.with` argument to return a value of a different type. If the extra argument `index=TRUE` is specified, the index of the selected value is given.

The `svalue<-` method can be used to set the selected value. This is done my name or if the argument `index=TRUE` is given by index. The value can be a data frame, in which case the first column is used to match against the current items.

The "[" method refers to the vector defining the items.

The "[<-" method can be used to change the vector defining the items.

The "length" method returns the number of items.

For gcombobox the argument `editable=TRUE` adds a text-edit box where the user can type in a selection. By default this value is returned as a character by `svalue`. Use `coerce.with` to coerce this prior to returning.

See Also

A combobox is one of several ways to select a value of a set of items. See also [gcheckbox](#), [gradio](#), [gcheckboxgroup](#), and [gtable](#).

Methods for gComponent objects are detailed in [gWidgets-methods](#).

Event Handlers are detailed in [gWidgets-handlers](#).

Examples

```
## Not run:
flavors <- c("vanilla", "chocolate", "strawberry")

f <- function(h,...) print(
  paste("Yum",
    paste(svalue(h$obj),collapse=" and "),
    sep = " ")

w <- gwindow("combobox example")
gp <- ggroup(container=w)
glabel("Favorite flavor:", cont=gp)
cb <- gcombobox(flavors, editable=TRUE, cont=gp, handler=f)

svalue(cb) <- "vanilla"
svalue(cb)
cbg[3] <- "raspberry"

## End(Not run)
```

gcommandline

A command line interface

Description

This constructs a simple command line interface for R

Usage

```
gcommandline(command = "", assignto = NULL, useGUI = TRUE, useConsole = FALSE, prompt = "",
  guiToolkit())
```

Arguments

| | |
|------------|--|
| command | Initial command to evaluate |
| assignto | Assigns output to this variable if non-NULL |
| useGUI | Is result also printed to GUI. Use FALSE to get text-only instance |
| useConsole | Is result also printed to console? |
| prompt | Prompt to use |
| width | Width of widget in pixels |
| height | Height of widget in pixels |
| container | Optional container to attach to |
| ... | Ignored for now |
| toolkit | Which GUI toolkit to use |

Details

Additional commands can be added programmatically with the `svalue<-` method. The value assigned is a string containing the command. If it has a `names` attribute, this is taken as the variable name to assign the output to.

The `svalue` method returns the command history.

The `"["` method can be used to retrieve the command history as well.

Examples

```
## Not run:
obj = gcommandline(container=TRUE)
svalue(obj) <- "2+2"
## assign to x
command = "rnorm(100)"; names(command) = "x"
svalue(obj) <- command
## look at history
obj[]
## End(Not run)
```

gdf

Constructor of widget to edit a data frame

Description

A widget used to edit data frames

Usage

```
gdf(items = NULL, name = deparse(substitute(items)), do.subset = FALSE,
container = NULL, ..., toolkit = guiToolkit())
```

```
gdfnotebook(items = NULL, container = NULL, ..., toolkit=guiToolkit())
```

Arguments

| | |
|-----------|--|
| items | data frame to be edited |
| name | Name of data frame |
| do.subset | A logical. If TRUE a means to filter the output using logical values is given. |
| container | An optional container to attach widget to |
| ... | Can be used to override default colors. |
| toolkit | Which GUI toolkit to use |

Details

This widget is similar to that provided by [gtable](#) only this is intended for editing of data frames. The `gdfnotebook` widget uses a notebook to hold several data frames at once.

In `gWidgetsRGtk2`, the table shown can be edited. Double click in a cell to allow this. When editing, the value is saved when the cell is left. This is done using the up or down arrow keys, the Enter key, the Tab key or the mouse. The down arrow key moves the cursor down a row, extending the size of the data frame if necessary. The Enter key moves the cursor down, but does not extend the data frame if at the bottom. The Tab key moves to the right. If at the end, a dialog to add a new variable pops up.

Right clicking in a cell that is not currently being edited pops up a menu to edit the column names, sort or apply a function to a row.

If `do.subset=TRUE` then one can filter using subsetting by a single variable. The variable may be selected by a droplist, the logical expression can be entered or one selected from a droplist.

If more complicated filtering is desired, the `visible<-` method may be used, its value should be a logical vector of length given by the number of rows of the data frame.

The `svalue` method returns the selected value.

The `svalue<-` method is used to select rows without a mouse, the value is a set of row numbers.

The `"["` method is used for data-frame like extraction from the object.

The `"[<-"` method can be used for data-frame like assignment, with limitations.

The `dim`, `dimnames`, `dimnames<-`, `length`, `names`, and `names<-` methods should work on the object as they do for data frames.

The `addhandlerchanged` handler responds to changes in the values of the data frame.

See Also

[gtable](#)

Examples

```
## Not run:
  obj <- gdf(mtcars, container=gwindow("mtcars"), do.subset=TRUE)
  obj[1,1]
  obj[1,]
  obj[,1]
  obj[1,1] <- 21
  obj[,] <- head(mtcars) ## replace df
## End(Not run)
```

gedit

Constructor for widget to handle single-line text input

Description

The `gedit` widget is used to enter single lines of text.

Usage

```
gedit(text = "", width = 25, coerce.with = NULL, handler = NULL, action = NULL, con
```

Arguments

| | |
|--------------------------|--|
| <code>text</code> | Initial text in widget |
| <code>width</code> | Width of widget. For <code>gedit</code> , this means the number of characters. |
| <code>coerce.with</code> | For <code>gedit</code> , when the value is retrieved this function is applied to the result. (The stored value is always a character, this can be used to make it numeric, to quote it, ...) |
| <code>handler</code> | Handler called when text is changed. For <code>gedit</code> , this means the enter key is pressed. |
| <code>action</code> | Passed to handler |
| <code>container</code> | Optional container to attach widget to |
| <code>...</code> | Passed to add method of container |
| <code>toolkit</code> | Which GUI toolkit to use |

Details

The `gedit` widget has the following methods:

The `svalue` method retrieves the value. If a function is given to the argument `coerce.with` it is applied before the value is returned. This can be used to coerce the text value (always of class `character`) to a numeric, or to a date, or to be quoted, ...

The `svalue<-` method is used to set the value.

The `"["` and `"[<-"` methods refer to the widgets "type-ahead" values. A familiar usage is when a url is typed into a web browser, matches appear from a users history that could possibly complete the typed url.

Examples

```
## Not run:
  gedit("type here", container=gwindow())

## change handler
obj <- gedit(container=gwindow())
addhandlerchanged(obj, handler=function(h,...)
  cat("You typed", svalue(h$obj), "\n"))
```

```

## coerce to numeric
obj <- gedit("7", container=gwindow(), coerce.with=as.numeric)
svalue(obj)

## End(Not run)

```

gfile

*Dialogs for file and date selection***Description**

These functions provide dialogs for file selection (files or directories) and date selections.

Usage

```

gfile(text = "", type = c("open", "save", "selectdir"), initialfilename = NULL,
      filter = list("All files" = list(patterns = c("*")), "R files" =
        list(patterns = c("*.R", "*.Rdata")),
        "text files" = list(mime.types = c("text/plain")))
      ), handler = NULL, action = NULL, ..., toolkit = guiToolkit())

```

```

gfilebrowse (text = "Select a file...", type = "open", quote = TRUE,
            container = NULL, ..., toolkit = guiToolkit())

```

```

gcalendar(text = "", format = "%Y-%m-%d", handler=NULL, action=NULL, container = NU

```

Arguments

| | |
|-----------------|--|
| text | Initial text |
| type | When selecting a file it can be selected for opening, for saving or you may want to select a directory. |
| initialfilename | Suggested name for file save |
| filter | Filter for files shown during selection |
| quote | Is result quoted |
| format | Format of date |
| handler | Handler for when file is changed. The component <code>file</code> of the first argument contains the file name |
| action | Passed to handler |
| container | Optional container to attach widget to |
| ... | Passed to <code>gedit</code> instance |
| toolkit | Which GUI toolkit to use |

Details

The `gfile` dialog is modal, meaning no action can take place until a selection is made. Whereas the `gfilebrowse` dialog consists of a `gedit` instance to hold a filename and a button to click if the dialog is desired to fill in this filename.

The `gcalendar` widget is similar to the `gfilebrowse` widget.

For both `gcalendar` and `gfilebrowse` any `...` arguments are passed to `gedit`. The `coerce.with` argument can be used to here to quote the values, or coerce them otherwise such as with `as.Date`. Otherwise, the `svalue` method returns a character string containing the value shown in the `gedit` box.

The `svalue<-()` method may be used to set the value for both `gcalendar` and `gfilebrowse`.

The return value is the filename selected. A `NA` value is returned if no file is selected.

Examples

```
## Not run:
## source a file using a handler
sourceFile <- function() gfile("Select a file",type="open", handler =
function(h,...) source(h$file))

## source a file using fact that dialog is modal
source(gfile())

## apply a generic action to the file
countLines <- function(filename) print(length(readLines(filename)))
chooseFile <- function() gfile("Select a file",type="open",
action="countLines", handler = function(h,...) do.call(h$action,list(h$file)))
## End(Not run)
```

`gformlayout`

A constructor for laying out groups of widgets from a template defined by a list

Description

This constructor takes a list that defines the layout of widgets and pieces them together to create a form or dialog. It is similar to `ggenericwidget` but offers more flexibility with the layout, but does not offer the automatic creation of the widget using a functions `formals`.

Usage

```
gformlayout(lst, container = NULL, ..., toolkit = guiToolkit())
```

Arguments

| | |
|------------------------|--|
| <code>lst</code> | A list that defines the layout of the containers. See the details section. |
| <code>container</code> | Optional parent container for the widget |
| <code>...</code> | Passed to <code>add</code> method of parent container when given |
| <code>toolkit</code> | Which GUI toolkit to use |

Details

The list defining the layout has the following key named components:

type The type is the name of a `gWidgets` constructor or "fieldset". The latter specifies that the children should be layed out using a table. The type can specify either a container constructor or component constructor

children For containers, this specifies the children using a list. Each child is a given as a component of this list. Children can be containers and hence contain other children, to match the heirarchical layout common in GUIs.

name If a name is specified, then this widget will be stored in a list that can be accessed by the methods `svalue` or `\[`

depends.on The name of a widget previously specified through the `name` argument. If given, then a handler is added to the widget that controls whether this new widget/container should be enabled.

depends.FUN When `depends.on` is specified, this function is consulted to see if the widget should be enabled. This function has argument `value` which is the return value of `svalue` on the named widget this new one depends on. It should return a logical value indicating if the new widget is to be enabled.

depends.signal By default, the signal the handler specified through `depends.FUN` is given by `addHandlerChanged`, this allows on to specify a different `addHandler` method. See the example.

If the type is `gnotebook`, then each child should have a `label` component.

The new constructor `fieldset` allows the organization of its children in a table. These children should not be other containers. If the component `label` is non-null, the table is packed into a `gframe` container. The default number of columns is just 1, but specifying `columns` in the list can increase this. Children are packed in row by row when more than one column is given.

The labels can be adjusted. The component `label.pos` can be "left" (the default) for a label to the left of the widget, or "top" to have the label on top of the widget. When the position is "left", then the `label.just` component is consulted for justification of the label. This can have a value of "right" (the default), "center" or "left"

If a component `label.font` is given, then this will be applied to each label through the `font` method of the label.

The children are specified as a list. Each child should have a `type` component, a `label` component and a `name` component. Other components are passed to the specified constructor in `type` through `do.call`.

The return object has a few methods defined for it.

The `\[` method returns a list with named components storing the objects created by the constructors. Subsetting is allowed. No `\[\[` method is given, instead the `drop=TRUE` argument can be used with a single index is given to return the component and not the list containing the component.

The `svalue` method is a convenience method that applies the `svalue` method to each component of the list returned by `\[`.

The `names` method is a convenience method that gives the names of the widgets store in the list returned by `\[`.

Note

The design of this borrows from the `FormPanel` and `FormLayout` constructors in the extjs.com library for javascript programming.

See Also

[ggenericwidget](#)

Examples

Not run:

```
## layout a collection of widgets to generate a t.test
tTest <- list(type = "ggroup",
             horizontal = FALSE,
             children = list(
               list(type="fieldset",
                   columns = 2,
                   label = "Variable(s)",
                   label.pos = "top",
                   label.font = c(weight="bold"),
                   children = list(
                     list(name = "x",
                          label = "x",
                          type = "gedit",
                          text = ""),
                     list(name = "y",
                          label = "y",
                          type = "gedit",
                          text = "",
                          depends.on = "x",
                          depends.FUN = function(value) nchar(value) > 0,
                          depends.signal = "addHandlerBlur"
                     )
                   )
             ),
             list(type = "fieldset",
                 label = "Hypotheses",
                 columns = 2,
                 children = list(
                   list(name = "mu",
                        type = "gedit",
```

```

        label = "Ho: mu=",
        text = "0",
        coerce.with = as.numeric),
      list(name = "alternative",
          type="gcombobox",
          label = "HA: ",
          items = c("two.sided", "less", "greater")
        )
      )
    ),
    list(type = "fieldset",
        label = "two sample test",
        columns = 2,
        depends.on = "y",
        depends.FUN = function(value) nchar(value) > 0,
        depends.signal = "addHandlerBlur",
        children = list(
          list(name = "paired",
              label = "paired samples",
              type = "gcombobox",
              items = c(FALSE, TRUE)
            ),
          list(name = "var.equal",
              label = "assume equal var",
              type = "gcombobox",
              items = c(FALSE, TRUE)
            )
        )
    ),
    list(type = "fieldset",
        columns = 1,
        children = list(
          list(name = "conf.level",
              label = "confidence level",
              type = "gedit",
              text = "0.95",
              coerce.with = as.numeric)
        )
    )
  )
)

w <- gwindow("t.test")
g <- ggroup(horizontal = FALSE, cont = w)
fl <- gformlayout(tTest, cont = g, expand=TRUE)
bg <- ggroup(cont = g)
addSpring(bg)
b <- gbutton("run t.test", cont = bg)
addHandlerChanged(b, function(h,...) {
  out <- svalue(fl)
  out$x <- svalue(out$x) # turn text string into numbers via get()
  if(out$y == "") {
    out$y <- out$paired <- NULL
  }
})

```

```

    } else {
      out$y <- svalue(out$y)
    }
    print(do.call("t.test", out))
  })
## End(Not run)

```

ggenericwidget *A constructor to create widgets for evaluating functions*

Description

This constructor creates a widget for collecting arguments for a function using a list to define the widget's components. When called with a function name a list is created on the fly which can be used as is, or modified as desired.

Usage

```
ggenericwidget(lst, cli = NULL, container = NULL, ..., toolkit = guiToolkit())
```

Arguments

| | |
|-----------|---|
| lst | Either a list defining the widget or a function name as a string. In the latter case, the defining list may be retrieved by the <code>svalue</code> method. |
| cli | An instance of <code>gcommandline</code> or <code>NULL</code> . If <code>NULL</code> , then a new command line pops up in its own window |
| container | Optional container to attach widget to |
| ... | Currently ignored by <code>ggenericwidget</code> , but passed along to <code>gedit</code> by <code>geditlist</code> and <code>geditnamedlist</code> |
| toolkit | Which GUI toolkit to use |

Details

This widget provides an easy way to create dialogs that collect the arguments for a function evaluation. When the OK button is clicked, the arguments are collected and passed along to the function specified via the `action` part of the list. When collecting the arguments, empty strings are not passed along.

The easiest usage is to simply provide a function name and have `autogenerategeneric` take a stab. However, in the long run it might be better to use `autogenerategeneric` to create an initial list, and then modify this to adjust the widget's look.

The list contains several named components

title The title for the widget

help What help page is called

type Either "text" or "graphic." Currently ignored.

variableType Describes the type of variable. Either "univariate", "univariatetable", "fleurl", "bivariate", "model", "lattice", "lmer" or NULL. This value is passed directly to `gvariables`. For non-NULL values, the widget shows an appropriate area for collecting the main variable. For the model and lattice interfaces buttons allow editing of fields by subsequent dialogs.

variableTypeExtras An optional list with components `name` and `value` containing a name and value passed along to the constructor for the variable type. Useful to override default

assignto If TRUE, creates box for collecting name for assigning output

action a list with named components `beginning` and `ending`. The arguments are collected and pasted together to form a string containing the R command to execute. These get put at the beginning and end of the string. A typical pair would be something like "prop.test(" and ")".

arguments a list with named components. In the simplest usage the names are argument names, and the components are lists with entries that create the corresponding widget. The first such component is called `type` and is the name of a `gWidget`, such as "gradio". Subsequent components are passed to this function using `do.call`.

The constructors `geditlist` and `geditnamedlist` can be used when the input is to be a list of values or a list of named values.

In the more complicated cases, these named components can be grouped into a list component. The name of this is then used to block the arguments. See the example.

The `svalue` method returns the value of the list. This can be used to retrieve the list that is created when the constructor is called with a function name.

Note

This function may be improved and the list defining it changed.

Examples

```
## Not run:
      ## a sample list definition
## Save some typing by defining a list to be used more than once later
TRUE.list <- list(
  type = "gradio",
  items = c("TRUE", "FALSE")
)

## define a list for producing a histogram widget
hist.list <- list(
  title = "hist()",
  help = "hist",
  action = list(
    beginning = "hist(",
    ending = ")"
  ),
  type = "graphic",
  variableType = "univariate",
  arguments = list(
    adjustments = list(
      breaks= list(
        type="gdroplist",
        # either text or graphic
        # single variable
      )
    )
  )
)
```

```

        items=c("\Sturges\","\Scott\","\Friedman-Diaconis\")
    ),
    probability = TRUE.list,
    include.lowest = TRUE.list,
    right = TRUE.list,
    shading = list(
        density = list(
            type="gedit",
            text=NULL
        ),
        angle = list(
            type="gedit",
            coerce.with="as.numeric",
            text="45"
        )
    )
)
)
)

ggenericwidget(hist.list, container=TRUE)
## or to autogenerate one
ggenericwidget("boxplot.default", container=TRUE)

## End(Not run)

```

ggraphics

Constructor for a toolkit specific plot device and a notebook to wrap plots in

Description

If a toolkit provides a graphics device, such as the `cairoDevice` package does for GTK, this constructor makes devices that can then be embedded in other widgets. The notebook interface is one such example.

Usage

```
ggraphics(width = dpi * 6, height = dpi * 6, dpi = 75, ps = 12, container =
NULL, ..., toolkit = guiToolkit())
```

```
ggraphicsnotebook(width=dpi*6, height=dpi*6,dpi=75, container = NULL, ..., toolkit =
```

Arguments

| | |
|---------------------|---|
| <code>width</code> | width in pixels of device |
| <code>height</code> | height in pixels of device |
| <code>dpi</code> | scale factor for default width and height |

| | |
|-----------|--|
| ps | pointsize |
| container | Optional container to attach widget to |
| ... | Passed to add method of container. |
| toolkit | Which GUI toolkit to use |

Details

When multiple graphics devices are present, clicking in the window of one will make that the current device

The `visible<-` method makes the object the current device.

The `svalue(obj, ..., value)` method will save the visible window to the file in `value`. In `gWidgetsRGtk2`, if the window has another window clipping part of it, this clipping will be shown. This "hack" is needed, as `dev.copy` does not currently work for the "cairo" graphic device. (In future versions, there will be support for pdf files within cairo.)

The `addhandlerclicked(obj, handler, action, ...)` method where `handler` has first argument `h` has the additional values `h$x` and `h$y` where these are values are returned using "usr" coordinates (see `help("par")`). (This was in NDC coordinates)

Examples

```
## Not run:
win <- gwindow("Graphics example")
gggraphics(ps=6, container=win)
hist(rnorm(100))
## End(Not run)
```

gggroup

Containers for packing in subsequent widgets

Description

Various box containers useful for laying out GUI controls. These containers pack in child widgets from left to right or top to bottom. A few arguments can be used to adjust the sizing and positioning.

Usage

```
gggroup(horizontal = TRUE, spacing = 5, use.scrollwindow = FALSE, container = NULL,

gframe(text = "", markup = FALSE, pos = 0, horizontal=TRUE, container = NULL,
..., toolkit = guiToolkit())
gexpandgroup(text = "", markup = FALSE, horizontal=TRUE, handler = NULL, action = N
container = NULL, ..., toolkit = guiToolkit())
```

Arguments

| | |
|-------------------------------|--|
| <code>horizontal</code> | Specifies if widgets are packed in left to right or top to bottom (FALSE) |
| <code>spacing</code> | Space in pixels around each widget. Can be changed with <code>svalue</code> |
| <code>text</code> | Text for label |
| <code>markup</code> | Optional markup. (See <code>glabel</code> for details.) |
| <code>pos</code> | Where to place label: 0 is to left, 1 to right, interpolates. |
| <code>handler</code> | Called when expand arrow is clicked |
| <code>action</code> | Passed to handler |
| <code>use.scrollwindow</code> | If TRUE then <code>group</code> is placed in a scrollwindow allowing panning with mouse. |
| <code>container</code> | Optional container to attach widget to. Not optional for <code>gWidgetstcltk</code> , or <code>gWidgetsRwxwidgets</code> |
| <code>...</code> | Passed to the <code>add</code> method of the container |
| <code>toolkit</code> | Which GUI toolkit to use |

Details

A `ggroup` is the primary container for packing in subsequent widgets, either from left to right or top to bottom. Widgets are packed in using the `add` method and can be removed with the `delete` method.

The `gframe` container adds a decorative border and optional label to the box container.

The `gexpandgroup` containers has an optional label and a trigger to click on which toggles the display of the the child widgets.

The containers pack in child widgets from either left to right (when `horizontal=TRUE`) or from top to bottom (when `horizontal=FALSE`).

Child widgets are added to box containers through their `add` method or through the containers `use` as the parent container when a widget is constructed. This is done by using the `container` argument of the widget. The `container` argument is optional for `gWidgetsRGtk2` and `gWidgetsrJava`, but not the other toolkits. It is suggested that it always be included for portability. When it is not included, widgets are added to the new `group` object through its `add` method. Otherwise, when a widget is created, the `group` is specified as the container and the `add` method is then called.

When the parent allocates space to a child widget is may be more than the space needed by the child widget. Different types of behavior are desirable for drawing the child. To have the widget grow to fill the space, use the argument `expand=TRUE`. (Growing is done in both x and y directions. To anchor the widget into a portion of the space provided, use the argument `anchor=c(a,b)` where `a` and `b` are values in -1,0,1 and specify the position using Cartesian coordinates.

The `svalue<-` method can be used to adjust the border width. By default it is 2 pixels.

The `spacing` value determines the number of pixels of padding between each widget when packed in. This can be set when the `group` is constructed, or later using `svalue<-`.

To put space between just two widgets, the `addSpace(obj, value, ...)` method may be used where `value` is the number of pixels of padding between the just packed widget, and the next one to be packed.

The `addSpring(obj, ...)` method will push the just packed widget and the next-to-be packed widget as far apart as possible.

For `ggroup`, in `gWidgetsSRGtk2` a few arguments add to the container. The argument `raise.on.dragmotion = TRUE` will cause the group to jump to the foreground when a drag action crosses it. The argument `use.scrollwindow = TRUE` will put the group

For `gframe` and `gexpandgroup` the label name can be retrieved or adjusted with the `names` method.

For `gexpandgroup` the `visible` method can be used to toggle the display programmatically.

See Also

For top-level containers [gwindow](#), for containers to display more than one child widget see [gpanedgroup](#), [gnotebook](#), [glayout](#)

Examples

```
## Not run:
## basic group
group <- ggroup(horizontal=FALSE, container=gwindow())
l <- glabel("widget 1") ## not in gWidgetstcltk -- needs a container
add(group, l)
glabel("widget 2", container = group) ## style for all toolkits

## nested groups
group <- ggroup(horizontal=FALSE, container=gwindow())
innergroup <- ggroup(container = group)
gtext("Text area", container=group)
gbutton("button 1", container = innergroup)
gbutton("button 2", container = innergroup)

## expand argument
group <- ggroup(horizontal=FALSE, container=gwindow())
gbutton("no expand", container=group)
gbutton("expand=TRUE", container=group, expand=TRUE)

## anchor argument
w <- gwindow("Anchor")
size(w) <- c(500,500)
group <- ggroup(container=w)
glabel("upper left", container=group, anchor=c(-1,1))
glabel("lower right", container=group, anchor=c(1,-1))

## add spring
group <- ggroup(container=gwindow("menubar-like example"))
gbutton("File", handler=function(h,...) print("file"), container=group)
gbutton("Edit", handler=function(h,...) print("edit"), container=group)
addSpring(group)
gbutton("Help", handler=function(h,...) print("help"), container=group)

## End(Not run)
```

 ghelp

Widget to interface with help pages

Description

A widget to interface with the help pages and that widget placed in a browser. The widget is a notebook capable of showing several pages at once.

Usage

```
ghelp(topic = NULL, package = NULL, container = NULL, ..., toolkit = guiToolkit())
ghelpbrowser(title = "Help browser", maxTerms=100, width=550, height=600, ..., tool
```

Arguments

| | |
|-----------|--|
| topic | Help topic |
| package | Which package to look for topic in |
| container | Optional container to attach widget to |
| title | Title of help browser |
| maxTerms | Maximum number of search responses |
| width | Width of browser window in pixels |
| height | Height of browser window in pixels |
| ... | Passed to add method of container |
| toolkit | Which GUI toolkit to use |

Details

Finding the help page can be a little slow.

This does not work with chm files on windows. However, the help browser in that environment is far superior than this, so no incentive exists to implement that.

The `add(obj, value)` method can be used to add a new page. The page may be specified to `value` in different ways: as a string containing the `topic`, as a list with components `topic` and `package`, or as a string in the form `package::topic`.

The `ghelpbrowser` constructor produces a stand alone GUI for browsing help pages, running the corresponding examples and viewing a packages vignettes. Unlike other **gWidgets** constructors, this has no `container` argument.

Examples

```
## Not run:
  obj <- ghelp(container=TRUE)
  add(obj, "base:::mean")

## End(Not run)
```

ghtml

Constructors for widgets to handle text input

Description

The ghtml widget is intended to show HTML text either from a url or from a character string. Currently no toolkits support this widget, although it is in gWidgetsWWW.

Usage

```
ghtml(x, handler = NULL, action = NULL, container = NULL, ..., toolkit = guiToolkit)
```

Arguments

| | |
|-----------|---|
| x | url or HTML-marked up character string to load into widget. |
| handler | Handles a click on a URL. The default is to open the clicked url in the widget. To override, the first argument, a list h, has component h\$url containing the url. |
| action | Passed along to the handler as h\$action |
| container | Optional container to attach widget to |
| ... | Passed to add method of container |
| toolkit | Which GUI toolkit to use |

Details

This widget loads the given url into a widget. Currently no toolkits support this.

The svalue method returns the current url or character string.

The svalue<- method loads the current url or character string in the widget.

Examples

```
## Not run:
  ghtml(system.file("html", "gedit.html", package="gWidgets"),
        container = gwindow())
## End(Not run)
```

gimage

*Constructor to show images***Description**

This constructor produces a widget intended to show images stored as files on the file system.

Usage

```
gimage(filename = "", dirname = "", size = "",
        handler = NULL, action = NULL, container = NULL, ..., toolkit = guiToolkit())
```

Arguments

| | |
|-----------|--|
| filename | Specifies location of image. May be a stock icon name or filename. (In the future may be a url.) |
| dirname | Directory of file. If "stock", then a stock icon is used. |
| size | Size of image when stock image is used. Values are in c("menu", "small_toolbar", "large_t |
| handler | Handler called on a click event |
| action | Passed to handler |
| container | Optional container to attach widget to. |
| ... | Passed to add method of container |
| toolkit | Which GUI toolkit to use |

Details

The `svalue()` method returns the filename of the figure or the stock icon name, if the icon was set from a stock icon.

The `svalue<-()` method can be used to set the value of the widget. The value is a filename containing the image to display.

The `addhandlerclicked` method is called on click events.

See Also

See [getStockIcons](#) to get a list of available icons and [addStockIcons](#) to add to this list.

Examples

```
## Not run:
w <- gwindow("Stock icon example")
gimage("ok",dirname="stock", cont = w)
## End(Not run)
```

glabel

*Constructors for label widget***Description**

This constructor produces a widget to display a line or multiline lines of text.

For some toolkits, the text can be marked up.

An option is available so that the displayed text can be edited.

Usage

```
glabel(text = "", markup = FALSE, editable = FALSE, handler = NULL,
       action = NULL, container = NULL, ..., toolkit = guiToolkit())
```

Arguments

| | |
|-----------|--|
| text | Text to show in the label or button. For buttons, if this text matches a stock icon name, an icon is shown as well |
| markup | Logical indicating if text for a label uses markup |
| editable | Logical. If TRUE, then the label's text can be set by clicking on the label and filling in the edit box. |
| handler | Handler called on a click event |
| action | Passed to handler |
| container | Optional container to attach widget to. |
| ... | Passed to add method of container |
| toolkit | Which GUI toolkit to use |

Details

The `svalue()` method returns the value of the widget. For a label, this is the text as a single string (which may not include a "\n" for newlines if not supported by the toolkit).

The `svalue<-()` method can be used to set the value of the widget. For labels and buttons, value with length greater than one are pasted together collapsed with "\n".

The `addhandlerclicked` method specifies a handler to be called on click events.

Although in some toolkits, labels are meant to hold static text, gWidgets treats label widgets like other widgets allowing the user to bind handlers to mouse clicks. For labels, if `editable=TRUE` is specified, clicking on the text allows one to edit the label's value overriding the click handler in the process. However, the `addhandlerchanged` handler can be given to respond to the text after it has been changed.

Examples

```
## Not run:
  glabel("a label", container=TRUE)
  glabel("Click me to edit label", editable=TRUE, container=TRUE)
  glabel("Click me for a message", container=TRUE,
        handler=function(h,...) {cat("Hi\n")})
## End(Not run)
```

glayout

A container for aligning widgets in a table

Description

A container for laying out widgets in a table. The widgets are added using matrix notation (`[i, j] <-` `widget`).

Usage

```
glayout(homogeneous = FALSE, spacing = 10, container = NULL, ..., toolkit = guiTool)
```

Arguments

| | |
|--------------------------|---|
| <code>homogeneous</code> | A logical indicating if the cells are all the same size |
| <code>spacing</code> | Spacing in pixels between cells |
| <code>container</code> | Optional container to attach widget to. |
| <code>...</code> | Passed to add method of container |
| <code>toolkit</code> | Which GUI toolkit to use |

Details

Widgets are added using matrix notation. A widget can span several cells, for instance `obj[1:2, 2:3] <- widget` would place the widget in the first and second rows and second and third columns. The matrix notation is to specify the space allocated to the widget. There is no corresponding extraction method via `[.]`.

For `gWidgetstcltk`, it is necessary for a child widget to have the layout object as its parent container and to call the `[<-` method to add the widget. (See the example.)

As a convenience, if the value to be assigned is a character it will be turned into a `glabel` object before being added.

Like `ggroup`, the extra argument `expand` can be used to force the widget to expand to fill all the space allocated to it.

Like `ggroup`, the extra argument `anchor` can be used to anchor the child within the space allocated when this space is larger than needed by the widget. This is specified as a pair of values from `-1,0,1` to indicating the `x` and `y` positioning of the widget within the cell.

Examples

```
## Not run:
## show part of mtcars dataframe in a layout
w <- gwindow("glayout example")
tbl <- glayout(container = w)
tbl[1,1] <- "a label"
## need container argument in gWidgetstcltk, gWidgetsRwxwidgets
## so we always use it.
tbl[1,2, expand = TRUE] <- gedit("edit here", container=tbl)
tbl[2,1, anchor = c(-1,-1)] <- glabel("ll", container = tbl)
## End(Not run)
```

gmenu

*Constructors to make menubar or toolbars***Description**

A menubar or toolbar are created using these constructors. These are specified using a lists, and these may be seen as simply mapping these lists into the corresponding widget.

Usage

```
gmenu(menulist, popup = FALSE, action=NULL, container = NULL, ..., toolkit = guiTk)

gtoolbar (toolbarlist, style = c("both", "icons", "text",
  "both-horiz"),
  action=NULL, container = NULL,
  ..., toolkit = guiToolkit())

gaction(label, tooltip = NULL, icon = NULL, key.accel = NULL,
  handler = NULL, action = NULL, ...,
  toolkit = guiToolkit())
```

Arguments

| | |
|-------------|--|
| menulist | A list defining a menu bar |
| popup | Logical indicating if this should return a popup menu |
| toolbarlist | A list defining a toolbar |
| style | What style to use |
| action | Passed to menubar handlers |
| container | Container to attach widget to. Should be a gwindow instance. |
| label | Label for action item |
| tooltip | tooltip |
| icon | icon to decorate instance of action |

| | |
|------------------------|--|
| <code>key.accel</code> | keyboard accelerator |
| <code>handler</code> | Handler called when object attached to action is activated |
| <code>...</code> | Passed to the <code>add</code> method of the container |
| <code>toolkit</code> | Which GUI toolkit to use |

Details

The `gaction` constructor creates reusable objects for use with buttons, menubars and toolbars. Once constructed, the main methods are `svalue`, and `svalue<-` for getting and setting the label text and `enabled<-` which can changes whether the widgets depending on the action are sensitive to user input. An action object contains a label, an optional icon, an optional keyboard accelerator, and a handler. The handler does not have the widget from which it is called passed in to the `obj` component of the first argument, but one can parametrize the argument with the `action` argument. The icon, tooltip, and keyboard accelerator are very much toolkit and OS dependent, and so may not always be available by the widget using the `gaction` object. The keyboard accelerator is not yet implemented in any toolkit.

The lists defining a menubar or toolbar are very similar.

Each is a list with *named* components. A component is a terminal node if it a) is a `gaction` instance or b) has a `handler` component, which is a function to be called (without arguments) when the menu item or toolbar item is selected. Optionally, an `icon` component can be given specifying a stock icon to accompany the text. A non-null component named `separator` will also indicate a terminal node. In this case, a visible separator will be displayed.

A menubar list can use the hierarchical nature of a list to generate submenus. For toolbars this is not the case.

These constructors map the list into the widget. The methods for the constructors refer to these list defining the widget.

The `svalue` method returns the list.

The `svalue<-` method can be used to change the list, and hence redo the menubar.

The `"["` method refers to the components of the list.

The `"[<-"` method can be used to change pieces of the menubar or toolbar.

The `add` method with signature `(obj, lst)` or `(obj, gmenu.instance)` can be used to apped to the current menubar/toolbar. The second argument is a list or an `gmenu` or `gtoolbar` instance.

The `delete` method can be used to delete part of the menubar/toolbar. The `value` argument can be either a character vector with the top-level names to delete, or a named list, or an instance of either `gmenu` or `gtoolbar`.

Popular usage reserves toolbars and menubars for top-level windows – not dialog sub windows, or sub groups within a GUI – as such, the container, specified at construction, should be a top-level `gwindow` instance

Examples

```
## Not run:
mbl <- list()
mbl$File$Open$handler = function(h,...) print("open")
```

```

mbl$File$Quit$handler = function(h,...) print("quit")
mbl$File$Quit$icon = "quit"
mbl$Edit$Find$handler = function(h,...) print("Find")
mbl$Edit$Replace$handler = function(h,...) print("Replace")

w <- gwindow("gmenu test")
mb <- gmenu(mbl, container=w)

tbl <- list()
tbl$New <- list(icon="new",handler = function(...) print("new"))
tbl$Print <- list(icon="print",handler = function(...) print("print"))

tb <- gtoolbar(tbl, container=w)

## example of using gaction
## works in gWidgetstcltk, but much better in gWidgetsRGtk2

## stub for handler
f <- function(h,...) print("stub")

## some actions. The icon is optional, as is tooltip
aOpen <- gaction(label="Open", icon="open", handler=f)
aClose <- gaction(label="Close", icon="close", handler=f)
aQuit <- gaction(label="Quit", icon="quit", handler=function(h,...) dispose(w))
aCut <- gaction(label="Cut", icon="cut", handler=f)
aCopy <- gaction(label="Copy", icon="copy", handler=f)
aPaste <- gaction(label="Paste", icon="paste", handler=f)

## set up groups of actions so that they can be disabled/enabled
## all at once
allActionsGroup <- list(aOpen, aClose, aQuit, aCut, aCopy, aPaste)
editActionsGroup <- list(aCut, aCopy, aPaste)

## define menubar list
ml <- list(File = list(
  open = aOpen,
  close = aClose,
  sep = list(separator = TRUE), # must be named component
  quit = aQuit),
  Edit = list(
    copy = aCopy,
    paste = aPaste))

## toolbar list has only one level
tl <- list(
  Open=aOpen,
  sep = list(separator = TRUE), # must be named component
  Quit = aQuit)

## set up main window
w <- gwindow()
gmenu(ml, cont = w)
gtoolbar(tl, cont = w)

```

```
## Now add a widget
gbutton(action = aQuit, cont = w)

## disable a group of action
sapply(editActionsGroup, function(i) enabled(i) <- FALSE)

## End(Not run)
```

gnotebook *constructor for notebook widget*

Description

A notebook widget organizes different pages using tabs, allowing only one page to be shown at once. Clicking on the tab raises the associated page.

Usage

```
gnotebook(tab.pos = 3, closebuttons = FALSE, dontCloseThese = NULL, container = NU
```

Arguments

| | |
|-----------------------------|---|
| <code>tab.pos</code> | Where to place tabs (1 bottom, 2 left side, 3 top, 4 right side) |
| <code>closebuttons</code> | Is there a close button in the tab? |
| <code>dontCloseThese</code> | If <code>closebuttons=TRUE</code> this will make it impossible to remove these tabs. Specified by tab number |
| <code>container</code> | Optional parent container to attach notebook widget to |
| <code>...</code> | passed to add method of parent container |
| <code>toolkit</code> | Which GUI toolkit to use |

Details

In what follows, it is useful to think of a notebook as a vector with named entries, each entry being a widget, the name being the tab label.

Notebooks have the following methods:

New pages are added with the `add` method. The extra argument `label` is used to specify the tab label. This may be a string, or in `gWidgetsRGtk2` a `glabel` instance. The extra argument `index` can be used to specify which page to add to. By default, a new page is created at the end of the notebook. In `gWidgetsRGtk2`, the extra argument `override.closebutton` can be used to add or not add a close button in the tab label.

The `dispose` method will remove the currently selected page unless it is overridden by the value of `dontCloseThese`.

The `svalue` method returns the current page number. The `svalue<-` method is used to set the page number.

The `delete(obj, widget, ...)` method will delete the widget on a given page. This can then be replaced with the `add` method.

The `length` method returns the number of pages.

The `names` method returns the tab labels.

The `names<-` method may be used to replace the tab labels. Something like `names(obj)[1]<-"new label"` should work.

See Also

See [gwindow](#) for top-level containers, [gggroup](#), [gframe](#) and [gexpandgroup](#) for box containers

Examples

```
## Not run:
w <- gwindow("gnotebook example")
nb <- gnotebook(container=w)
glabel("Widget 1", cont=nb, label="page 1")
glabel("Widget 2", cont=nb, label="page 2")

length(nb)
names(nb)
names(nb)[1] <- "Page 1"

svalue(nb) <- 2
dispose(nb) ## dispose current tab
length(nb)
## End(Not run)
```

| | |
|-------------|--|
| gpanedgroup | <i>A paned group holds two child components with a handle, or sash, between them to adjust the amount of space allocated to each</i> |
|-------------|--|

Description

A constructor for a paned group.

Usage

```
gpanedgroup(widget1=NULL, widget2=NULL, horizontal = TRUE, container = NULL, ..., t
```

Arguments

| | |
|------------|---|
| widget1 | Left (top) widget. Can be added at time of construction, or the <code>add</code> method can be used to add the child widgets one at a time. |
| widget2 | Right (bottom) widget |
| horizontal | Left/right (TRUE) or top/bottom |

| | |
|-----------|--|
| container | Optional container to attach widget to |
| ... | Passed to add method of container |
| toolkit | Which GUI toolkit to use |

Details

The add method can be used to one child at a time.

The svalue method returns the sash position with a value between 0 and 1.

The svalue<- method can be used to specify the sash position with a value between 0 and 1.

Examples

```
## Not run:
w <- gwindow("gpanedgroup example")
pg <- gpanedgroup(cont=w)
gvarbrowser(cont = pg) ## first is left/top
gtext(cont = pg)
svalue(pg) <- 0.25
## End(Not run)
```

gradio

Radio button group widget

Description

A radio group allows the user to select one value from a set of items. The items may be displayed horizontally or vertically.

Usage

```
gradio(items, selected = 1, horizontal = FALSE, handler
= NULL, action = NULL, container = NULL, ..., toolkit = guiToolkit())
```

Arguments

| | |
|------------|---|
| items | Vector of values to select from |
| selected | For gradio the initial selected value (as an index) For a drop list, the first selected value. Use 0 to leave blank |
| horizontal | A logical specifying the layout for gradio and gcheckboxgroup |
| handler | Called when selection is changed |
| action | Passed to handler when called. |
| container | Optional container to attach widget to |
| ... | Passed to add method of container |
| toolkit | Which GUI toolkit to use |

Details

The `svalue` method returns the selected value by name. If the extra argument `index=TRUE` is specified, the indices of the selected value is given.

The `svalue<-` method can be used to set the selected value. One can specify the value by name or by index if `index=TRUE` is specified.

The `"["` method refers to the vector defining the items.

The `"[<-"` method can be used to change the vector defining the items. The length will most likely need to be the same.

The `"length"` method returns the number of items.

See Also

The radio group is one of several widgets useful to selecting a value or values from a set of items. See also [gcheckbox](#), [gcheckboxgroup](#), [gcombobox](#), and [gtable](#)

Methods for `GComponent` objects are detailed in [gWidgets-methods](#).

Event Handlers are detailed in [gWidgets-handlers](#).

Examples

```
## Not run:
  flavors <- c("vanilla", "chocolate", "strawberry")

  f <- function(h,...) print(
    paste("Yum",
          paste(svalue(h$obj),collapse=" and "),
          sep = " "))

  w <- gwindow("Radio example")
  gp <- ggroup(container=w)
  glabel("Favorite flavor:",cont=gp)
  rb <- gradio(flavors, cont=gp, handler = f)

  w <- gwindow("combobox example")
  gp <- ggroup(container=w)
  glabel("Favorite flavor:", cont=gp)
  gcombobox(flavors, editable=TRUE, cont=gp, handler=f)

  w <- gwindow("checkbox example")
  gp <- ggroup(container=w)
  glabel("Favorite flavors:",cont=gp)
  cbg <- gcheckboxgroup(flavors, cont=gp, handler=f)

  svalue(cbg) <- c(TRUE, FALSE, TRUE)
  svalue(cbg)
  cbg[3] <- "raspberry"

## End(Not run)
```

| | |
|------------|--------------------------------------|
| gseparator | <i>Widget to draw separator line</i> |
|------------|--------------------------------------|

Description

Simple widget to draw a line used clarify layout of widgets.

Usage

```
gseparator(horizontal = TRUE, container = NULL, ..., toolkit = guiToolkit())
```

Arguments

| | |
|------------|---|
| horizontal | If TRUE line is horizontal, otherwise vertical. |
| container | Optional container to attach widget to |
| ... | Ignored |
| toolkit | Which GUI toolkit to use |

Examples

```
## Not run:
w <- gwindow("separator example")
gp <- ggroup(container=w)
glabel("left widget", cont=gp)
gseparator(horizontal=FALSE, cont=gp, expand=TRUE)
glabel("right widget", cont=gp)
## End(Not run)
```

| | |
|---------|--|
| gslider | <i>Constructors for widgets to select a value from a sequence.</i> |
|---------|--|

Description

The gslider widget and gspinbutton widget allow the user to select a value from a sequence using the mouse. In the slider case, a slider is dragged left or right (or up or down) to change the value. For a spin button a text box with arrows beside allow the user to scroll through the values by clicking the arrows.

Some toolkits only allow integer values for these.

Usage

```
gslider(from = 0, to = 100, by = 1, value = from, horizontal = TRUE, handler = NULL)
```

```
gspinbutton (from = 0, to = 10, by = 1, value = from, digits = 0,
  handler = NULL, action = NULL, container = NULL, ..., toolkit = guiToolkit())
```

Arguments

| | |
|------------|--|
| from | Starting point in range |
| to | Ending point in range |
| by | Step size between values in the sequence |
| value | The initial value |
| digits | The number of digits shown |
| horizontal | Specifies orientation of gslider widget |
| handler | Called on a change event. |
| action | Passed to handler |
| container | Optional container to attach widget to |
| ... | Passed to add method of container |
| toolkit | Which GUI toolkit to use |

Details

The `svalue` method returns the selected value.

The `svalue<-` method is used to set the selected value.

The `[<-` method can be used to change the sequence that the value is selected from. It expects a regular sequence.

The `addhandlerchanged` handler is called when the widgets' value is changed.

Examples

```
## Not run:
x <- rnorm(100)

## our handler
plotHist <- function(...)
  hist(x, col=gray(svalue(sb)), breaks = svalue(sl))

w <- gwindow("Slider and spinbox example")
tbl = glayout(cont=w)
tbl[1,1] <- "Slide value to adjust shade"
tbl[1,2] <- (sb <- gspinbutton(from=0,to=1,by=0.05,value=.5, container=tbl,
  handler=plotHist))
tbl[2,1] <- "No. breaks"
tbl[2,2, expand=TRUE] <- (sl <- gslider(from = 1, to= 100, by=1, value = 10,
  cont = tbl, handler = plotHist))

## update slider using [<-
sl[] <- seq(2,50, by=2)
## End(Not run)
```

gstatusbar *Constructor of status bar widget*

Description

A status bar widget is used to send message to the user. A familiar instance is the bottom area of a web browser.

Usage

```
gstatusbar(text = "", container = NULL, ..., toolkit = guiToolkit())
```

Arguments

| | |
|-----------|--|
| text | Initial text of status bar |
| container | Optional container to attach widget to. Should be gwindow object |
| ... | Passed to add method of container |
| toolkit | Which GUI toolkit to use |

Details

The statusbar keeps a message stack. The `svalue` method pops the last message from the stack. The `svalue<-` method pushes a new message onto the stack.

Statusbars should be added to the top-level gwindow instance.

Examples

```
## Not run:
w <- gwindow("status bar example")
tbl <- list(quit=list(icon="quit",
  handler = function(...) dispose(w)))
tb <- gtoolbar(tbl, cont=w)
sb <- gstatusbar("", cont=w)
txt <- gtext("type here", cont=w)
addHandlerChanged(txt, handler=function(h,...)
  svalue(sb) <- paste("You typed",svalue(txt),"in the box",collapse=" "))
## End(Not run)
```

| | |
|--------|---|
| gtable | <i>Constructor for widget to display tabular data</i> |
|--------|---|

Description

This widget displays either a vector, matrix or data frame in a tabular format. The main usage is for user selection of a row or rows.

Usage

```
gtable(items, multiple = FALSE, chosencol = 1, icon.FUN = NULL, filter.column = NULL)
```

Arguments

| | |
|---------------|--|
| items | A vector, matrix or data frame to be displayed. A vector and matrix is coerced into a data frame. |
| multiple | A logical. If TRUE multiple lines can be selected |
| chosencol | By default, only the value in this column is returned by the <code>svalue</code> method. |
| icon.FUN | If given, this function is applied to the data frame to be shown. It should return a vector of stock icon names |
| filter.column | If not NULL a filter by droplist is given which can be used to filter the displayed values shown using the values in this column. |
| filter.labels | If filtering is desired, but by a more complicated means that just a columns value, then these labels are used for the drop list. These are used by <code>filter.FUN</code> |
| filter.FUN | Either a function with signature <code>(obj, filter.by)</code> to specify a vector of logical values indicating which rows should be shown or the character "manual" in which filtering is done directly through the <code>visible</code> method and not through a popup box |
| handler | Called on a double click event |
| action | Passed to handler |
| container | Optional container to attach widget to |
| ... | ignored |
| toolkit | Which GUI toolkit to use |

Details

The `svalue` method returns the selected value(s). By default, only the value(s) in the chosen column are returned. Use the argument `drop=FALSE` to return the entire row. To return the row index, use the argument `index=TRUE`. This index refers to the whole data store, not just the visible portion when filtering is being used.

The "[" notion treats the object like a data frame. When filtering, this notation refers to the entire data frame, not the visible data frame. The comment about the returned index by `svalue` can be

described by the fact that `obj[svalue(obj, index=TRUE),]` should be the same value as `svalue(obj)`.

Assignment via "`[<-`" is possible with limitations imposed by the toolkits. The graphical display of tabular matter is usually done, similar to data frames, in terms of columns each having the same type. For some toolkits, all values are converted to characters, for others, the type must be maintained. In R, coercion of types may occur when assigning to a data frame, but this won't be so with the underlying toolkit widget. To be portable across toolkits, the column type should not change during assignment, nor should the number of rows be reduced.

In particular, assignment with "`[<-`" for factors can cause warnings if the values are not in the factor's levels. When the value being assigned is a matrix there is a coercion to a data frame which may change the type.

The `visible` and `visible<-` methods refer to which rows of the data store are visible in the widget. These are specified by a vector of class logical. This may be used when there is filtering, not sorting. The value returned by `svalue` is a logical vector of length given by the number of rows of the data store, with `TRUE` indicating that the row is displayed. When setting the visibility of a row through `svalue` the vector of values should have the same length as the number of rows, otherwise recycling occurs. (So `visible(obj) <- TRUE` will work to display all the rows.)

The `length` method returns the length of the underlying data store.

The `dim` method returns the dimension of the underlying data store.

The `names` method returns the names of the underlying data store.

The `names<-` method can be used to set the names of the underlying data store and the values displayed in the column headers of the widget.

Row names are ignored in the display of this widget.

A single click is used for selection of a value.

The `addHandlerDoubleClick` handler can be used to define a callback to respond to a double click event.

See Also

See also [gtree](#) for displaying tree-like data and [gdf](#) for tabular data meant to be edited

Examples

```
## Not run:
## example to select CRAN mirror
m <- getCRANmirrors()[,c(1,4)]
setCRAN <- function(URL) { ## see chooseCRANmirror
  repos = getOption("repos")
  repos["CRAN"] <- gsub("/$", "", URL)
  options(repos=repos)
}
w <- gwindow("gtable example",width=400)
gp <- ggroup(horizontal=FALSE, cont=w)
tab <- gtable(m, chosencol = 2, cont=gp, expand=TRUE,
  handler = function(h,...) setCRAN(svalue(h$obj)))
bg <- ggroup(cont=gp)
addSpring(bg)
```

```

gbutton("dismiss", cont=bg, handler = function(h,...) dispose(w))

## an example with icons.
## Select variables from a data frame

## find icons by class
icon.FUN <- function(items) {
  dfName <- svalue(cb)
  df <- try(get(dfName, envir=.GlobalEnv), silent=TRUE)
  if(inherits(df,"try-error"))
    return(rep(NULL,dim(items)[1]))
  if(is.data.frame(items))
    items <- items[,1, drop=TRUE]
  sapply(items, function(i) {
    class(df[,i])[1]
  })
}
## list data frames in an environment
lsDF <- function(envir=.GlobalEnv) {
  varNames <- ls(envir=envir)
  dfs <- sapply(varNames, function(i) inherits(get(i,envir=envir),"data.frame"))
  varNames[dfs]
}
## set up GUI
w <- gwindow("Select variables",width=250)
g <- ggroup(horizontal=FALSE, cont=w)
l <- glabel("Data frame", cont=g)
cb <- gcombobox(lsDF(), cont=g)
blankDF = data.frame(variables=character(0), stringsAsFactors=FALSE)
tbl <- gtable(blankDF, icon.FUN=icon.FUN, cont=g, expand=TRUE)
## add handlers
addHandlerChanged(cb, handler <- function(h,...) {
  dfName <- svalue(h$obj)
  dfNames <- names(get(dfName,envir=.GlobalEnv))
  tbl[,] <- data.frame(variables=dfNames, stringsAsFactors=FALSE)
})
addHandlerClicked(tbl, handler = function(h,...) {
  cat("Do something with",svalue(cb),"::", svalue(h$obj),"\n")
})

## End(Not run)

```

gtext

Constructor for widget for multi-line editable text input

Description

The `gtext` widget creates a text buffer for handling multiple lines of text.

Usage

```
gtext (text = NULL, width = NULL, height = 300, font.attr = NULL,
      wrap = TRUE, handler = NULL, action = NULL, container = NULL,
      ..., toolkit = guiToolkit())
```

Arguments

| | |
|-----------|---|
| text | Initial text in widget |
| width | Width of widget in pixels |
| height | Height of gtext widget in pixels |
| font.attr | Optional specification of font attributes |
| wrap | For gtext, are long lines wrapped? |
| handler | Handler called when text is changed. |
| action | Passed to handler |
| container | Optional container to attach widget to |
| ... | Passed to add method of container |
| toolkit | Which GUI toolkit to use |

Details

The `gtext` widget has the following methods.

The `svalue` method returns the text held in the buffer. If `drop=TRUE`, then only the text in the buffer selected by the mouse is returned.

The `svalue<-` method replaces the text in the buffer with the new text.

New text is added with the `insert` method. The basic usage is `insert(obj, text)` where "text" could be a single line or a vector of text, or –for `gWidgetsRGtk2` – a `gwidget` (although some, like `gedit`, are kind of flaky). Extra arguments include `do.newline` a logical indicating if a new line after the last line should be added (default is `TRUE`); `font.attr` to specify any font attributes; `where` indicating where to add the text (either `end` or `beginning`). The `insert` generic replaces the overused `add` for `gtext`, but `add` will still work.

The font can be changed. The `font.attr` argument to the constructor and to `add` specifies fonts using a named character vector. For instance `c(style="normal", weights="bold", sizes="medium")`. The command `obj[['tags']]` will produce a list containing all the available attributes.

The `font<-` method is used to change the font of the currently selected text. It too takes a named character vector specifying the font attributes.

The `dispose` method clears the text in the buffer.

The `addHandlerKeystroke` method for `gedit` and `gtext` is called for each keystroke. In `gtext` or `RGtk2` the component `key` of the `h` argument contains the keystroke.

Examples

```
## Not run:
  ## gtext example
  obj <- gtext("First line", container=gwindow())
  insert(obj,"second line", font.attr=c(family="monospace"))
  insert(obj,"third line", font.attr=c(foreground.colors="red"))
## End(Not run)
```

 gtree

Constructor for widget to display heirarchical data

Description

This widget allows tree-like data to be presented. Each node on the tree should be a data frame with the same column structure. The first column is treated like a key, and should be unique. Offspring are specified through a function of the keys which are ancestors. This function returns the data frame to be displayed. Values in the tree can be selected with the mouse. This value can be retrieved through a method, or a handler can be assigned to double click events.

Usage

```
gtree(offspring = NULL, hasOffspring = NULL, offspring.data = NULL,
      col.types = NULL, icon.FUN = NULL, chosen.col = 1, multiple = FALSE,
      handler = NULL, action = NULL, container = NULL, ..., toolkit = guiToolkit())
```

Arguments

- | | |
|----------------|---|
| offspring | A function to produce a data frame. The first column of the data frame is used as a key. It should be unique, otherwise the updating will not work properly. The <code>offspring</code> function has two arguments, the first being the path (the first column of the offspring data frame is the key, and the path is the vector of keys) and the value of <code>offspring.data</code> . The data frame can determine whether an entry has offspring, by having the second column be a logical vector, <code>TRUE</code> if there is offspring, <code>FALSE</code> if not. |
| hasOffspring | Whether an entry has an offspring is determined by a) if this function is non- <code>NULL</code> and it returns a <code>TRUE</code> value when called on the offspring data frame for this row, b) if the second column of the offspring data frame is a logical vector and for this row is <code>TRUE</code> . If this function is <code>NULL</code> and the second column is not a logical vector then it is assumed that there are no offspring. |
| offspring.data | Passed to <code>offspring</code> function to parameterize that function. |
| col.types | Used to determine the type of column, given as a data frame with 1 or more rows. Otherwise it is determined by first row of offspring data frame. If this can return an empty data frame, then this argument should be given. |

| | |
|------------------------|--|
| <code>icon.FUN</code> | An optional function to determine an icon place into the first column. This function gets called with the data in <code>offspring</code> , and should return a row vector of length <code>nrow(offspring)</code> . The icons are stock icons, and should be referenced by name. The helper function <code>getStockIcons</code> list all the available stock icons. |
| <code>chosencol</code> | The column used when requesting the selected row's value. Defaults to <code>first</code> |
| <code>multiple</code> | A logical to determine if multiple selection is allowed. Defaults to <code>FALSE</code> |
| <code>handler</code> | Handler for double click events |
| <code>action</code> | Passed to handler |
| <code>container</code> | Optional container to attach widget to. |
| <code>...</code> | Passed to add method of container |
| <code>toolkit</code> | Which GUI toolkit to use |

Details

In an abstract sense, these trees are specified by a function which produces the value at a given node from the ancestry of the given node, and a function specifying if a node has offspring.

The `offspring` function determines the displayed data for a certain node. It has signature `(path, offspring.data)`, where the `path` consists of the ancestors and `offspring.data` is an optional value passed in when the tree object is constructed. This function returns a data frame. Its first column should consist of unique values, as it is treated like a key.

The `hasOffspring` function is called on the return value of `offspring`. It should return a logical indicating which rows have offspring. If this argument is not present, then the second column of the return values of `offspring` are consulted. If these are logical, then they are used to determine if offspring are present. Otherwise, no offspring are assumed.

The `icon.FUN` function is called on the return value of `offspring`. If present, it should return a vector of stock icon names.

The `svalue` method returns the current key. If `index` is set to a column number, that column's value will be returned.

The `"["` method refers to the vector of keys for the selected object.

The `addhandlerdoubleclick` handler can be set to respond to a double click event.

Examples

```
## Not run:
## function to find offspring
offspring <- function(path, user.data=NULL) {
  if(length(path) > 0)
    directory <- paste(getwd(), "/", paste(path, sep="/", collapse=""), sep=" ", collapse="")
  else
    directory <- getwd()
  files <- file.info(dir(path=directory))[,c(2,1,3)]

  files <- cbind(rownames(files), files)
  names(files)[1] <- "filename"
  return(files)
}
```

```

}
hasOffspring <- function(children,user.data=NULL, ...) {
  return(children$isdir)
}
icon.FUN <- function(children,user.data=NULL, ...) {
  x <- rep("file", length=nrow(children))
  x[children$isdir] <- "directory"
  return(x)
}
## shows isdir directory, as hasOffspring is specified
w <- gwindow("test with isdir showing")
gtree(offspring, hasOffspring, icon.FUN = icon.FUN, container=w)

## does not show isdir directory, as hasOffspring=NULL and
## second column is a logical
w <- gwindow("tree test no dir column")
gtree(offspring, hasOffspring=NULL, icon.FUN = icon.FUN, container=w)
## End(Not run)

```

guiToolkit

Function to select the GUI toolkit used by gWidgets

Description

A GUI toolkit is a separate package that implements the gWidgets API. This function allows one to select the toolkit to be used by default.

Usage

```
guiToolkit(name = NULL)
```

Arguments

| | |
|------|---|
| name | The name matches the package name without the initial gWidgets. For instance, "RGtk2" refers to the implementation provided by gWidgetsRGtk2. |
|------|---|

Details

Toolkits are named gWidgetsXXX.

Value

This function returns a subclass of guiWidgetsToolkit that is used for dispatching purposes by gWidgets. For example, the method svalue dispatches on its first argument and the value of the toolkit class stored in the toolkit slot of the object.

Examples

```
guiToolkit("RGtk2")
```

```
guiWidget-class    Class "guiWidget" ~~~
```

Description

Base class for gWidget objects

Objects from the Class

Objects can be created by calls of the form `new("guiWidget", ...)`.

These objects have two slots: a widget provided by a guiToolkit and a toolkit. Method dispatch is done on both values.

Slots

toolkit: Object of class "guiWidgetsToolkit". A specification of which GUI toolkit the widget uses.

widget: Object of class "ANY". A widget returned by the corresponding toolkit function.

Methods

```
.add signature(obj = "guiWidget", toolkit = "guiWidgetsToolkitRGtk2",
  value = "ANY"):...
.add signature(obj = "guiWidget", toolkit = "guiWidgetsToolkitRGtk2",
  value = "guiWidgetORgWidgetRGtkORRGtkObject"):...
.add signature(obj = "gContainerRGtk", toolkit = "guiWidgetsToolkitRGtk2",
  value = "guiWidget"):...
.add signature(obj = "gMenuRGtk", toolkit = "guiWidgetsToolkitRGtk2",
  value = "guiWidget"):...
.add signature(obj = "gNotebookRGtk", toolkit = "guiWidgetsToolkitRGtk2",
  value = "guiWidget"):...
.add signature(obj = "gTextRGtk", toolkit = "guiWidgetsToolkitRGtk2",
  value = "guiWidget"):...
.delete signature(obj = "gContainerRGtk", toolkit = "guiWidgetsToolkitRGtk2",
  widget = "guiWidget"):...
.delete signature(obj = "RGtkObject", toolkit = "guiWidgetsToolkitRGtk2",
  widget = "guiWidget"):...
.delete signature(obj = "gMenuRGtk", toolkit = "guiWidgetsToolkitRGtk2",
  widget = "guiWidget"):...
.svalue<- signature(obj = "gMenuRGtk", toolkit = "guiWidgetsToolkitRGtk2",
  value = "guiWidget"):...
.tag signature(obj = "guiWidget", toolkit = "guiWidgetsToolkitRGtk2"):
...
```

```
.tag<- signature(obj = "guiWidget", toolkit = "guiWidgetsToolkitRGtk2"):
...
[ signature(x = "guiWidget"):...
[<- signature(x = "guiWidget"):...
add3rdmousepopupmenu signature(obj = "guiWidget"):...
add signature(obj = "guiWidget"):...
addSpace signature(obj = "guiWidget"):...
addSpring signature(obj = "guiWidget"):...
adddropmotion signature(obj = "guiWidget"):...
adddropsourc signature(obj = "guiWidget"):...
adddroptarget signature(obj = "guiWidget"):...
addhandlerchanged signature(obj = "guiWidget"):...
addhandlerclicked signature(obj = "guiWidget"):...
addhandlerdestroy signature(obj = "guiWidget"):...
addhandlerdoubleclick signature(obj = "guiWidget"):...
addhandlerexpose signature(obj = "guiWidget"):...
addhandleridle signature(obj = "guiWidget"):...
addhandlerkeystroke signature(obj = "guiWidget"):...
addhandlerrightclick signature(obj = "guiWidget"):...
addhandlerunrealize signature(obj = "guiWidget"):...
addpopupmenu signature(obj = "guiWidget"):...
defaultWidget signature(obj = "guiWidget"):...
defaultWidget<- signature(obj = "guiWidget"):...
delete signature(obj = "guiWidget"):...
dim signature(x = "guiWidget"):...
dimnames signature(x = "guiWidget"):...
dimnames<- signature(x = "guiWidget"):...
dispose signature(obj = "guiWidget"):...
enabled signature(obj = "guiWidget"):...
enabled<- signature(obj = "guiWidget"):...
focus signature(obj = "guiWidget"):...
focus<- signature(obj = "guiWidget"):...
tooltip<- signature(obj = "guiWidget"):...
font signature(obj = "guiWidget"):...
font<- signature(obj = "guiWidget"):...
id signature(obj = "guiWidget"):...
id<- signature(obj = "guiWidget"):...
```

```

length signature(x = "guiWidget"): ...
names signature(x = "guiWidget"): ...
names<- signature(x = "guiWidget"): ...
removehandler signature(obj = "guiWidget"): ...
size signature(obj = "guiWidget"): ...
size<- signature(obj = "guiWidget"): ...
svalue signature(obj = "guiWidget"): ...
svalue<- signature(obj = "guiWidget"): ...
tag signature(obj = "guiWidget"): ...
tag<- signature(obj = "guiWidget"): ...
update signature(object = "guiWidget"): ...
visible signature(obj = "guiWidget"): ...
visible<- signature(obj = "guiWidget"): ...
isExtant signature(obj = "guiWidget"): ...

```

Note

Within gWidgets there are three main subclasses guiContainers, guiComponents and guiDialogs. The distinctions are more clear at the toolkit level.

Author(s)

John Verzani

gvarbrowser

Widget for browsing environment

Description

A widget to browse the objects in the current global environment

Usage

```

gvarbrowser(handler = NULL,
            action = "summary", container = NULL, ...,
            toolkit = guiToolkit())

```

Arguments

| | |
|-----------|--|
| handler | Handler for double click. Default is to call value of action on the object |
| action | Passed to handler. |
| container | Optional container to attach widget to |
| ... | ignored |
| toolkit | Which GUI toolkit to use |

Details

In gWidgetsRGtk2 there is an idle handler that updates the top-level components. However, changes are made below this, they will not be updated automatically. Rather, the user must close and expand that level.

The handler is called on the widget that provides the display, and not the gvarbrowser widget. If you want that in the handler, be sure to pass it in via the action argument.

svalue returns a character string containing the selected variable.

Examples

```
## Not run:
gvarbrowser(container=TRUE)
## End(Not run)
```

gWidgets-dialogs *Basic dialog constructors*

Description

A dialog is a widget that draws its own window. These dialogs are used for simple things – confirming a choice, gathering a single line of input, etc. Dialogs are always modal, meaning they must be closed before R can be interacted with again.

Usage

```
gmessage(message, title="message",
  icon = c("info", "warning", "error", "question"),
  parent = NULL,
  handler = NULL,
  action = NULL, ..., toolkit=guiToolkit())

ginput(message, text="", title="Input", icon = c("info", "warning",
  "error", "question"), parent=NULL,
  handler = NULL, action = NULL, ..., toolkit=guiToolkit())

gconfirm(message, title="Confirm", icon = c("info", "warning", "error",
  "question"), parent=NULL,
  handler = NULL, action = NULL, ..., toolkit=guiToolkit())

gbasicdialog(title = "Dialog", widget, parent=NULL, handler = NULL, action=NULL, ...)

galert(message, title = "message", delay=3, parent=NULL, ..., toolkit=guiToolkit())
```

Arguments

| | |
|---------|--|
| message | Message shown for widget |
| title | Title of window |
| icon | Which icon to show |
| text | default value for ginput text |
| widget | Widget to place in basic dialog. If missing, dialog returns a container. |
| parent | A gwindow() instance. If specified, dialog will be located in relation to this |
| handler | Handler called on OK selection. |
| action | Value passed to handler |
| delay | For galert, how long the transient message will appear |
| ... | Ignored |
| toolkit | Toolkit to use for GUI |

Details

These basic dialogs do slightly different things.

The `gmessage` dialog shows a message with an icon and a dismiss button. This dialog returns `TRUE` or `FALSE` as appropriate.

The `gconfirm` dialog shows a message with an icon and an OK button and a dismiss button. A handler may be attached to the OK button selection. This dialog returns `TRUE` or `FALSE` as appropriate.

The `ginput` dialog adds an edit box for gathering user information. The `text` argument sets the default value. This is then passed to the handler via the component `input` of the first argument of the handler. This dialog returns the value of the string if OK is clicked, otherwise `NA`.

For toolkits which support constructors without parent containers, the `gbasicdialog` dialog wraps a widget around an OK and dismiss button. The handler is called when the OK button is clicked. This dialog returns the `TRUE` when OK is clicked, otherwise `FALSE`.

If the toolkit does not support parentless constructors, then the use is different. The `widget` argument is missing and the return value is a container. This is basically a top-level window with OK and cancel buttons. The container is shown when the `visible` method is called with a value of `set=TRUE`. This creates a modal dialog. The handler specified to the constructor is called when OK is clicked and `TRUE` is returned. The value of `FALSE` is returned on cancel, and `NA` otherwise.

These dialogs are modal. This means that the R session freezes until the dialog is dismissed. This may be confusing to users if the window should appear below a currently drawn window.

The `galert` dialog is non-modal and does not grab the focus. Like `gmessage` it shows a message but unlike it, only for a short period of time and is unobtrusive.

Examples

```
## Not run:
gmessage("Hi there")
gconfirm("Are we having fun?", handler = function(h,...)
print("Yes"))
```

```

ginput("Enter your name", icon="question", handler = function(h,...) cat("Hi",h$input,"\n"))

## gbasicdialog
w <- gbasicdialog(title="Select a state", handler = function(h,...)
  print(svalue(tbl)))
tbl <- gtable(data.frame(states = rownames(state.x77)), expand=TRUE, cont = w)
visible(w, set=TRUE) ## show dialog

## End(Not run)

```

gWidgets-dnd

Functions to add drag and drop ability to widgets

Description

These functions allow drag and drop between widgets. The basic idea is that one creates drop sources from which one defines values which may be dragged and drop target where values may be dropped. These values can be text, or widgets.

Usage

```

adddropsource (obj, targetType = "text", handler = NULL, action = NULL,
  ...)
adddropmotion (obj, handler = NULL, action = NULL, ...)
adddroptarget (obj, targetType = "text", handler = NULL, action = NULL,
  ...)

```

Arguments

| | |
|------------|---|
| obj | Object to put drop handler on |
| targetType | What type of drop target: either "text" or "pixmap" or "entry". |
| handler | Handler called for the drop motion |
| action | action passed to handler |
| ... | |

Details

To specify if one can drag values from a widget use `adddropsource` called on the object. The argument `targetType` can be set to "object" when the drop value is to be a widget, and not a string. The arguments `handler` and `action` can be used to describe what gets dropped. The default is to drop the widget's contents as returned by `svalue`.

To specify if an object is a drop target the `adddroptarget` method is called on the object. The argument `handler` (but no `action`) is used to handle the drop.

The handler's first argument is a list with named components. The `obj` component refers to the object that has the target placed on it. The component `dropdata` is set by the `adddropsource`

method. The dropdata is typically a string, but a mechanism is in place to drop widgets. The default handler for `adddroptarget` is to replace the widget's value with the dropped data.

To add an action to a motion event, use the `adddropmotion` method. The `adddroptarget` must first have been added to the object.

Value

These functions return an ID returned when registering a handler. The function `removehandler` uses this information to remove a drag and drop handler.

Author(s)

Implementation of Simon Urbanek's `iwidgets` API was done by Michael Lawrence and John Verzani

See Also

[gWidgets-methods](#)

Examples

```
## Not run:
## simple dnd
lab = glabel("drag me", container=gwindow())
ed = gedit("drop here", container = gwindow())
adddropsourcelab)
adddroptarget(ed)
adddropmotion(ed, handler=function(h, ...) print("bombs away"))

## more complicated
## this shows that rows of editable data frame can be dropped.
## by assigning to the changed signal, the graphs can be dynamic.
## That is, drop a column, then edit it. The graph will update. The key
## is referring to the "value" stored in gd. This refers to the column
## in the editable data frame.
## By using svaluel() and id(), the dropped value can also be a
## character string referring to a variable in the workspace.
adf = gdf(mtcars, cont = TRUE)
gd = gggraphics(cont = TRUE)
plotData = function() {
  dropvalue = tag(gd, "value")
  theValues = svaluel(dropvalue)
  theName = id(dropvalue)
  hist(theValues, xlab=theName, main="")
}

ids = adddroptarget(gd, targetType="object", handler = function(h, ...) {
  tag(gd, "value") <- h$dropdata
  plotData()

  if(is.gdataframecolumn(h$dropdata)) {
    view.col = h$dropdata
    id = addhandlerchanged(view.col, handler=function(h, ...) plotData())
  }
})
```

```
    }  
  })  
  ## End(Not run)
```

gWidgets-handlers *Methods to add event handlers to objects*

Description

In the gWidgets API handlers are called in response to certain events such as keystrokes or clicks. This set of methods makes a consistent interface to some typical events. Not all handlers are defined for each widget.

Usage

```
addhandlerchanged(obj, handler = NULL, action = NULL, ...)  
addhandlerkeystroke(obj, handler = NULL, action = NULL, ...)  
addhandlerclicked(obj, handler = NULL, action = NULL, ...)  
addhandlerdoubleclick(obj, handler = NULL, action = NULL, ...)  
addhandlerrightclick(obj, handler = NULL, action = NULL, ...)  
addhandlerfocus(obj, handler = NULL, action = NULL, ...)  
addhandlerblur(obj, handler = NULL, action = NULL, ...)  
addhandlermousemotion(obj, handler = NULL, action = NULL, ...)  
addhandlerexpose(obj, handler = NULL, action = NULL, ...)  
addhandlerunrealize(obj, handler = NULL, action = NULL, ...)  
addhandlerdestroy(obj, handler = NULL, action = NULL, ...)  
addhandleridle (obj, handler = NULL, action = NULL, interval = 1000, ...)  
addpopupmenu(obj, menulist, action=NULL, ...)  
add3rdmousepopupmenu(obj, menulist, action=NULL, ...)  
removehandler(obj, ID, ...)
```

Arguments

| | |
|-----------------------|---|
| <code>obj</code> | The object to assign handler to |
| <code>handler</code> | A function to call if the given event occurs. The function's first argument is a list with some specific components. The component <code>obj</code> contains the object that the handler was assigned to. The <code>action</code> component contains the value given to the argument <code>action</code> . This can be used with <code>do.call</code> to make simple handlers. Or, this can be used to pass in other widgets, etc. Sometimes there are other components. For drag and drop handlers the component <code>dropdata</code> refers to the dropped data. For <code>ggraphics</code> the <code>addhandlerclicked</code> contains components <code>x</code> and <code>y</code> indicating where the click occurred. |
| <code>action</code> | Used to pass extra information into handlers |
| <code>interval</code> | For <code>addhandleridle</code> this specifies the time in milliseconds between calls to the handler. |
| <code>menulist</code> | For <code>addpopupmenu</code> and <code>add3rdmousepopupmenu</code> this specifies a menubar using a list which is in turn passed to <code>gmenu</code> . |
| <code>ID</code> | When a handler is assigned, an id is returned. This id can be used to remove a handler from an object. |
| <code>...</code> | |

Details

At first these handlers were all lowercase. These functions are still available, although the mixed case usage is encouraged

In GTK, and other toolkits, an event causes a signal to be triggered and these handlers are called in response to that signal.

These signals have various names known to the GTK programmer. say. These functions attempt to shield the gWidgets user from needing to learn these signals. For `gWidgetsRGtk`, if these handlers prove insufficient then the non-exported `addhandler` function has an additional `signal` argument: (`obj, signal, handler, action, ...`) for specifying a GTK signal. By avoiding this, we can make the gWidgets API non-toolkit specific.

The signals are defined to match the event described by the method name, e.g., "doubleclick."

The handlers all have signature (`h, ...`) where the first argument is a list with components `obj` containing the widget the handler is added to and `action` containing the values passed along to the `action` argument. This can be used to pass in other widget's names, when they can not be found from a function closure, say.

The handlers do not have lazy evaluation. The value of `action` is the one at the time of creation of the widget. (See the example). In GTK, a means to cheat this is to pass in a gWidget instance, as the underlying GTK objects are stored as pointers, not copies, so that when queried, their current state is used.

addHandlerChanged: This handler is called when a widget is "changed." This is interpreted differently by the various widgets. For `gedit` `change` refers to a changed value, not a keystroke change (when ENTER is pressed). For notebooks, this is called when a page is changed.

addHandlerKeystroke: This handler is called when keys are pressed in the text widgets. The extra argument `key` is used to pass back the key code of the pressed key.

addHandlerClicked: This handler is called when a widget, such as a button or label, is clicked.

addHandlerDoubleClick: This handler is called when a widget is doubleclicked, like in the tree widget. Not all widgets receive a double click signal. Only when a single mouse click is needed for selection is this implemented.

addHandlerRightclick: This handler is called when a widget is clicked with the right mouse button

addHandlerFocus: This handler is called when a widget gains focus

addHandlerBlur: This handler is called when a widget loses focus

addHandlerMouseMotion: This handler is called when a the mouse moves over a widget. In some toolkits it is called just once per visit to the widget, for others maybe multiple times. This is like a mouseover for web pages. The drag motion handler is similar, only it is called when a drag event is dragged over a widget.

addHandlerExpose: handler is called when a widget is exposed. For instance when a page in a notebook is exposed.

addHandlerUnrealize: handler is called when a widget is being unrealized.

addHandlerDestroy: handler is called when a widget is being destroyed. For top level windows, this usually allows one to intercept the window destroy event for purposes of saving work etc.

addHandlerIdle: handler is called every so often, and can be used to update a widget's content. This method has an extra argument `interval` specifying the interval in milliseconds with a default of 1000 or 1 second.

Although not handlers, the `addPopupMenu` method adds a popup menu to a mouse click. The popup menu is specified using a list that is passed to `gmenu`.

A refinement of this is the `add3rdMousePopmenu` method which puts the popupmenu on the right mouse click.

See Also

[gWidgets-methods](#)

Examples

```
## Not run:
## a default handler, useful for when action is enough to
## specify desired results

handler.default = function(h,...) do.call(h$action, list(svalue(h$obj)))
group = ggroup(horizontal=FALSE, container=gwindow("Click
button"))
button = gbutton("Click me", container=group)
addhandlerclicked(button, handler=handler.default, action="print")

## use two widgets, one to update the other
```

```

group = ggroup(horizontal=FALSE, container=gwindow("two widgets"))
button = gbutton("click me", container=group)
label = glabel("Button has not been clicked", container=group)
addhandlerclicked(button, handler = function(h,...) {
  svalue(h$obj) <- "click me again"
  svalue(h$action) <- "Button has been clicked"
}, action = label)

## lazy evaluation is not used here
obj = 4
gbutton("click",cont=TRUE, handler=function(h,...)
print(h$action), action=obj)
obj = 2
## now click button and value of 4 will be printed, not 2

## Whereas, if one uses a gWidget we get the same as lazy
## loading
obj = gedit("4")
gbutton("click",cont=TRUE, handler=function(h,...)
  print(svalue(h$action)), action=obj)
svalue(obj) <- "2"
## Now click and "2" is printed.
## End(Not run)

```

gWidgets-icons

Functions for adding icons

Description

Two functions for listing "stock" icons, and adding "stock" icons. A stock icon can be referenced within gWidgets by its simple name, such as "ok" or "close".

Usage

```

addStockIcons(iconNames, iconFiles, ..., toolkit=guiToolkit())

getStockIcons(..., toolkit=guiToolkit())

```

Arguments

| | |
|-----------|--|
| iconNames | A vector of icon names |
| iconFiles | A matching vector of filenames for the icons |
| ... | ignored |
| toolkit | Which toolkit to use |

Details

The file type must be supported by the toolkit. The `tcltk` supports few filetypes without additional libraries.

Examples

```
## Not run:
  iconNames <- c("larrow","rarrow")
  iconFiles <= c("/usr/share/icons/larrow.png",
                "/usr/share/icons/rarrow.png")
  addStockIcons(iconNames, iconFiles)
## End (Not run)
```

gWidgets-methods *Methods for gWidgets instances*

Description

Methods introduced by the gWidgets API.

Details

The base class for this gWidgets implementation are gWidget and its subclass gComponent and gContainer. However, it is expected that the toolkit implementations have several classes of their own. The following methods defined in gWidgets simply dispatch to a similarly named widget in the toolkit. For instance, the method svalue is defined like

```
svalue(obj, ...) <- .svalue(obj@widget, obj@toolkit, ...) where .svalue()
and obj@widget are in toolkit, and obj@toolkit is used for dispatching in the appropriate
toolkit.
```

The gComponent methods are:

svalue(obj, index=NULL, drop=NULL, ...): This method returns the "selected" value in a widget. Selection varies from widget to widget, but should generally is what can be added to the widget by mouse click or typing. For some widgets, the extra argument `index=TRUE` will return the index of the selected value, not the value. For some widget, the argument `drop` is given to either prevent or encourage dropping of information.

svalue<-(obj, index=NULL, ... , value): This method is used to set the selected value in a widget programatically. The `index` argument is used when the value is set by index.

[(x, i, j, ..., drop=TRUE): For widgets where selection is a choice from a collection of items, the `svalue` method refers to the choice and the square bracket notation refers to the items. For instance, in a radio button (`gradio`) the `svalue` method returns the selected value, the `"["` method refers to the vector of possible values. Whereas in a notebook (`gnotebook`), the `svalue` method refers to the currently opened page and the `"["` refers to all the pages.

"[<-(x, i, j, ..., value) : In those cases where it makes sense assignment to pieces of the widget can be made with the square bracket notation. For instance, for the radio widget, this can be used to change the labels.

size(obj, ...) or size<-(obj, ..., value): Returns or sets the size of the object. For setting the size, the value is given in terms of width and height of widget in pixels.

- visible(obj ...)** or **visible<-(obj, ..., value)**: Used to check if widget is visible or not. When setting the visibility, `value` should be a logical. "Visibility" differs from widget to widget. For `gwindow` it refers to whether the base container is shown or not. For the dataframe-like widgets `gdf` and `gtable` visibility refers to which rows are shown.
- visible(obj ...)** Used to check if a `gwindow` object is extant (not been destroyed). An R object can point to a window that can no longer be shown, as it may have been closed by the window manager.
- enabled(obj, ...)** or **enabled<-(obj, ..., value)** When a widget is disabled, the toolkit makes it unresponsive to user input and changes the color of it, usually by graying it out, to indicate it is disabled. This method is used to change the state.
- focus(obj, ...)** or **focus<-(obj, ..., value)**: method to check if a widget has focus, or to force focus on a widget.
- tooltip<-(obj, value)** Add a tooltip to the widget. Tooltips are toolkit and eventloop dependent, so may be unreliable.
- defaultWidget(obj, ...)** or **defaultWidget<-(obj, ..., value)** Sets the widget to be activated when the parent window has focus and the enter key is pressed.
- font(obj, ...)** or **font<-(obj, ..., value)**: Can be used to check or set font attributes in a widget. In `gWidgetsRGtk`, the font attributes are given as a named vector. The available names are `family` with a value of "normal", "sans", "serif", or "monospace"; `style` with a value of "normal", "oblique", or "italic"; `weight` with a value of "ultra-light", "light", "normal", "bold", "ultra-bold", or "heavy"; and `color` which for `gWidgetsRGtk` is any of the values returned by `colors`. [Prior to version 0.0-22 the weight and style were switched. Old code can be run as before.]
- tag(obj, i, drop=TRUE, ...)** or **tag<-(obj, i, replace=TRUE, ..., value)**: These functions work like the `attr` function – they set values within an object. In `RGtk`, these are carried with the pointer which is passed into functions – not a copy. This allows values to be set without worrying about the scope of the assignment.
- When setting a tag, the argument `replace` can be set to `FALSE` so that the value appends. The tags are stored internally in a list. Calling `tag(obj)` will return this list.
- id(obj, ...)** or **id<-(obj, ..., value)**: An id is a name for a widget. This is primarily used internally with the spread-sheet like widgets so that columns can have values – the data in the column, and an id – the column name. Objects can be given an id like a name. For non-widget items, the id is an attribute.
- update(object, ...)**: Some classes use this method to update the state of the widget
- add(obj, value, ...)**: This widget is used to add something to a widget. What "adding" means varies from widget to widget.
- For this method, there are several different arguments that can be passed in via the "... " argument. When the API is cleaned up this should change.*
- For the containers (`gwindow`, `ggroup`, ...) adding adds a widget to be packed in. For the parent container produced by `gwindow` only one item can be added. For groups, this is not the case. For `ggroup`, `gframe` and `gexpandgroup` the extra argument `expand=TRUE` will cause the widget to take up all possible space within the container.
- For the components, `add` has different meanings. For notebooks (`gnotebook`, ...) `add` is used to add pages. In this case the extra arguments are:
- label** to assign the label. This may be a text string or a `gWidget`

override.closebutton To override the placing of a close button

For the text buffer widget (`gtext`) `insert` (originally called `add` which still works but is deprecated) is used to insert text into the buffer. In this case, extra arguments are available:

font.attr can be used to specify font attributes for the text

do.newline a logical indicating if a newline should be added after the text

where An indicator of where to place the text: "beginning", "ending", or "at.cursor", although the latter may not be implemented.

delete(obj, widget, ...): For `gContainers` this method is used to delete a widget that has been added with `add`. In `RGtk`, the widget is actually detached and can be added at a later time. Any handler assigned by `addhandlerunrealize` is called when the widget is detached

For notebooks, the `delete` method removes a page in the notebook.

dispose(obj, ...): This method is used to remove an object.

For top-level windows it destroys the window.

For notebooks, it removes the current page.

In `RGtk2`, for other objects it will destroy the top-level window.

addSpace(obj, value, horizontal=TRUE, ...): Used to add space between widgets in a container

addSpring(obj, ...): When packing widgets into a group the widget abut each other filling in from left to right or top to bottom. This puts a "spring" between two widgets forcing the ones to the right of (or below) the spring to be pushed as far as possible to the right (or bottom).

Note

See package vignette for more examples

See Also

[gWidgets-handlers](#) for methods related to handlers.

gWidgets-undocumented

Undocumented, but exported, functions

Description

Undocumented, but exported functions from `gWidgets`.

gwindow

*Constructor for base container***Description**

Widgets are packed inside containers which may in turn be packed inside other containers. The base container is known as a window. Only one container may be packed inside a window.

Usage

```
gwindow(title = "Window", visible = TRUE, name=title,
        width = NULL, height= NULL, parent=NULL,
        handler = NULL, action = NULL,
        ..., toolkit = guiToolkit())
```

Arguments

| | |
|---------|--|
| title | Title of window |
| visible | If TRUE window is drawn when constructed. Otherwise, window can be drawn latter using <code>visible<-</code> . |
| name | Name for registry of windows |
| width | Default width for window at creation |
| height | Default height for window at creation |
| parent | If non-NULL, can be used to suggest default location of window. The argument name was changed from location to parent. This can be a coordinate pair (x,y) with (0,0) the upper left corner, or a gwindow instance. In the latter case the location is suggested by the location of the current window. This is useful for placing dialogs near the parent window. |
| handler | Handler for destroy event. |
| action | Passed to handler |
| ... | Not used |
| toolkit | Which GUI toolkit to use |

Details

A base window can also be created using the argument `container=TRUE` when constructing a widget.

The `svalue` method refers to the window title. Use `svalue<-` to change the title.

The `add` method is used to add a widget or container to the base window. For top-level windows, some toolkits only support adding one widter, so in `gWidgets` only one widget should be added to a window, so usually it would be another container.

Additionally the `menubar`, `toolbar` and `statusbar` widgets should now be added and deleted from the top-level window (Not yet implemented in `gWidgetsrJava`). Outside of `RGtk2`, the other toolkits expect these items to be properties of a top-level window.

The `dispose` method destroys the window.

The `size` method sets the minimum size. Use the `width` and `height` arguments to set the default size when the window is constructed.

A window is destroyed in response to a `destroy` event. However, when the window manager tries to close a window first a "delete-event" is issued. If this has the right value then the "destroy" event is fired. The `addHandlerUnrealize` handler can be called to intercept the closing of the window. Its handler should return a logical: `TRUE` to prevent the closing, `FALSE` to proceed.

Examples

```
## Not run:
## window with handler
win <- gwindow("Window example",
  handler=function(h,...) {
    print("See ya")
  })
gbutton("cancel", cont=win,
  handler = function(h,...) dispose(win))

## block closing of window
win <- gwindow("Window example")
addHandlerUnrealize(win, handler = function(h,...) {
  val <- gconfirm("Really close window", parent=h$obj)
  if(as.logical(val))
    return(FALSE)           # destroy
  else
    return(TRUE)           # don't destroy
})

## transient dialog (gWidgetsRGtk2)
pwin <- gwindow("Parent window")
cwin <- gwindow("Child window", parent = pwin)
## clicking button close parent causing child to close too
gbutton("close both", cont=cwin,
  handler = function(h,...) dispose(pwin))
## End(Not run)
```

Index

*Topic **classes**

guiWidget-class, 45

*Topic **interface**

gbutton, 3
gcheckbox, 4
gcheckboxgroup, 5
gcombobox, 7
gcommandline, 9
gdf, 10
gedit, 11
gfile, 12
gformlayout, 14
ggenericwidget, 17
ggraphics, 19
ggroup, 21
ghelp, 23
ghtml, 24
gimage, 25
glabel, 26
glayout, 27
gmenu, 28
gnotebook, 31
gpanedgroup, 33
gradio, 34
gseparator, 35
gslider, 36
gstatusbar, 37
gtable, 38
gtext, 41
gtree, 42
guiToolkit, 44
gvarbrowser, 48
gWidgets-dialogs, 49
gWidgets-dnd, 50
gWidgets-handlers, 52
gWidgets-icons, 56
gWidgets-methods, 57
gWidgets-undocumented, 59
gwindow, 59

*Topic **package**

gWidgets-package, 2
.add, ANY, ANY, gWidgetANY-method
(*gWidgets-undocumented*), 59
.add, gContainerRGtk, guiWidgetsToolkitRGtk2, guiWidget-
(*gWidgets-methods*), 57
.add, gHelpANY, ANY, ANY-method
(*gWidgets-methods*), 57
.add, gHelpANY-method
(*gWidgets-undocumented*), 59
.add, gMenuRGtk, guiWidgetsToolkitRGtk2, guiWidget-
(*gWidgets-methods*), 57
.add, gNotebookRGtk, guiWidgetsToolkitRGtk2, guiWidget-
(*gWidgets-methods*), 57
.add, gTextRGtk, guiWidgetsToolkitRGtk2, guiWidget-
(*gWidgets-methods*), 57
.add, guiWidget, guiWidgetsToolkitRGtk2, ANY-method
(*gWidgets-methods*), 57
.add, guiWidget, guiWidgetsToolkitRGtk2, guiWidgetO
(*gWidgets-methods*), 57
.addStockIcons, ANY-method
(*gWidgets-icons*), 56
.delete, RGtkObject, guiWidgetsToolkitRGtk2, guiWidget-
(*gWidgets-methods*), 57
.delete, gContainerRGtk, guiWidgetsToolkitRGtk2, gu
(*gWidgets-methods*), 57
.delete, gMenuRGtk, guiWidgetsToolkitRGtk2, guiWidget
(*gWidgets-methods*), 57
.dispose, gHelpANY-method
(*gWidgets-undocumented*), 59
.fixFontMessUp
(*gWidgets-undocumented*), 59
.gcommandline, ANY-method
(*gWidgets-undocumented*), 59
.getStockIcons, ANY-method
(*gWidgets-icons*), 56
.gformlayout, ANY-method
(*gWidgets-undocumented*), 59
.ggenericwidget, ANY-method

- (gWidgets-undocumented)*, 59
- .ghelp, ANY-method
(gWidgets-undocumented), 59
- .ghelpbrowser, ANY-method
(gWidgets-undocumented), 59
- .leftBracket, gCommandLineANY-method
(gWidgets-undocumented), 59
- .leftBracket, gFormLayoutANY-method
(gWidgets-undocumented), 59
- .length, gHelpANY-method
(gWidgets-undocumented), 59
- .names, gFormLayoutANY-method
(gWidgets-undocumented), 59
- .stockIconFromClass, ANY-method
(gWidgets-icons), 56
- .stockIconFromObject, ANY-method
(gWidgets-icons), 56
- .svalue, character-method
(gWidgets-undocumented), 59
- .svalue, gAddargANY-method
(gWidgets-undocumented), 59
- .svalue, gBivariateANY-method
(gWidgets-undocumented), 59
- .svalue, gCommandLineANY-method
(gWidgets-undocumented), 59
- .svalue, gFileURLANY-method
(gWidgets-undocumented), 59
- .svalue, gFormLayoutANY-method
(gWidgets-undocumented), 59
- .svalue, gGenericWidgetANY-method
(gWidgets-undocumented), 59
- .svalue, gHelpANY-method
(gWidgets-undocumented), 59
- .svalue, gLmerANY-method
(gWidgets-undocumented), 59
- .svalue, gModelANY-method
(gWidgets-undocumented), 59
- .svalue, gUnivariateANY-method
(gWidgets-undocumented), 59
- .svalue, gUnivariateTableANY-method
(gWidgets-undocumented), 59
- .svalue<-, gAddargANY-method
(gWidgets-undocumented), 59
- .svalue<-, gBivariateANY-method
(gWidgets-undocumented), 59
- .svalue<-, gCommandLineANY-method
(gWidgets-undocumented), 59
- .svalue<-, gMenuRGtk, guiWidgetsToolkitRGtk2, *(gWidgets-undocumented)*, 59
- (gWidgets-methods)*, 57
- .svalue<-, gModelANY-method
(gWidgets-undocumented), 59
- .tag, gWidgetANY-method
(gWidgets-undocumented), 59
- .tag, guiWidget, guiWidgetsToolkitRGtk2-method
(gWidgets-methods), 57
- .tag, guiWidget-method
(gWidgets-undocumented), 59
- .tag<-, gWidgetANY-method
(gWidgets-undocumented), 59
- .tag<-, guiWidget, guiWidgetsToolkitRGtk2-method
(gWidgets-methods), 57
- .tag<-, guiWidget-method
(gWidgets-undocumented), 59
- [, gCommandLineANY-method
(gWidgets-undocumented), 59
- [, gFormLayoutANY-method
(gWidgets-undocumented), 59
- [, gWidgetANY-method
(gWidgets-undocumented), 59
- [, guiWidget-method
(gWidgets-methods), 57
- [<-, gWidgetANY-method
(gWidgets-undocumented), 59
- [<-, guiWidget-method
(gWidgets-methods), 57
- add *(gWidgets-methods)*, 57
- add, ANY-method
(gWidgets-undocumented), 59
- add, guiComponent-method
(gWidgets-methods), 57
- add, guiContainer-method
(gWidgets-methods), 57
- add, guiWidget-method
(gWidgets-methods), 57
- add3rdMousePopupmenu
(gWidgets-handlers), 52
- add3rdmousepopupmenu
(gWidgets-handlers), 52
- add3rdMousePopupmenu, guiWidget-method
(gWidgets-methods), 57
- add3rdmousepopupmenu, guiWidget-method
(gWidgets-methods), 57
- addDropMotion *(gWidgets-dnd)*, 50
- adddropmotion *(gWidgets-dnd)*, 50
- addDropMotion, guiWidget-method

- addDropMotion, *guiWidget-method*
(*gWidgets-methods*), 57
- addDropSource (*gWidgets-dnd*), 50
- addDropSource (*gWidgets-dnd*), 50
- addDropSource, *guiWidget-method*
(*gWidgets-methods*), 57
- addDropSource, *guiWidget-method*
(*gWidgets-methods*), 57
- addDropTarget (*gWidgets-dnd*), 50
- addDropTarget (*gWidgets-dnd*), 50
- addDropTarget, *guiWidget-method*
(*gWidgets-methods*), 57
- addDropTarget, *guiWidget-method*
(*gWidgets-methods*), 57
- addHandler (*gWidgets-handlers*), 52
- addhandler (*gWidgets-handlers*), 52
- addHandler, *guiWidget-method*
(*gWidgets-methods*), 57
- addhandler, *guiWidget-method*
(*gWidgets-methods*), 57
- addHandlerBlur
(*gWidgets-handlers*), 52
- addhandlerblur
(*gWidgets-handlers*), 52
- addHandlerBlur, *guiWidget-method*
(*gWidgets-undocumented*), 59
- addhandlerblur, *guiWidget-method*
(*gWidgets-undocumented*), 59
- addHandlerChanged
(*gWidgets-handlers*), 52
- addhandlerchanged
(*gWidgets-handlers*), 52
- addHandlerChanged, *guiWidget-method*
(*gWidgets-methods*), 57
- addhandlerchanged, *guiWidget-method*
(*gWidgets-methods*), 57
- addHandlerClicked
(*gWidgets-handlers*), 52
- addhandlerclicked
(*gWidgets-handlers*), 52
- addHandlerClicked, *guiWidget-method*
(*gWidgets-methods*), 57
- addhandlerclicked, *guiWidget-method*
(*gWidgets-methods*), 57
- addHandlerDestroy
(*gWidgets-handlers*), 52
- addhandlerdestroy
(*gWidgets-handlers*), 52
- addHandlerDestroy, *guiWidget-method*
(*gWidgets-methods*), 57
- addhandlerdestroy, *guiWidget-method*
(*gWidgets-methods*), 57
- addHandlerDoubleclick
(*gWidgets-handlers*), 52
- addhandlerdoubleclick
(*gWidgets-handlers*), 52
- addHandlerDoubleclick, *guiWidget-method*
(*gWidgets-methods*), 57
- addhandlerdoubleclick, *guiWidget-method*
(*gWidgets-methods*), 57
- addHandlerExpose
(*gWidgets-handlers*), 52
- addhandlerexpose
(*gWidgets-handlers*), 52
- addHandlerExpose, *guiWidget-method*
(*gWidgets-methods*), 57
- addhandlerexpose, *guiWidget-method*
(*gWidgets-methods*), 57
- addHandlerFocus
(*gWidgets-handlers*), 52
- addhandlerfocus
(*gWidgets-handlers*), 52
- addHandlerFocus, *guiWidget-method*
(*gWidgets-undocumented*), 59
- addhandlerfocus, *guiWidget-method*
(*gWidgets-undocumented*), 59
- addHandlerIdle
(*gWidgets-handlers*), 52
- addhandleridle
(*gWidgets-handlers*), 52
- addHandlerIdle, *guiWidget-method*
(*gWidgets-methods*), 57
- addhandleridle, *guiWidget-method*
(*gWidgets-methods*), 57
- addHandlerKeystroke
(*gWidgets-handlers*), 52
- addhandlerkeystroke
(*gWidgets-handlers*), 52
- addHandlerKeystroke, *guiWidget-method*
(*gWidgets-methods*), 57
- addhandlerkeystroke, *guiWidget-method*
(*gWidgets-methods*), 57
- addHandlerMouseMotion
(*gWidgets-handlers*), 52
- addhandlermousemotion
(*gWidgets-handlers*), 52

- addHandlerMouseMotion, guiWidget-method
(*gWidgets-methods*), 57
- addhandlermousemotion, guiWidget-method
(*gWidgets-methods*), 57
- addHandlerRightclick
(*gWidgets-handlers*), 52
- addhandlerrightclick
(*gWidgets-handlers*), 52
- addHandlerRightclick, guiWidget-method
(*gWidgets-methods*), 57
- addhandlerrightclick, guiWidget-method
(*gWidgets-methods*), 57
- addHandlerUnrealize
(*gWidgets-handlers*), 52
- addhandlerunrealize
(*gWidgets-handlers*), 52
- addHandlerUnrealize, guiWidget-method
(*gWidgets-methods*), 57
- addhandlerunrealize, guiWidget-method
(*gWidgets-methods*), 57
- addPopupMenu (*gWidgets-handlers*),
52
- addpopupmenu (*gWidgets-handlers*),
52
- addPopupMenu, guiWidget-method
(*gWidgets-methods*), 57
- addpopupmenu, guiWidget-method
(*gWidgets-methods*), 57
- addSpace (*gWidgets-methods*), 57
- addSpace, guiWidget-method
(*gWidgets-methods*), 57
- addSpring (*gWidgets-methods*), 57
- addSpring, guiWidget-method
(*gWidgets-methods*), 57
- addStockIcons, 26
- addStockIcons (*gWidgets-icons*), 56

- blockHandler (*gWidgets-handlers*),
52
- blockhandler (*gWidgets-methods*),
57
- blockHandler, guiWidget-method
(*gWidgets-methods*), 57
- blockhandler, guiWidget-method
(*gWidgets-methods*), 57

- defaultWidget (*gWidgets-methods*),
57
- defaultWidget, guiWidget-method
(*gWidgets-methods*), 57
- defaultWidget<-
(*gWidgets-methods*), 57
- defaultWidget<-, guiWidget-method
(*gWidgets-methods*), 57
- delete (*gWidgets-methods*), 57
- delete, guiWidget-method
(*gWidgets-methods*), 57
- dim, guiWidget-method
(*gWidgets-methods*), 57
- dimnames, guiWidget-method
(*gWidgets-methods*), 57
- dimnames, gWidgetANY-method
(*gWidgets-undocumented*), 59
- dimnames<-, guiWidget-method
(*gWidgets-methods*), 57
- dimnames<-, gWidgetANY-method
(*gWidgets-undocumented*), 59
- dispose (*gWidgets-methods*), 57
- dispose, guiWidget-method
(*gWidgets-methods*), 57

- editFormulaDialog
(*gWidgets-undocumented*), 59
- editSelectDialog
(*gWidgets-undocumented*), 59
- editSubsetDialog
(*gWidgets-undocumented*), 59
- enabled (*gWidgets-methods*), 57
- enabled, guiWidget-method
(*gWidgets-methods*), 57
- enabled, gWidgetANY-method
(*gWidgets-undocumented*), 59
- enabled<- (*gWidgets-methods*), 57
- enabled<-, guiWidget-method
(*gWidgets-methods*), 57
- enabled<-, gWidgetANY-method
(*gWidgets-undocumented*), 59

- focus (*gWidgets-methods*), 57
- focus, guiWidget-method
(*gWidgets-methods*), 57
- focus, gWidgetANY-method
(*gWidgets-undocumented*), 59
- focus<- (*gWidgets-methods*), 57
- focus<-, guiWidget-method
(*gWidgets-methods*), 57

- focus<-, gWidgetANY-method
(*gWidgets-undocumented*), 59
- font (*gWidgets-methods*), 57
- font, guiWidget-method
(*gWidgets-methods*), 57
- font<- (*gWidgets-methods*), 57
- font<-, guiWidget-method
(*gWidgets-methods*), 57
- font<-, gWidgetANY-method
(*gWidgets-undocumented*), 59

- gaction (*gmenu*), 28
- galert (*gWidgets-dialogs*), 49
- gbasicdialog (*gWidgets-dialogs*),
49
- gbutton, 3
- gcalendar (*gfile*), 12
- gcheckbox, 4, 6, 8, 34
- gcheckboxgroup, 5, 8, 34
- gcombobox, 6, 7, 34
- gcommandline, 9
- gComponentANY-class
(*guiWidget-class*), 45
- gconfirm (*gWidgets-dialogs*), 49
- gContainerANY-class
(*guiWidget-class*), 45
- gdf, 10, 40
- gdfnotebook (*gdf*), 10
- gdroplist (*gcombobox*), 7
- gedit, 11
- getStockIcons, 26
- getStockIcons (*gWidgets-icons*), 56
- getToolkitWidget
(*gWidgets-methods*), 57
- getToolkitWidget, guiWidget-method
(*gWidgets-methods*), 57
- gexpandgroup, 32
- gexpandgroup (*ggroup*), 21
- gfile, 12
- gfilebrowse (*gfile*), 12
- gformlayout, 14
- gframe, 32
- gframe (*ggroup*), 21
- ggenericwidget, 15, 17
- ggraphics, 19
- ggraphicsnotebook (*ggraphics*), 19
- ggroup, 21, 32
- ghelp, 23
- ghelpbrowser (*ghelp*), 23

- ghtml, 24
- gimage, 25
- ginput (*gWidgets-dialogs*), 49
- glabel, 21, 26
- glayout, 22, 27
- gmenu, 28
- gmessage (*gWidgets-dialogs*), 49
- gnotebook, 22, 31
- gpanedgroup, 22, 33
- gradio, 6, 8, 34
- gseparator, 35
- gslider, 36
- gspinbutton (*gslider*), 36
- gstatusbar, 37
- gtable, 6, 8, 10, 11, 34, 38
- gtext, 41
- gtoolbar (*gmenu*), 28
- gtree, 40, 42
- guiComponent-class
(*guiWidget-class*), 45
- guiContainer-class
(*guiWidget-class*), 45
- guiDialog-class
(*guiWidget-class*), 45
- guiToolkit, 44
- guiWidget-class, 45
- guiWidgetOrNULL-class
(*guiWidget-class*), 45
- guiWidgetsToolkit-class
(*guiToolkit*), 44
- guiWidgetsToolkitRGtk2-class
(*guiToolkit*), 44
- guiWidgetsToolkitrJava-class
(*guiToolkit*), 44
- guiWidgetsToolkitRwxWidgets-class
(*guiToolkit*), 44
- guiWidgetsToolkitSJava-class
(*guiToolkit*), 44
- guiWidgetsToolkitctk-class
(*guiToolkit*), 44
- gvarbrowser, 48
- gwCat (*gWidgets-undocumented*), 59
- gWidgetANY-class
(*guiWidget-class*), 45
- gWidgets (*gWidgets-package*), 2
- gWidgets-handlers, 5, 6, 8, 34, 59
- gWidgets-methods, 5, 6, 8, 34, 55
- gWidgets-dialogs, 49

- gWidgets-dnd, [50](#)
- gWidgets-handlers, [52](#)
- gWidgets-icons, [56](#)
- gWidgets-methods, [52](#), [57](#)
- gWidgets-package, [2](#)
- gWidgets-undocumented, [59](#)
- gWidgets-undocumented.Rd
(*gWidgets-undocumented*), [59](#)
- gwindow, [22](#), [32](#), [59](#)

- id (*gWidgets-methods*), [57](#)
- id, guiWidget-method
(*gWidgets-methods*), [57](#)
- id<- (*gWidgets-methods*), [57](#)
- id<-, guiWidget-method
(*gWidgets-methods*), [57](#)
- insert (*gWidgets-methods*), [57](#)
- insert, guiComponent-method
(*gWidgets-methods*), [57](#)
- installing_gWidgets_toolkits
(*gWidgets-package*), [2](#)
- isExtant (*gWidgets-methods*), [57](#)
- isExtant, guiWidget-method
(*gWidgets-methods*), [57](#)
- isExtant, gWidgetANY-method
(*gWidgets-undocumented*), [59](#)

- length, guiWidget-method
(*gWidgets-methods*), [57](#)

- names, guiWidget-method
(*gWidgets-methods*), [57](#)
- names, gWidgetANY-method
(*gWidgets-undocumented*), [59](#)
- names<-, guiWidget-method
(*gWidgets-methods*), [57](#)
- names<-, gWidgetANY-method
(*gWidgets-undocumented*), [59](#)

- removeHandler
(*gWidgets-handlers*), [52](#)
- removehandler
(*gWidgets-handlers*), [52](#)
- removeHandler, guiWidget-method
(*gWidgets-methods*), [57](#)
- removehandler, guiWidget-method
(*gWidgets-methods*), [57](#)

- show, guiWidget-method
(*gWidgets-undocumented*), [59](#)

- size (*gWidgets-methods*), [57](#)
- size, guiWidget-method
(*gWidgets-methods*), [57](#)
- size<- (*gWidgets-methods*), [57](#)
- size<-, guiWidget-method
(*gWidgets-methods*), [57](#)
- size<-, gWidgetANY-method
(*gWidgets-undocumented*), [59](#)
- stockIconFromClass
(*gWidgets-icons*), [56](#)
- stockIconFromObject
(*gWidgets-icons*), [56](#)
- str2 (*gWidgets-undocumented*), [59](#)
- svalue (*gWidgets-methods*), [57](#)
- svalue, character-method
(*gWidgets-undocumented*), [59](#)
- svalue, guiWidget-method
(*gWidgets-methods*), [57](#)
- svalue, gWidgetANY-method
(*gWidgets-undocumented*), [59](#)
- svalue, numeric-method
(*gWidgets-undocumented*), [59](#)
- svalue<- (*gWidgets-methods*), [57](#)
- svalue<-, guiWidget-method
(*gWidgets-methods*), [57](#)
- svalue<-, gWidgetANY-method
(*gWidgets-undocumented*), [59](#)

- tag (*gWidgets-methods*), [57](#)
- tag, guiWidget-method
(*gWidgets-methods*), [57](#)
- tag, gWidgetANY-method
(*gWidgets-undocumented*), [59](#)
- tag<- (*gWidgets-methods*), [57](#)
- tag<-, guiWidget-method
(*gWidgets-methods*), [57](#)
- tag<-, gWidgetANY-method
(*gWidgets-undocumented*), [59](#)
- tooltip<- (*gWidgets-methods*), [57](#)
- tooltip<-, guiWidget-method
(*gWidgets-methods*), [57](#)

- unblockHandler
(*gWidgets-handlers*), [52](#)
- unblockhandler
(*gWidgets-methods*), [57](#)
- unblockHandler, guiWidget-method
(*gWidgets-methods*), [57](#)

unblockhandler, guiWidget-method
 (*gWidgets-methods*), 57
update, guiWidget-method
 (*gWidgets-methods*), 57
update, gWidgetANY-method
 (*gWidgets-undocumented*), 59

visible (*gWidgets-methods*), 57
visible, guiWidget-method
 (*gWidgets-methods*), 57
visible, gWidgetANY-method
 (*gWidgets-undocumented*), 59
visible<- (*gWidgets-methods*), 57
visible<-, guiWidget-method
 (*gWidgets-methods*), 57
visible<-, gWidgetANY-method
 (*gWidgets-undocumented*), 59