

# Package ‘gWidgets2’

May 8, 2026

**Type** Package

**Title** Rewrite of gWidgets API for Simplified GUI Construction

**Version** 1.0-10

**URL** <https://github.com/gWidgets3/gWidgets2>

**Description** Re-implementation of the 'gWidgets' API. The API is defined in this package. A second, toolkit-specific package is required to use it. At this point only 'gWidgets2ctk' is viable.

**Depends** methods, digest

**License** GPL (>= 3)

**LazyLoad** yes

**Collate** 'misc.R' 'guiToolkit.R' 'BasicInterface.R' 'S4-methods.R' 'S3-methods.R' 'command-stack.R' 'methods.R' 'dialogs.R' 'gWidgets2-package.R' 'gaction.R' 'gbutton.R' 'gcalendar.R' 'gcheckbox.R' 'gcheckboxgroup.R' 'gcombobox.R' 'gdf.R' 'gdfnotebook.R' 'gedit.R' 'ggroup.R' 'gframe.R' 'gexpandgroup.R' 'gfile.R' 'gfilter.R' 'gformlayout.R' 'ggraphics.R' 'ggraphicsnotebook.R' 'ghtml.R' 'gimage.R' 'glabel.R' 'glayout.R' 'gmenu.R' 'gnotebook.R' 'gpanedgroup.R' 'gprogressbar.R' 'gradio.R' 'gseparator.R' 'gslider.R' 'gspinbutton.R' 'gstackwidget.R' 'gstatusbar.R' 'gtable.R' 'gtext.R' 'gtimer.R' 'gtoolbar.R' 'gtree.R' 'ws-model.R' 'gvarbrowser.R' 'gwindow.R' 'handler-methods.R' 'icons.R'

**RoxygenNote** 7.1.2

**NeedsCompilation** no

**Author** John Verzani [aut, cre]

**Maintainer** John Verzani <jverzani@gmail.com>

**Repository** CRAN

**Date/Publication** 2024-12-07 19:00:07 UTC

## Contents

gWidgets2-package	3
.gdfnotebook.default	7
add	8
addHandlerChanged.GGraphics	9
addSpring	14
addStockIcons	15
call_meth	17
check_deprecated	18
check_return_class	18
dispose	19
editable	19
enabled	20
flatten	20
focus	21
font	21
gaction	22
galert	23
gbasicdialog	24
gbutton	26
gcalendar	28
gcheckbox	29
gcheckboxgroup	31
gcombobox	34
gconfirm	36
gcontainer	37
gdf	37
gdfnotebook	39
gedit	39
getToolkitWidget	41
getWithDefault	42
get_index_in_list	42
get_object_from_string	43
gexpandgroup	43
gfile	45
gfilter	47
gformlayout	50
gframe	51
ggraphics	53
ggraphicsnotebook	54
ggroup	56
ghtml	58
gimage	59
ginput	60
glabel	61
glayout	62
gmenu	64

gmessage . . . . .	66
gnotebook . . . . .	67
gpanedgroup . . . . .	69
gprogressbar . . . . .	70
gradio . . . . .	71
gseparator . . . . .	74
gslider . . . . .	75
gspinbutton . . . . .	77
gstackwidget . . . . .	79
gstatusbar . . . . .	80
gtable . . . . .	81
gtext . . . . .	84
gtimer . . . . .	87
gtoolbar . . . . .	88
gtoolkit . . . . .	89
gtree . . . . .	90
guiToolkit . . . . .	94
guiWidgetsToolkit-class . . . . .	94
gvarbrowser . . . . .	96
gwidget . . . . .	97
gwindow . . . . .	98
installing_gWidgets_toolkits . . . . .	100
isExtant . . . . .	101
is_empty . . . . .	101
is_MacOSX . . . . .	102
is_Windows . . . . .	102
observer . . . . .	102
redo . . . . .	103
short_summary . . . . .	103
size . . . . .	104
svalue . . . . .	105
tag . . . . .	106
tooltip . . . . .	107
undo . . . . .	107
visible . . . . .	108
XXX . . . . .	108
[.GDefaultWidget . . . . .	109

## Description

The **gWidgets2** package provides a programming interface for making graphical user interfaces within R. The package is a rewrite of the **gWidgets** package, introducing a few external changes but a significant number of internal ones. The package relies on one of several underlying toolkit packages providing access to the graphical libraries. These will include **RGtk2**, **tcltk**, **qtbase**, and a collection of browser widgets provided by ExtJS. As of now, only **gWidgets2RGtk2** is near completion.

## Details

The package provides constructors to produce controls, the widgets that a user interacts with; containers, GUI objects used to organize the controls within a window; and dialogs, simple one-off windows for gathering quick user feedback. These objects are manipulated through various methods. The package provides a few new generics and, as much as possible, leverages existing methods for R.

### Control constructors:

Controls are created by constructors. The package API includes the following. As much as possible these are implemented in the toolkit packages, but there may be instances where that is not possible.

- `gbutton` Provides a basic button to initiate an action
- `gcalendar` Provides a text entry area with selector for a date
- `gcheckbox` Provides a labeled checkbox to allow a user to toggle a selection
- `gcheckboxgroup` Provides a collection of checkboxes allowing selection of 0, 1, or several from many items
- `gcombobox` Provides a drop down list of choices to select from and possible and entry area for free response
- `gdf` Provides a data frame editing widget
- `gedit` Provides a single line text entry widget
- `ggraphics` Provides an embeddable graphic device
- `gimage` Provides a widget to hold images
- `glabel` Provides a widget to hold labels for other controls
- `gmenu` Provides menus for top-level windows and popup menus
- `gradio` Provides a means to select one of many items
- `gseparator` Provides a visual line to separate off parts of a window
- `gslider` Provides a means to select one value from a (seeming) continuum of values
- `gspinbutton` Provides means to select a value from s sequence of values
- `gstatusbar` Provides a widget to display status messages in a top-level window
- `gtable` Provides a widget to display tabular data for selection
- `gtext` Provides a multiline text-editing widget
- `gtimer` Provides a one-shot or repeatable timer
- `gtoolbar` Provides toolbars for top-level windows
- `gtree` Provides a display for heirarchical data
- `gvarbrowser` Provides a widget showing a shapshot of the current global workspace
- `gaction` Provides a means to encapsulate actions for use with menu bars, tool bars and buttons.

Containers are used to organize controls with a window. The package provides the following:

`gexpandgroup` Provides a container with an option to disclose or hide its children  
`gframe` Provides a framed box container  
`ggroup` Provides a horizontal or vertical box container for packing in child components  
`glayout` Provides a container to organize data by row and column cell  
`gnotebook` Provides a notebook container  
`gpanedgroup` Provides a divided container with adjustable divider  
`gstackwidget` Provides a container like a notebook, but without tab labels  
`gwindow` Provides a top-level window

### Dialog constructors:

Dialogs in **gWidgets2** are typically modal, meaning they block input to any other window and the R process. They do not return objects to be manipulated through methods, but rather return values selected by the user.

`gmessage` Produces a simple dialog to display a message  
`gconfirm` Produces a dialog for a user to confirm an action  
`ginput` Provides a dialog to gather user input  
`gbasicdialog` Provides a means to produce general modal dialogs  
`galert` Provides a short transient message dialog  
`gfile` Provides a dialog to select a filename or directory name

### Methods:

Except for dialogs, the constructors produce objects for which several methods are defined that allow the programmer access to getting and setting of the object state. For the most part these are S3 methods. The actual returned object is a reference class instance, as provided by an underlying toolkit. These may have toolkit-specific methods defined as reference class methods (i.e., call them using `$meth_name`). Any such methods are documented in the toolkit packages.

`svalue, svalue<-` The main new method. This is used to retrieve or set the main property associated with a widget  
`enabled, enabled<-` A widget is enabled if it is sensitive to user input. Non-enabled widgets typically are rendered in a greyed out state.  
`visible, visible<-` The generic idea of a visible widget is one that is drawn. However, several classes override this to mean part of the widget is visible or not visible.  
`focus, focus<-` A widget with focus receives any keyboard input.  
`editable, editable<-` A widget is editable if it can receive keyboard input.  
`font, font<-` The font for an object is specified through this method using a convention illustrated in the help page.  
`size, size<-` The size of a widget is retrieved or requested through these methods  
`tooltip, tooltip<-` A tooltip provides contextual information when a mouse hovers over an object  
`undo, redo` Some widgets support an undo and redo stack  
`isExtant` A method to check if the GUI part of a widget still exists. (A constructor produces an R object and GUI object through the toolkit.)

`tag, tag<-` A method used to set attributes for an object that are stored in an environment so that they are passed by reference, not copy. This allows event handlers to manipulate an object's attributes outside the scope of the callback.

`getToolkitWidget` Returns the underlying toolkit object that is packaged into a **gWidgets2** object

`add` Method used to add a child component to a parent container

`delete` Method used to delete a component from its parent

`dispose` Method used to delete a component

The package overloads some familiar R methods.

`length, length<-` Returns the length of an object, typically related to the number of children a container has, or the length of the items that a user can selection from.

`dim` Used to return row and column size information as applicable.

`names, names<-` Used to set the names associated to an object. These may be column names in the table widget, or tab names in the notebook container.

`dimnames, dimnames<-` Used to set row and column names, as applicable.

`[, [<-` Used to update the underlying items that a selection widget offers. Also used to specify layout in glayout

`update` Call to update the state of a widget, when applicable.

### Event handlers:

Graphical User Interfaces are made interactive by assigning a function (a callback) to be called when some event happens. In **gWidgets2** the `addHandlerXXX` methods are defined to assign this callback to a type of event specified through the `XXX`, detailed below. The generic `addHandlerChanged` is the most common event for a widget. This event can also have a handler specified through the `handler` argument of the widget constructor.

In **gWidgets2** handlers are functions which when called are passed a list as the first argument, and possibly toolkit-specific arguments for subsequent arguments. As such the signature typically looks like `(h, ...)`, where the list `h` has components `obj`, containing a reference to the widget emitting the event and `action` passing in any information specified to the `action` argument. Some events also pass back extra information, such as `x` and `y` for position, or `key` for key events, as appropriate.

`addHandlerChanged` Assigns callback for the most generic event

`addHandlerClicked` Assigns callback for a mouse click event

`addHandlerDoubleClick` Assigns callback for a mouse double-click event

`addHandlerRightclick` Assigns callback for a mouse right-click event

`addHandlerColumnclicked` Assigns callback for a column-click event

`addHandlerColumnDoubleclicked` Assigns callback for a column-double-click event

`addHandlerColumnRightclicked` Assigns callback for a column-right-click event

`addHandlerSelect` Assigns callback when the underlying selection is changed

`addHandlerFocus` Assigns a callback for when a widget receives focus

`addHandlerBlur` Assigns a callback for when a widget loses focus

`addHandlerDestroy` Assigns a callback for when a widget is destroyed

`addHandlerUnrealize` For `gwindow` this is called before the `destroy` event and may stop that from happening.

[addHandlerExpose](#) Assigns callback to be called when a widget is exposed  
[addHandlerKeystroke](#) Assigns callback to be called when a key event occurs  
[addHandlerMouseMotion](#) Assigns callback to be called when a mouse moves over a widget  
[addHandler](#) Base method to add a callback though rarely called, as it is toolkit specific  
[addHandlerIdle](#) Assign a callback to be called at periodic intervals. See also [gtimer](#)  
[addPopupMenu](#) Add a popup menu  
[addRightclickPopupMenu](#) Add a popup menu for the right mouse (context menu)  
[addDropSource](#) Specify widget as a source (drag area) for drag and drop  
[addDropTarget](#) Specify widget as a target (drop area) for drag and drop  
[addDragMotion](#) Assign callback for event that a drag event crosses a widget  
[blockHandlers](#), [blockHandler](#) Block all handlers for a widget (or by single ID)  
[unblockHandlers](#), [unblockHandler](#) Unblock any blocked handlers (or by single ID)  
[removeHandler](#) Remove a handler by it ID

### Author(s)

John Verzani <jverzani@gmail.com>

Maintainer: John Verzani <jverzani@gmail.com>

---

.gdfnotebook.default *Toolkit constructor*

---

### Description

Toolkit constructor

### Usage

```
## Default S3 method:  
.gdfnotebook(toolkit, items, container = NULL, ...)
```

### Arguments

toolkit	toolkit
items	data frame for initial page, when given
container	parent container
...	passed to add method of parent container

---

 add
 

---



---

*Add a child object to parent container*


---

### Description

Add packs in child objects.

Delete may or may not remove a child. This is toolkit specific. It may also be tied up with garbage collection. To avoid that, keep a reference to the child object before deleting.

### Usage

```
add(obj, child, expand = FALSE, fill = NULL, anchor = NULL, ...)
```

```
## Default S3 method:
```

```
add(obj, child, expand = FALSE, fill = NULL, anchor = NULL, ...)
```

```
delete(obj, child)
```

```
## S3 method for class 'GContainer'
```

```
delete(obj, child)
```

### Arguments

obj	parent object
child	child widget
expand	NULL or logical. For box containers controls whether a child will expand to fill the allocated space.
fill	NULL or character. For box containers. The value of fill (not always respected) is used to control if expansion happens vertically (y), horizontally (x) or both (both or TRUE). For vertically filled box containers, children always fill horizontally (atleast) and for horizontally filled box containers, children always fill vertically (atleast). This is important to realize when trying to size buttons, say.
anchor	NULL or integer. For box containers. The anchor argument is used to position the child within the parent when there is more space allocated than the child requests. This is specified with a Cartesian pair in $\{-1,0,1\} \times \{-1, 0, 1\}$ .
...	passed on to the

---

addHandlerChanged.GGraphics  
*change handler for ggraphics*

---

## Description

The change handler for ggraphics is called when a rubber-band selection is completed

The click handler is called on a mouse click. The handler object should pass in value for x, y

A GUI is made interactive by assigning handlers to user-generated events, such as a mouse click, change of widget state, or keyboard press. In **gWidgets2** handlers are assigned through various addHandlerXXX methods. The handlers are functions whose first argument should expect a list with components obj (to pass in the receiver object) and action (to pass in any user-supplied value to the action argument). Some handlers add other components, such as mouse position information on a click, or key information on a keyboard event.

The "changed" event varies wildly amongst the widgets, but is meant to be the most "obvious" one. Typically this is also similar to "selected".

This may not be supported by all toolkits.

This may not be supported by all toolkits.

This may not be supported by all toolkits.

For table widgets (gtable, gdf) clicking the column header should trigger this event. The column that is clicked on is passed back in the component column.

If defined (gtable, gdf) calls event handler for double click event. Passes back column information through column component.

The select event defaults to the "changed" event.

The "select" event is when a user "selects" an object, the "selection changed" event is when the selection changes. The distinction is in table and tree widgets where a user may select values with a single click yet want to initiate an action with a double click. The latter is the "addHandlerSelect" event, the former the "addHandlerSelectionChanged" event.

When a widget has the focus, it will receive the keyboard input. This handler is called when a widget receives the focus.

A blur or focus out event for a widget triggers this event handler

When a widget is destroyed, a handler can be assigned to perform any clean up tasks that are needed.

For gwindow objects this handler is called before the window is closed. If this handler returns TRUE the window will be closed, if FALSE the window will not be closed. In contrast, the "destroy" handler does not allow conditional destruction.

The "h" argument has components key for the key and possibly modifier for the modifier.

deprecated. See [gtimer](#).

Defaults to adding a right-click mouse popup menu, better known as a context menu, though some toolkits have both this and the latter provided.

These menus are also known as context menus, though there isn't really a good mechanism within **gWidgets2** to make the menu items context sensitive.

Drag and drop requires one to register widgets as sources for dragging, a widget as a target for dropping.

The handler is called on the drop event. The component's `dropData` passes in the value being transferred by dragging.

Block all handlers for an object. Removed via `unblockHandlers`.

The block is a counter that gets decremented. If more `blockHandlers` calls are made than `unblockHandlers`, the handlers will still be blocked.

## Usage

```
## Default S3 method:
addHandlerChanged(obj, handler, action = NULL, ...)

## Default S3 method:
addHandlerClicked(obj, handler, action = NULL, ...)

addHandler(obj, signal, handler, action = NULL, ...)

## Default S3 method:
addHandler(obj, signal, handler, action = NULL, ...)

addHandlerChanged(obj, handler, action = NULL, ...)

## Default S3 method:
addHandlerChanged(obj, handler, action = NULL, ...)

addHandlerClicked(obj, handler, action = NULL, ...)

## Default S3 method:
addHandlerClicked(obj, handler, action = NULL, ...)

addHandlerDoubleClick(obj, handler, action = NULL, ...)

## Default S3 method:
addHandlerDoubleClick(obj, handler, action = NULL, ...)

addHandlerRightclick(obj, handler, action = NULL, ...)

## Default S3 method:
addHandlerRightclick(obj, handler, action = NULL, ...)

addHandlerShiftclick(obj, handler, action = NULL, ...)

## Default S3 method:
addHandlerShiftclick(obj, handler, action = NULL, ...)

addHandlerControlclick(obj, handler, action = NULL, ...)
```

```
## Default S3 method:
addHandlerControlclick(obj, handler, action = NULL, ...)

addHandlerColumnclicked(obj, handler, action = NULL, ...)

## Default S3 method:
addHandlerColumnclicked(obj, handler, action = NULL, ...)

addHandlerColumnDoubleclicked(obj, handler, action = NULL, ...)

## Default S3 method:
addHandlerColumnDoubleclicked(obj, handler, action = NULL, ...)

addHandlerColumnRightclicked(obj, handler, action = NULL, ...)

## Default S3 method:
addHandlerColumnRightclicked(obj, handler, action = NULL, ...)

addHandlerSelect(obj, handler, action = NULL, ...)

## Default S3 method:
addHandlerSelect(obj, handler, action = NULL, ...)

addHandlerSelectionChanged(obj, handler, action = NULL, ...)

## Default S3 method:
addHandlerSelectionChanged(obj, handler, action = NULL, ...)

addHandlerFocus(obj, handler, action = NULL, ...)

## Default S3 method:
addHandlerFocus(obj, handler, action = NULL, ...)

addHandlerBlur(obj, handler, action = NULL, ...)

## Default S3 method:
addHandlerBlur(obj, handler, action = NULL, ...)

addHandlerDestroy(obj, handler, action = NULL, ...)

## Default S3 method:
addHandlerDestroy(obj, handler, action = NULL, ...)

addHandlerUnrealize(obj, handler, action = NULL, ...)

## Default S3 method:
addHandlerUnrealize(obj, handler, action = NULL, ...)
```

```
addHandlerExpose(obj, handler, action = NULL, ...)  
  
## Default S3 method:  
addHandlerExpose(obj, handler, action = NULL, ...)  
  
addHandlerKeystroke(obj, handler, action = NULL, ...)  
  
## Default S3 method:  
addHandlerKeystroke(obj, handler, action = NULL, ...)  
  
addHandlerMouseMotion(obj, handler, action = NULL, ...)  
  
## Default S3 method:  
addHandlerMouseMotion(obj, handler, action = NULL, ...)  
  
addHandlerIdle(...)  
  
addPopupMenu(obj, menulist, action = NULL, ...)  
  
## Default S3 method:  
addPopupMenu(obj, menulist, action = NULL, ...)  
  
addRightclickPopupMenu(obj, menulist, action = NULL, ...)  
  
## Default S3 method:  
addRightclickPopupMenu(obj, menulist, action = NULL, ...)  
  
## Default S3 method:  
addRightclickPopupMenu(obj, menulist, action = NULL, ...)  
  
## Default S3 method:  
addRightclickPopupMenu(obj, menulist, action = NULL, ...)  
  
addDropSource(  
  obj,  
  handler,  
  action = NULL,  
  data.type = c("text", "object"),  
  ...  
)  
  
## Default S3 method:  
addDropSource(  
  obj,  
  handler,  
  action = NULL,  
  data.type = c("text", "object"),  
  ...  
)
```

```

)

addDropTarget(obj, handler, action = NULL, ...)

## Default S3 method:
addDropTarget(obj, handler, action = NULL, ...)

addDragMotion(obj, handler, action = NULL, ...)

## Default S3 method:
addDragMotion(obj, handler, action = NULL, ...)

blockHandlers(obj, ...)

## Default S3 method:
blockHandlers(obj, ...)

blockHandler(obj, ID, ...)

## Default S3 method:
blockHandler(obj, ID, ...)

unblockHandlers(obj, ...)

## Default S3 method:
unblockHandlers(obj, ...)

unblockHandler(obj, ID, ...)

## Default S3 method:
unblockHandler(obj, ID, ...)

removeHandler(obj, ID, ...)

## Default S3 method:
removeHandler(obj, ID, ...)

```

### Arguments

obj	object receiving event and emitting a signal to the handler
handler	handler to assign when signal is emitted. A handler is a function, its first argument should expect a list with components obj containing a reference to the object and action. Some handlers are passed additional values.
action	passed to handler to parameterize call.
...	passed along
signal	toolkit signal, e.g. "clicked"
menulist	a list of gaction items or a gmenu instance

data.type	Type of data returned. It is either text or an object
ID	returned by addHandler. If missing will try to block all handler passed to constructor

### Details

Although the `add_handler` method, to which `addHandler` dispatches, is basically the workhorse to add a handler to response to a signal, it generally isn't called directly, as its use is not cross toolkit. Rather, if possible, one should use the `addHandlerXXX` methods to add a handler. These dispatch to this (basically) but do so in a toolkit independent manner.

This call (and the others) returns a handler ID which may be used for some toolkits later on to remove, block or unblock the call. All handlers for a widget may be blocked or unblocked via `blockHandlers` and `unblockHandlers`.

The "changed" event is also the one that a handler passed to the constructor is called on.

To specify the values that is transferred in a drag and drop event, the handler specified here should return the value to pass via drag and drop. It will appear as the `dropdata` component of the list passed in as the first argument of the drop handler

### Value

a handler ID which can be used to block/unblock or remove the handler

### Note

This method is not toolkit independent, as the signal value depends on the toolkit

For the `gWidgets2Qt` package one can not block, unblock or remove a single handler, but rather must do all the objects handlers at once.

### See Also

[blockHandlers](#), [unblockHandlers](#), [blockHandler](#), [unblockHandler](#), and [removeHandler](#)  
[addHandlerUnrealize](#).

[blockHandlers](#) to block all handlers for widget

---

addSpring

*Add a spring to box containers*

---

### Description

A spring will separate the children packed in the box container prior to the spring be added and those being added, pushing the two as far apart as the allocated space will allow.

Add spring to `GContainer` class

Inserts a specific amount of space between the previously packed child and next one.

Add space to `GContainer` class

**Usage**

```
addSpring(obj)

## S3 method for class 'GContainer'
addSpring(obj)

addSpace(obj, value)

## S3 method for class 'GContainer'
addSpace(obj, value)
```

**Arguments**

obj	GContainer object
value	space in pixels to add

---

addStockIcons	<i>Method to add icon to list of stock icons</i>
---------------	--

---

**Description**

Method to add icon to list of stock icons

generic for dispatch

toolkit implementation

return list of available stock icons

generic for toolkit dispatch

default

Return stock icon name, filename, icon object from its by name

generic

default implementation

Find a stock icon from the given class

generic for dispatch

Default stock icon for a given class name

Find stock icon from the given object

generic for dispatch

get stock icon from object by class

**Usage**

```

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

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

## Default S3 method:
.addStockIcons(toolkit, iconNames, iconFiles, ...)

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

.getStockIcons(toolkit, ...)

## Default S3 method:
.getStockIcons(toolkit, ...)

getStockIconByName(name, ..., toolkit = guiToolkit())

.getStockIconByName(toolkit, name, ...)

## Default S3 method:
.getStockIconByName(toolkit, name, file = TRUE, ...)

stockIconFromClass(theClass, ..., toolkit = guiToolkit())

.stockIconFromClass(toolkit, theClass, ...)

## Default S3 method:
.stockIconFromClass(toolkit, theClass, ...)

stockIconFromObject(obj, ..., toolkit = guiToolkit())

.stockIconFromObject(toolkit, obj, ...)

## Default S3 method:
.stockIconFromObject(toolkit, obj, ...)

```

**Arguments**

iconNames	names of icons
iconFiles	path of icons
...	ignored
toolkit	used to dispatch into toolkit if a separate implementation is made
name	of stock icon
file	logical If TRUE, return filename. If FALSE, return toolkit icon object (if possible).
theClass	name of class
obj	an R object

**Value**

list of icons with names the icon name and values the icon file name or icon object (as needed by the toolkit)

name of icon.

**Examples**

```
## Not run:
## we can add icon sets, say those of glyphicons.com. Steps are download files, unzip
## then point x to path, y to name. Imagine we download and current directory is
## png directory. (Won't work with tcltk by default as these are png files)
x <- Sys.glob("*.png")
path <- paste(getwd(), x, sep=.Platform$file.sep)
nm <- gsub("\\.png", "", x)
nm <- gsub("-", "_", nm)
nm <- gsub("\\+", "_plus", nm)
addStockIcons(nm, path)

## End(Not run)
```

---

call\_meth

*helper function to bypass lack of cached value in method call*

---

**Description**

helper function to bypass lack of cached value in method call

**Usage**

```
call_meth(meth, obj)
```

**Arguments**

meth	method name
obj	method of object's class

**Value**

the method

**Note**

use as `do.call(call_meth, args)`

---

check_deprecated	<i>Method to send message if any deprecated arguments are being used</i>
------------------	--

---

**Description**

Many arguments were deprecated due to various reasons. This is meant to ease porting of code.

**Usage**

```
check_deprecated(deprecated_args = list(), ...)
```

**Arguments**

deprecated_args	named list of deprecated args
...	named avlues

---

check_return_class	<i>check that toolkit object return the right class</i>
--------------------	---

---

**Description**

The S3 dispatch assumes naming conventions in the class names. This offers some check.

**Usage**

```
check_return_class(obj, ret_class)
```

**Arguments**

obj	object with expected return class
ret_class	character string of class expected

**Value**

throws error if a mismatch

---

dispose	<i>Dispose of object</i>
---------	--------------------------

---

**Description**

Dispose of object, primarily a window though this is modified in GNoteBook and GText.

**Usage**

```
dispose(obj, ...)
```

```
## S3 method for class 'GComponent'
```

```
dispose(obj, ...)
```

**Arguments**

obj	object to dispose
...	passed along

---

editable	<i>Controls whether widget is editable or not</i>
----------	---

---

**Description**

Some widgets may be editable. If possible, the setter method can be used to toggle the state. This method indicates the state.

**Usage**

```
editable(obj, i)
```

```
## Default S3 method:
```

```
editable(obj, i)
```

```
editable(obj, i) <- value
```

```
editable(obj, i) <- value
```

**Arguments**

obj	object
i	index to apply to, when applicable
value	logical. Set editable state.

---

enabled	<i>enabled</i>
---------	----------------

---

**Description**

A widget is enabled if it is sensitive to user input

**Usage**

```
enabled(obj)

## Default S3 method:
enabled(obj)

enabled(obj) <- value

enabled(obj) <- value
```

**Arguments**

obj	object
value	logical

**Value**

logical indicating if widget is enabled  
if value is logical and FALSE widget will be insensitive to user input and rendered in a muted state.

---

flatten	<i>Flatten a nested list</i>
---------	------------------------------

---

**Description**

Flatten a nested list

**Usage**

```
flatten(x)
```

**Arguments**

x	a list
---	--------

**Author(s)**

Tommy (<http://stackoverflow.com/questions/8139677/how-to-flatten-a-list-to-a-list-without-coercion>)

---

focus	<i>Does widget have focus</i>
-------	-------------------------------

---

**Description**

a widget has focus if it will receive input events

For some widgets, this sets user focus (e.g. gedit gets focus for typing). For others, setting the focus calls the raise methods. (for gwindow, it will raise the window)

**Usage**

```
focus(obj)
```

```
## Default S3 method:  
focus(obj)
```

```
focus(obj) <- value
```

```
focus(obj) <- value
```

**Arguments**

obj	object
value	logical. Set focus state.

---

font	<i>Returns font specification for widget, if available</i>
------	--

---

**Description**

The font assignment method is used to change the font of the currently selected text.

**Usage**

```
font(obj)
```

```
## Default S3 method:  
font(obj)
```

```
font(obj) <- value
```

```
font(obj) <- value
```

```
## S3 replacement method for class 'GText'  
font(obj) <- value
```

**Arguments**

obj	object
value	<p>The font specification is given in terms of a named vector or list where the names indicate a font attribute and the value a reasonable choice:</p> <p><b>weight</b> c("light", "normal", "medium", "bold", "heavy")</p> <p><b>style</b> c("normal", "oblique", "italic")</p> <p><b>family</b> c("sans", "helvetica", "times", "monospace")</p> <p><b>size</b> an integer, say c(6,8,10,11,12,14,16,18,20, 24,36,72)</p> <p><b>color (or foreground)</b> One of colors()</p> <p><b>background</b> One of colors()</p> <p><b>scale</b> c("xx-large", "x-large", "large", "medium", "small", "x-small", "xx-small")</p> <p>These are from Gtk's font specs, which though fairly standard, may not be totally supported in the other toolkits.</p>

gaction

*An action constructor***Description**

A action object encapsulates an action (a callback) adding textual and graphic information. Actions may be proxied in buttons, menu bars or tool bars.

**Usage**

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

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

**Arguments**

label	label for action
tooltip	tooltip for action
icon	icon (stock icon name) for icon
key.accel	keyboard accelerator. If given, parent must be specified.
handler	handler to call when action is invoked
action	values passed to parameterize action
parent	parent window. Needed if keyboard accelerator used.
...	These values are passed to the add method of the parent container, and occasionally have been used to sneak in hidden arguments to toolkit implementations.
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <a href="#">guiToolkit</a> .

**Value**

a gaction instance

---

galert	<i>Alert dialog to display transient messages</i>
--------	---

---

**Description**

Alert dialog to display transient messages  
generic for toolkit dispatch

**Usage**

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

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

**Arguments**

msg	character. main message. If length is 2, second component is used for detail, providing it is available.
title	Title (may not be displayed)
delay	length of time (in seconds) to display
parent	parent object to show near
...	ignored
toolkit	toolkit

**See Also**

[gmessage](#), [gconfirm](#), [gbasicdialog](#), [galert](#)

---

gbasicdialog

*Constructor for modal dialog that can contain an arbitrary widget*

---

**Description**

The basic dialog is basically a modal window. To use there is a 3 step process: 1) Create a container by calling this constructor, say `dlg`; 2) use `dlg` as a container for your subsequent GUI; 3) set the dialog to be modal by calling `visible(dlg)`. (One can't call `visible(dlg) <- TRUE`.)

We overrode the basic use of `visible` for the `gbasicdialog` container to have it become visible and modal after this call. The better suited call `visible(dlg) <-TRUE` does not work as wanted, for we want to capture the return value.

dispose method for a basic dialog

**Usage**

```
gbasicdialog(
  title = "Dialog",
  parent = NULL,
  do.buttons = TRUE,
  handler = NULL,
  action = NULL,
  ...,
  toolkit = guiToolkit()
)
```

```
.gbasicdialog(
  toolkit,
  title = "Dialog",
  parent = NULL,
  do.buttons = TRUE,
  handler = NULL,
```

```

    action = NULL,
    ...
)

## S3 method for class 'GBasicDialog'
visible(obj, ...)

## S3 method for class 'GBasicDialog'
dispose(obj, ...)

```

### Arguments

title	title for window
parent	parent to display by
do.buttons	FALSE to suppress buttons when no parent
handler	handler called when Ok button invoked
action	passed to handler for OK button
...	ignored
toolkit	toolkit
obj	dialog object

### Value

A GBasicDialog instance with a visible method  
 logical indicating which button was pushed (or TRUE if no buttons present)

### See Also

[gmessage](#), [gconfirm](#), [gbasicdialog](#), [galert](#)

### Examples

```

## Not run:
## a modal dialog for editing a data frme
fix_df <- function(DF, ...) {
  dfname <- deparse(substitute(DF))
  w <- gbasicdialog(..., handler=function(h,...) {
    assign(dfname, df[,], .GlobalEnv)
  })
  g <- ggroup(cont=w, horizontal=FALSE)
  glabel("Edit a data frame", cont=g)
  df <- gdf(DF, cont=g, expand=TRUE)
  size(w) <- c(400, 400)
  out <- visible(w)
}

m <- mtcars[1:3, 1:4]
fix_df(m)

```

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

---

```
gbutton
```

```
Basic button widget
```

---

## Description

The basic button widget is a standard means to provide the user a mechanism to invoke an action. This action may be specified by a handler or by a gaction object. The main property for GButton is the label text. If this text matches a stock icon name and the toolkit supports it, an icon will accompany the button.

The svalue method for a button object refers to its main property, the button label

## Usage

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

```
.gbutton(toolkit, text, handler, action, container, ...)
```

```
## S3 method for class 'GButton'
addHandlerChanged(obj, handler, action = NULL, ...)
```

```
## S3 method for class 'GButton'
svalue(obj, index = NULL, drop = NULL, ...)
```

## Arguments

text	label text. If text matches a stock icon name, that is used as well
handler	A handler assigned to the default change signal. Handlers are called when some event triggers a widget to emit a signal. For each widget some default signal is assumed, and handlers may be assigned to that through addHandlerChanged or at construction time. Handlers are functions whose first argument, h in the documentation, is a list with atleast two components obj, referring to the object emitting the signal and action, which passes in user-specified data to parameterize the function call.  Handlers may also be added via addHandlerXXX methods for the widgets, where XXX indicates the signal, with a default signal mapped to addHandlerChanged (cf. <a href="#">addHandler</a> for a listing). These methods pass back a handler ID that can be used with blockHandler and unblockHandler to suppress temporarily the calling of the handler.

action	User supplied data passed to the handler when it is called
container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <b>gWidgets2tcltk</b> or <b>gWidgets2WWW2</b> .)
...	These values are passed to the add method of the parent container. Examples of values are <code>expand</code> , <code>fill</code> , and <code>anchor</code> , although they're not always supported by a given widget. For more details see <a href="#">add</a> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with <code>gaction</code> and the dialogs.
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <code>guiToolkit</code> .
obj	object receiving event and emitting a signal to the handler
index	NULL or logical. If TRUE and widget supports it an index, instead of a value will be returned.
drop	NULL or logical. If widget supports it, drop will work as it does in a data frame or perhaps someother means.

### Value

a `GButton` instance. While this object has its own (reference) methods, one primarily interacts with it through S3 methods defined within the package.

### Examples

```
if(interactive()) {

  w <- gwindow("Buttons", visible=FALSE)
  g <- ggroup(cont=w, horizontal=FALSE)

  ## various buttons

  ## with icons
  b1 <- gbutton("open", cont=g)

  ## without icon
  b2 <- gbutton("ouvrir", cont=g)

  ## by an action
  act <- gaction("open", tooltip="open", icon="open", handler=function(...) {})
  b3 <- gbutton(action=act, cont=g)

  ## with a handler
  b4 <- gbutton("click me", cont=g, handler=function(h,...) {
    if(svalue(b2) == "open")
      svalue(b2) <- "ouvrir"
    else
      svalue(b2) <- "open"
  })
}
```

```

    })

    ## handlers can be blocked/unblocked
    b5 <- gbutton("Click me for a message", cont=g)
    id <- addHandlerClicked(b5, function(h,...) print("Ouch"))
    b6 <- gcheckbox("toggle handler message", cont=g, use.togglebutton=TRUE, handler=function(h,...) {
      if (svalue(b6)) {
        blockHandler(b5, id)
      } else {
        unblockHandler(b5, id)
      }
    })
  })

  visible(w) <- TRUE
}

```

---

gcalendar

*A constructor for a date selection widget*


---

### Description

The date is the main property of this widget

The svalue method for a calendar object returns the selected date

### Usage

```

gcalendar(
  text = "",
  format = "%Y-%m-%d",
  handler = NULL,
  action = NULL,
  container = NULL,
  ...,
  toolkit = guiToolkit()
)

```

```

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

```

```

## S3 method for class 'GCalendar'

```

```

addHandlerChanged(obj, handler, action = NULL, ...)

## S3 method for class 'GCalendar'
svalue(obj, index = NULL, drop = NULL, ...)

```

### Arguments

text	initial text
format	Date format
handler	handler called when changed
action	passed to handler
container	parent container
...	passed to add method of parent
toolkit	toolkit
obj	receiver object
index	ignored
drop	if TRUE return a character, else a Date class object.

### Value

Returns an object of class GCalendar for which the following methods are overridden:

1. svalue get the date
2. svalue<- set the date

The change handler is inherited from [gedit](#)

If drop=TRUE a character string, else a Date class object.

---

gcheckbox	<i>constructor for checkbox widget</i>
-----------	--

---

### Description

A checkbox widget is used to toggle the state of a labeled boolean variable. The main property of this widget is that state, not the label. This variable may be proxied in the usual way – with a box that indicates or check if TRUE – or through a toggle button.

The change handler for GCheckbox is called when the value toggles. You can inspect the current value in the callback to have an action based on the state.

The object state is referred to by svalue as a logical (TRUE for checked). The svalue<- method ensures the value is a logical vector of length 1.

The item to select is referred to by the [] method, with only the first element being used.

**Usage**

```

gcheckbox(
  text = "",
  checked = FALSE,
  use.togglebutton = FALSE,
  handler = NULL,
  action = NULL,
  container = NULL,
  ...,
  toolkit = guiToolkit()
)

.gcheckbox(
  toolkit,
  text,
  checked = FALSE,
  use.togglebutton = FALSE,
  handler = NULL,
  action = NULL,
  container = NULL,
  ...
)

## S3 method for class 'GCheckbox'
addHandlerChanged(obj, handler, action = NULL, ...)

## S3 replacement method for class 'GCheckbox'
svalue(obj, index=NULL, ...) <- value

## S3 replacement method for class 'GCheckbox'
x[i, j, ...] <- value

```

**Arguments**

text	label text
checked	is button selected
use.togglebutton	Use a toggle button (shows depressed) not a check box
handler	Callback called when toggle is changed.
action	passed to handler
container	parent container
...	passed to add method of container
toolkit	toolkit
obj	receiver object
index	ignored. Input is coerced to logical.
value	assignment value

x	checkbox object
i	item index
j	ignored

**Value**

Returns an object of class GCheckbox.

**Note**

The value is coerced to character, then only first element used for checkbox label

**Examples**

```
if(interactive()) {
  w <- gwindow("Selection widgets")
  g <- gvbox(cont=w)

  fl <- gformlayout(cont=g)
  gcheckbox("checkbox", checked=TRUE, cont=f1, label="checkbox")
  gradio(state.name[1:4], selected=2, horizontal=TRUE, cont=f1, label="gradio")
  gcheckboxgroup(state.name[1:4], horizontal=FALSE, cont=f1, label="checkbox group")

  bg <- ggroup(cont=g)
  gbutton("ok", cont=bg, handler=function(h,...) print(sapply(fl$children, svalue)))
}
```

---

gcheckboxgroup	<i>Constructor for checkbox group. A linked group of checkboxes, but not exclusive.</i>
----------------	---

---

**Description**

Change handler for a GCheckboxGroup is called when any of the checkboxes changes state.

The svalue methods refer to the selected values. By default these are the item values, coerced to character. When index=TRUE is specified, then the index is returned as an integer vector. For setting, one may also use a vector of logicals (which is recycled) for the index.

**Usage**

```
gcheckboxgroup(
  items,
  checked = FALSE,
  horizontal = FALSE,
  use.table = FALSE,
```

```

    handler = NULL,
    action = NULL,
    container = NULL,
    ...,
    toolkit = guiToolkit()
)

.gcheckboxgroup(
    toolkit,
    items,
    checked = FALSE,
    horizontal = FALSE,
    use.table = FALSE,
    handler = NULL,
    action = NULL,
    container = NULL,
    ...
)

## S3 method for class 'GCheckboxGroup'
addHandlerChanged(obj, handler, action = NULL, ...)

## S3 method for class 'GCheckboxGroup'
svalue(obj, index = NULL, drop = NULL, ...)

```

### Arguments

items	checkbox labels
checked	logical. Are values checked
horizontal	logical. If true displayed horizontally, else vertically
use.table	logical. If supported, and TRUE then uses a table widget with scrollbars
handler	<p>A handler assigned to the default change signal. Handlers are called when some event triggers a widget to emit a signal. For each widget some default signal is assumed, and handlers may be assigned to that through <code>addHandlerChanged</code> or at construction time. Handlers are functions whose first argument, <code>h</code> in the documentation, is a list with atleast two components <code>obj</code>, referring to the object emitting the signal and <code>action</code>, which passes in user-specified data to parameterize the function call.</p> <p>Handlers may also be added via <code>addHandlerXXX</code> methods for the widgets, where <code>XXX</code> indicates the signal, with a default signal mapped to <code>addHandlerChanged</code> (cf. <a href="#">addHandler</a> for a listing). These methods pass back a handler ID that can be used with <code>blockHandler</code> and <code>unblockHandler</code> to suppress temporarily the calling of the handler.</p>
action	User supplied data passed to the handler when it is called
container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <code>gWidgets2tcltk</code> or <code>gWidgets2WWW2</code> .)

...	These values are passed to the add method of the parent container. Examples of values are expand, fill, and anchor, although they're not always supported by a given widget. For more details see <a href="#">add</a> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with gaction and the dialogs.
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <a href="#">guiToolkit</a> .
obj	receiver object
index	NULL or logical. If TRUE and widget supports it an index, instead of a value will be returned.
drop	NULL or logical. If widget supports it, drop will work as it does in a data frame or perhaps someother means.

## Value

Returns an object of class GCheckboxGroup for which the following methods are overridden:

- `svalue` Return the selected values or an empty character vector. If `index=TRUE`, returns indices of selected values.
- `svalue<-` Set the selected values one of three ways: by label name, by a logical variable indicating which are selected (if ambiguous, logical wins), if `index=TRUE` by the indices to select.
- `[]` returns labels
- `[<-` set the label values. Should be able to shorten or lengthen list

## Examples

```
if(interactive()) {
  w <- gwindow("Selection widgets")
  g <- gvbox(cont=w)

  fl <- gformlayout(cont=g)
  gcheckbox("checkbox", checked=TRUE, cont=f1, label="checkbox")
  gradio(state.name[1:4], selected=2, horizontal=TRUE, cont=f1, label="gradio")
  gcheckboxgroup(state.name[1:4], horizontal=FALSE, cont=f1, label="checkbox group")

  bg <- ggroup(cont=g)
  gbutton("ok", cont=bg, handler=function(h,...) print(sapply(fl$children, svalue)))
}
```

gcombobox

*constructor for a combobox***Description**

A combobox can be either a drop down list (`editable=FALSE`), or a drop-down list and edit area (a combobox).

Non exported helper function to coerce items into a data frame. First column contains the values, second stock icons, third tooltips

Ensure that value is a data frame. One can pass a vector or a one-column data frame to indicate the possible values for selection, a second column is used for an icons (if possible), a third for a tooltip (if possible).

Change handler for a non-editable combobox is called when a new value is selected. For editable comboboxes, the handler is also called when the text entry box is activated.

The `svalue` method for a combobox object refers to its main property, the selected value. When `index=FALSE` (or `NULL`) the value is returned. If `index=TRUE` the index of the object within the set of items is used.

**Usage**

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

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

```
gdroplist(...)
```

```

.make_gcombobox_items(value)

## S3 replacement method for class 'GComboBox'
x[i , j, ...] <- value

## S3 method for class 'GComboBox'
addHandlerChanged(obj, handler, action = NULL, ...)

## S3 method for class 'GComboBox'
svalue(obj, index = NULL, drop = NULL, ...)

```

### Arguments

items	Items to select from. A vector or a data frame. If a data frame, then first column is values. Second is optional, but can specify a stock icon name, third is optional and can be used to specify a tooltip. These may not be supported in all toolkits.
selected	integer. Which item (by index) is selected. Use -1 for no selection
editable	logical. Is user allowed to edit value
coerce.with	A function of function name to be called before selected value is returned by svalue
handler	Called when combobox value is changed.
action	passed to handler
container	parent container
...	passed to parent container's add method
toolkit	toolkit
value	new items for selection
x	combobox object
i	item index
j	ignored
obj	object receiving event and emitting a signal to the handler
index	NULL or logical. If TRUE and widget supports it an index, instead of a value will be returned.
drop	NULL or logical. If widget supports it, drop will work as it does in a data frame or perhaps someother means.

### Value

Returns an object of class GComboBox for which the following methods are overridden:

1. svalue Return selected value by name or (if index=TRUE by index). The latter only if editable=FALSE.
2. svalue<- Set the selected value by value or if index=TRUE by index.
3. [ return items to select from
4. [<- Set items to select from.

---

`gconfirm`*Constructor for modal dialog to get confirmation*

---

**Description**

Constructor for modal dialog to get confirmation  
generic for toolkit dispatch

**Usage**

```
gconfirm(  
    msg,  
    title = "Confirm",  
    icon = c("info", "warning", "error", "question"),  
    parent = NULL,  
    ...,  
    toolkit = guiToolkit()  
)  
  
.gconfirm(  
    toolkit,  
    msg,  
    title = "Confirm",  
    icon = c("info", "warning", "error", "question"),  
    parent = NULL,  
    ...  
)
```

**Arguments**

<code>msg</code>	Character. Message to display.
<code>title</code>	Character. Title of window
<code>icon</code>	which icon to display
<code>parent</code>	gives hint as to where to place dialog
<code>...</code>	ignored
<code>toolkit</code>	toolkit

**Value**

logical indicating confirmation

**See Also**

[gmessage](#), [gconfirm](#), [gbasicdialog](#), [galert](#)

---

gcontainer	<i>Common parts of a container widget</i>
------------	---

---

**Description**

Used as template for documentation

**Usage**

```
gcontainer(container = NULL, ..., toolkit = guiToolkit())
```

**Arguments**

container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <b>gWidgets2tcltk</b> or <b>gWidgets2WWW2</b> .)
...	These values are passed to the add method of the parent container, and occasionally have been used to sneak in hidden arguments to toolkit implementations.
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <a href="#">guiToolkit</a> .

---

gdf	<i>Constructor for a data frame editor</i>
-----	--

---

**Description**

Implementation varies wildly, but should provide at minimum functionality of `edit.data.frame`. A single mouse click on a cell should select that cell, a double click should initiate editing of that cell.

Assign handler to be called when a cell, row or column changes

For `gdf` `svalue` refers to the selected values.

`visible` is used to refer to which rows are being shown.

**Usage**

```
gdf(
  items = NULL,
  handler = NULL,
  action = NULL,
  container = NULL,
  ...,
  toolkit = guiToolkit()
)
```

```

.gdf(
  toolkit,
  items = NULL,
  handler = NULL,
  action = NULL,
  container = NULL,
  ...
)

## S3 method for class 'Gdf'
addHandlerChanged(obj, handler, action = NULL, ...)

## S3 method for class 'Gdf'
svalue(obj, index = NULL, drop = TRUE, ...)

## S3 replacement method for class 'Gdf'
visible(obj) <- value

```

### Arguments

items	data frame to edit
handler	called on cell change
action	passed to handler
container	parent container
...	passed to container's add method
toolkit	toolkit
obj	object receiving event and emitting a signal to the handler
index	NULL or logical. If TRUE and widget supports it an index, instead of a value will be returned.
drop	NULL or logical. If widget supports it, drop will work as it does in a data frame or perhaps someother means.
value	value to assign for selection or property

### Details

Contents of the data frame can be accessed via `[]` and manipulated with `[<-`.

The `save_data` reference class method can be called to save the data into a variable with the specified name.

example in `inst/examples/ex-gdf.R`

### Value

An object of class `gDf`.

---

gdfnotebook	<i>A notebook container for many gdf instances</i>
-------------	--

---

**Description**

A notebook container for many gdf instances  
 S3 generic whose methods are implemented in the toolkit packages

**Usage**

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

**Arguments**

items	data frame for initial page, when given
container	parent container
...	passed to add method of parent container
toolkit	toolkit

---

gedit	<i>Single line text edit constructor</i>
-------	--

---

**Description**

The default change handler is called when the return key is pressed. It can be useful to also call a handler when the widget loses focus. For that, the `addHandlerBlur` method is of use. (This was the default, but is now not, as it was hard to decouple the two when that was desirable.)

The default change handler call is when the user activates the entry by pressing the enter key. Other possible events to consider are covered by: `addhandlerBlur` (when the widget loses focuses) and `addHandlerKeystroke` (called after each keystroke). For the latter, if the toolkit supports it, the handler's first argument has a component key passing back the keystroke information.

The `svalue` method for a edit object refers to its main property, the text in the box.

**Usage**

```
gedit(
  text = "",
  width = 25,
  coerce.with = NULL,
  initial.msg = "",
  handler = NULL,
```

```

    action = NULL,
    container = NULL,
    ...,
    toolkit = guiToolkit()
)

.gedit(
  toolkit,
  text = "",
  width = 25,
  coerce.with = NULL,
  initial.msg = "",
  handler = NULL,
  action = NULL,
  container = NULL,
  ...
)

## S3 method for class 'GEdit'
addHandlerChanged(obj, handler, action = NULL, ...)

## S3 method for class 'GEdit'
svalue(obj, index = NULL, drop = NULL, ...)

```

### Arguments

text	initial text
width	number of characters
coerce.with	A function or name of function to coerce value with before returning by svalue
initial.msg	If no initial text is given but an initial message is, then this message is displayed until the widget receives the focus
handler	Change handler. Called when return key is hit. Use addHandleBlur to add a handler when the widget loses focus, such as through tab-key navigation.
action	passed to handler
container	parent container
...	passed to add method of parent
toolkit	toolkit
obj	object receiving event and emitting a signal to the handler
index	NULL or logical. If TRUE and widget supports it an index, instead of a value will be returned.
drop	NULL or logical. If widget supports it, drop will work as it does in a data frame or perhaps someother means.

### Value

An object of class `GEdit`. This has sub-classed methods:

1. `svalue` to retrieve the text
2. `svalue<-` to set the text
3. `[]` to get the autocomplete values
4. `[-]` Character. To set autocomplete values
5. `visible<-` to specify a character to display instead of text (for passwords)

---

<code>getToolkitWidget</code>	<i>Get underlying toolkit widget</i>
-------------------------------	--------------------------------------

---

### Description

At times a user may wish to access the underlying toolkit widget. Although this is not cross-platform, one often has access to many more methods of the object, than through those provided by `gWidgets`.

For `GWindow`, the block is `NULL`

### Usage

```
getToolkitWidget(obj)

## Default S3 method:
getToolkitWidget(obj)

getWidget(obj)

## S3 method for class 'GComponent'
getWidget(obj)

getBlock(obj)

## S3 method for class 'GComponent'
getBlock(obj)

## S3 method for class 'GWindow'
getBlock(obj)

getTopLevel(obj)

## S3 method for class 'GComponent'
getTopLevel(obj)
```

### Arguments

`obj`                    `object`

---

getWithDefault	<i>Return x unless NULL, NA, length 0, ..., in which case we give default</i>
----------------	---

---

**Description**

Return x unless NULL, NA, length 0, ..., in which case we give default

**Usage**

```
getWithDefault(x, default)
```

**Arguments**

x	value to return or its default
default	default value

**Value**

x or default

---

get_index_in_list	<i>get index of element of list</i>
-------------------	-------------------------------------

---

**Description**

Like match, but works with a list

**Usage**

```
get_index_in_list(lst, ele)
```

**Arguments**

lst	a list to search through
ele	element of list

**Value**

returns index of element or integer(0)

---

 get\_object\_from\_string

*Get an object from an environment specified by a string.*


---

**Description**

Get an object from an environment specified by a string.

**Usage**

```
get_object_from_string(value, envir = .GlobalEnv)
```

**Arguments**

value	A single character value dispatches to get. For a length 2 or more, then assumes object is recursive and extracts named components
envir	environment to look for values.

**Value**

the object or an error

---

 gexpandgroup

*Constructor of box container widget with disclosure trigger and label*


---

**Description**

For gexpandgroup, the visible assignment method is overridden to change the disclosure state

The change handler for a expandGroup is called when the group changes visibility

**Usage**

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

```
.gexpandgroup(
  toolkit,
```

```

    text = "",
    markup = FALSE,
    horizontal = TRUE,
    handler = NULL,
    action = NULL,
    container = NULL,
    ...
)

## S3 replacement method for class 'GExpandGroup'
visible(obj) <- value

## S3 method for class 'GExpandGroup'
addHandlerChanged(obj, handler, action = NULL, ...)

```

### Arguments

text	Label text
markup	logical. Does text have markup? (Toolkit dependent: only implemented for RGtk2, in qtbase one can pass HTML formatted text)
horizontal	horizontal (TRUE) or vertical packing.
handler	handler called when state is toggled
action	passed to handler
container	parent container
...	passed to parent's add method
toolkit	toolkit
obj	object receiving event and emitting a signal to the handler
value	logical. If TRUE show, FALSE hide.

### Value

An object of class GExpandGroup inheriting from GFrame

### See Also

[ggroup](#) and [gframe](#)

### Examples

```

if(interactive()) {
  w <- gwindow("Box containers")
  g <- gvbox(cont=w) # ggroup(horizontal=FALSE, ...)
  nb <- gnotebook(cont=g); gbutton("one", label="one", cont=nb)
  gframe("Frame", cont=g)
  pg <- gpanedgroup(cont=g);
  gbutton("one", cont=pg);
  gbutton("two", cont=pg)
}

```

```
    eg <- gexpandgroup(cont=g, horizontal=FALSE);
    glabel("Click above to hide", cont=eg);
    gbutton("one", cont=eg);
    gbutton("two", cont=eg)
  }
```

---

gfile

*dialog for file and directory selection*

---

## Description

Basically an entry box instance with a button to initiate gfile.

## Usage

```
gfile(
  text = "",
  type = c("open", "save", "selectdir"),
  initial.filename = NULL,
  initial.dir = getwd(),
  filter = list(),
  multi = FALSE,
  ...,
  toolkit = guiToolkit()
)
```

```
.gfile(
  toolkit,
  text = "",
  type = c("open", "save", "selectdir"),
  initial.filename = NULL,
  initial.dir = getwd(),
  filter = list(),
  multi = FALSE,
  ...
)
```

```
gfilebrowse(
  text = "Select a file...",
  type = c("open", "save", "selectdir"),
  initial.filename = NULL,
  initial.dir = getwd(),
  filter = list(),
  quote = TRUE,
  handler = NULL,
  action = NULL,
)
```

```

    container = NULL,
    ...,
    toolkit = guiToolkit()
)

.gfilebrowse(
  toolkit,
  text = "Select a file...",
  type = c("open", "save", "selectdir"),
  initial.filename = NULL,
  initial.dir = getwd(),
  filter = list(),
  quote = TRUE,
  handler = NULL,
  action = NULL,
  container = NULL,
  ...
)

```

### Arguments

<code>text</code>	initial text
<code>type</code>	type of browser: to open a file, to save a file or to select a directory
<code>initial.filename</code>	Suggested file name
<code>initial.dir</code>	initial directory. If a filename is given, and is not an absolute name, this will be prepended. If filename given initial directory will be taken from that.
<code>filter</code>	A filter specification. This can be a named character vector of file extensions or something toolkit specific. Here are some examples: <b>character</b> <code>c("csv"="csv", "txt"="txt")</code> <b>RGtk2</b> Something like <pre>list("All files" = list(patterns = c("*")),      "R files" = list(patterns = c("*.R", "*.Rdata")),      "text files" = list(mime.types = c("text/plain"))) )</pre> <b>tcltk</b> <b>Qt</b>
<code>multi</code>	Logical. Allow multiple files to be selected?
<code>...</code>	These values are passed to the <code>add</code> method of the parent container. Examples of values are <code>expand</code> , <code>fill</code> , and <code>anchor</code> , although they're not always supported by a given widget. For more details see <a href="#">add</a> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with <code>gaction</code> and the dialogs.

toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <code>guiToolkit</code> .
quote	quote output
handler	<p>A handler assigned to the default change signal. Handlers are called when some event triggers a widget to emit a signal. For each widget some default signal is assumed, and handlers may be assigned to that through <code>addHandlerChanged</code> or at construction time. Handlers are functions whose first argument, <code>h</code> in the documentation, is a list with atleast two components <code>obj</code>, referring to the object emitting the signal and <code>action</code>, which passes in user-specified data to parameterize the function call.</p> <p>Handlers may also be added via <code>addHandlerXXX</code> methods for the widgets, where <code>XXX</code> indicates the signal, with a default signal mapped to <code>addHandlerChanged</code> (cf. <code>addHandler</code> for a listing). These methods pass back a handler ID that can be used with <code>blockHandler</code> and <code>unblockHandler</code> to suppress temporarily the calling of the handler.</p>
action	User supplied data passed to the handler when it is called
container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <code>gWidgets2tcltk</code> or <code>gWidgets2WWW2</code> .)

**Value**

returns filename(s) or character(0) if no selection.

Returns an object of class `gFilebrowse`. This should inherit the methods of `gedit` instances.

---

gfilter	<i>A widget for filtering a data frame</i>
---------	--

---

**Description**

This widget provides a simple means to subset, or filter, a data frame.

The `svalue` method for a filter object returns a logical containing which rows are selected. There is no assignment method.

**Usage**

```
gfilter(
  DF,
  allow.edit = TRUE,
  initial.vars = NULL,
  handler = NULL,
  action = NULL,
  container = NULL,
  ...,
  toolkit = guiToolkit()
```

```

)

.gfilter(
  toolkit,
  DF,
  allow.edit = TRUE,
  initial.vars = NULL,
  handler = NULL,
  action = NULL,
  container = NULL,
  ...
)

## S3 method for class 'GFilter'
svalue(obj, index = NULL, drop = NULL, ...)

## S3 method for class 'GFilter'
x[i, j, ..., drop = TRUE]

## Default S3 method:
.gfilter(
  toolkit = guiToolkit(),
  DF,
  allow.edit = TRUE,
  initial.vars = NULL,
  handler = NULL,
  action = NULL,
  container = NULL,
  ...
)

```

## Arguments

DF	a data frame or Gdf instance to look variables up within.
allow.edit	logical. If TRUE a user may add new variables to filter by. If FALSE, then one should specify the variables a user can filter by to <code>initial.vars</code> .
initial.vars	When given, this is a data frame whose first column specifies the variables within DF to filter by and whose second column indicates the type of filter desired. The available types are <code>single</code> to select one from many, <code>multiple</code> , for multiple selection; and <code>range</code> , to specify a from and to value.
handler	A handler assigned to the default change signal. Handlers are called when some event triggers a widget to emit a signal. For each widget some default signal is assumed, and handlers may be assigned to that through <code>addHandlerChanged</code> or at construction time. Handlers are functions whose first argument, <code>h</code> in the documentation, is a list with atleast two components <code>obj</code> , referring to the object emitting the signal and <code>action</code> , which passes in user-specified data to parameterize the function call.

Handlers may also be added via `addHandlerXXX` methods for the widgets, where `XXX` indicates the signal, with a default signal mapped to `addHandlerChanged` (cf. [addHandler](#) for a listing). These methods pass back a handler ID that can be used with `blockHandler` and `unblockHandler` to suppress temporarily the calling of the handler.

<code>action</code>	User supplied data passed to the handler when it is called
<code>container</code>	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <code>gWidgets2tcltk</code> or <code>gWidgets2WWW2</code> .)
<code>...</code>	dots argument
<code>toolkit</code>	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <code>guiToolkit</code> .
<code>obj</code>	object of method call
<code>index</code>	NULL or logical. If TRUE and widget supports it an index, instead of a value will be returned.
<code>drop</code>	NULL or logical. If widget supports it, drop will work as it does in a data frame or perhaps someother means.
<code>x</code>	the GFilter object
<code>i</code>	passed to <code>get_items</code>
<code>j</code>	passed to <code>get_items</code>

### Value

returns GFilter object

### Examples

```
## Not run:
DF <- mtcars[, c("mpg", "cyl", "hp", "am", "wt")]
w <- gwindow("Example of gfilter", visible=FALSE)
pg <- ggroup(container=w)
df <- gtable(DF, container=pg)
a <- gfilter(df, initial.vars=data.frame(names(DF), names(DF),
                                     c("single", "multiple", "range", "single", "range"),
                                     stringsAsFactors=FALSE),
            allow.edit=TRUE,
            container=pg,
            handler=function(h,...) {
              visible(df) <- h$obj$get_value()
            }
            )
size(w) <- c(600, 600)
visible(w) <- TRUE

## End(Not run)
```

gformlayout

*A form layout container***Description**

This convenience container is basically a simpler form of `gLayout` to be used to layout two columns forms with a label on the left. The label can be passed in to the `add` method of the container as is done with notebook labels

The `svalue` method for `GFormLayout` returns a list of values created by calling `svalue` on each child. The returned list is named by the corresponding labels.

**Usage**

```
gformlayout(
  align = c("default", "left", "center"),
  spacing = 5,
  container = NULL,
  ...,
  toolkit = guiToolkit()
)

.gformlayout(toolkit, align = "left", spacing = 5, container = NULL, ...)

## S3 method for class 'GFormLayout'
svalue(obj, index = NULL, drop = NULL, ...)
```

**Arguments**

<code>align</code>	alignment of label. Left justify or center balance. Leave as "default" for underlying toolkit default.
<code>spacing</code>	spacing between columns
<code>container</code>	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <code>gWidgets2tcltk</code> or <code>gWidgets2WWW2</code> .)
<code>...</code>	These values are passed to the <code>add</code> method of the parent container. Examples of values are <code>expand</code> , <code>fill</code> , and <code>anchor</code> , although they're not always supported by a given widget. For more details see <a href="#">add</a> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with <code>gaction</code> and the dialogs.
<code>toolkit</code>	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <code>guiToolkit</code> .
<code>obj</code>	object of method call

index	NULL or logical. If TRUE and widget supports it an index, instead of a value will be returned.
drop	NULL or logical. If widget supports it, drop will work as it does in a data frame or perhaps someother means.

### Examples

```
## Not run:
w <- gwindow("gformlayout", visible=FALSE)
g <- gvbox(container=w)

flyt <- gformlayout(container=g)
gedit("", label="Name:", container=flyt)
gedit("", label="Rank:", container=flyt)
gedit("", label="Serial No.:", container=flyt)

b <- gbutton("Show me", container=g, handler=function(h,...) {
  print(svalue(flyt))
})

addSpring(g) ## better with Qt, else flyt expands to fill.
visible(w) <- TRUE

## End(Not run)
```

---

gframe

*Constructor for framed box container with label*


---

### Description

The framed box container inherits from ggroup. The main addition is a label, which is accessed via the name method.

### Usage

```
gframe(
  text = "",
  markup = FALSE,
  pos = 0,
  horizontal = TRUE,
  spacing = 5,
  container = NULL,
  ...,
  toolkit = guiToolkit()
)

.gframe(
  toolkit,
```

```

text = "",
markup = FALSE,
pos = 0,
horizontal = TRUE,
spacing = 5,
container = NULL,
...
)

```

### Arguments

text	frame label
markup	does label use markup (toolkit specific)
pos	position of label: 0=left, 1=right, some toolkit allow values in between
horizontal	logical. If TRUE, left to right layout, otherwise top to bottom
spacing	spacing around widget
container	parent container
...	passed through
toolkit	toolkit

### Note

to include a scrollwindow, place a ggroup within this window.

### See Also

[ggroup](#) and [gexpandgroup](#)

### Examples

```

## Not run:
w <- gwindow("gformlayout", visible=FALSE)
f <- gframe("frame", horizontal=FALSE, container=w)
glabel("Lorem ipsum dolor sit amet, \nconsectetur adipiscing elit.", container=f)
gbutton("change name", container=f, handler=function(h,...) {
  names(f) <- "new name"
})
visible(w) <- TRUE

## End(Not run)

```

---

`ggraphics`*Constructor for an embeddable graphics device*

---

## Description

Some toolkits provide an embeddable graphics device. When this is the case, this widget provides same.

## Usage

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

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

## Arguments

<code>width</code>	width of device (pixels)
<code>height</code>	hieght of widget (pixels)
<code>dpi</code>	dots per inch
<code>ps</code>	pointsize
<code>handler</code>	A handler assigned to the default change signal. Handlers are called when some event triggers a widget to emit a signal. For each widget some default signal is assumed, and handlers may be assigned to that through <code>addHandlerChanged</code> or at construction time. Handlers are functions whose first argument, <code>h</code> in the documentation, is a list with atleast two components <code>obj</code> , referring to the object

emitting the signal and action, which passes in user-specified data to parameterize the function call.

Handlers may also be added via `addHandlerXXX` methods for the widgets, where `XXX` indicates the signal, with a default signal mapped to `addHandlerChanged` (cf. [addHandler](#) for a listing). These methods pass back a handler ID that can be used with `blockHandler` and `unblockHandler` to suppress temporarily the calling of the handler.

action	User supplied data passed to the handler when it is called
container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <code>gWidgets2tcltk</code> or <code>gWidgets2WWW2</code> .)
...	These values are passed to the <code>add</code> method of the parent container. Examples of values are <code>expand</code> , <code>fill</code> , and <code>anchor</code> , although they're not always supported by a given widget. For more details see <a href="#">add</a> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with <code>gaction</code> and the dialogs.
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <code>guiToolkit</code> .

## Examples

```
## Not run:
## This shows how to use the device within a notebook

w <- gwindow("notebook example")
nb <- gnotebook(cont=w)

devs <- lapply(1:5, function(i) ggraphics(cont=nb, label=as.character(i)))

addHandlerChanged(nb, handler=function(h,...) {
  ## Tricky part is svalue(h$obj) is not the new page number -- but old
  ## so we use the pageno component here
  gg <- h$obj[h$page]
  visible(gg) <- TRUE
})

## End(Not run)
```

**Description**

A notebook widget holding plot devices

S3 generic whose methods are implemented in the toolkit packages

toolkit implementation

**Usage**

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

```
.ggraphicsnotebook(toolkit, width, height, dpi, container, ...)
```

```
## Default S3 method:
```

```
.ggraphicsnotebook(toolkit, width, height, dpi, container, ...)
```

**Arguments**

width            width in pixels

height           height in pixels

dpi               screen resolution

container        A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. **gWidgets2tcltk** or **gWidgets2WWW2**.)

...                These values are passed to the add method of the parent container. Examples of values are `expand`, `fill`, and `anchor`, although they're not always supported by a given widget. For more details see [add](#). Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with `gaction` and the dialogs.

toolkit           Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through `guiToolkit`.

ggroup

*Basic box container***Description**

The `svalue` method refers to the main property of the box container, its spacing. There are generally two types of spacing: padding around border of the box and spacing between each child that is packed in. The spacing here is the between-child-component spacing. The reference class method `set_borderwidth` can be used for the other.

Avoids need to type `horizontal=FALSE`

**Usage**

```
ggroup(
  horizontal = TRUE,
  spacing = 5,
  use.scrollwindow = FALSE,
  container = NULL,
  ...,
  toolkit = guiToolkit()
)

.ggroup(
  toolkit,
  horizontal = TRUE,
  spacing = 5,
  use.scrollwindow = FALSE,
  container = NULL,
  ...
)

## S3 replacement method for class 'GGroup'
svalue(obj, index=TRUE, ...) <- value

gvbox(
  spacing = 5,
  use.scrollwindow = FALSE,
  container = NULL,
  ...,
  toolkit = guiToolkit()
)
```

**Arguments**

<code>horizontal</code>	logical. If TRUE, left to right layout, otherwise top to bottom
<code>spacing</code>	spacing around widget

<code>use.scrollwindow</code>	logical. Either TRUE, "TRUE", FALSE, "FALSE", "y", or "x". For all toolkits a non-FALSE value will place the child components into a scrollable container. For some toolkits this will only be in the direction of packing. If the toolkit allows it (RGtk2), then values of "x" or "y" can be used to override the default scrolling directions. A box container with scrollwindows should have its size set either directly or through packing with <code>expand=TRUE</code> as its size request will not reflect the size of its child components.
<code>container</code>	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <code>gWidgets2tcltk</code> or <code>gWidgets2WWW2</code> .)
<code>...</code>	These values are passed to the <code>add</code> method of the parent container. Examples of values are <code>expand</code> , <code>fill</code> , and <code>anchor</code> , although they're not always supported by a given widget. For more details see <a href="#">add</a> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with <code>gaction</code> and the dialogs.
<code>toolkit</code>	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <code>guiToolkit</code> .
<code>obj</code>	GGroup object
<code>index</code>	ignored
<code>value</code>	value (in pixels) for between child spacing

### Details

Child components are typically added to a box container through the child components constructor. The argument `expand`, `fill`, and `anchor` determine how the child is positioned within the container.

### Value

a GGroup instance.  
a GGroup instance with vertical packing.

### See Also

[gframe](#) and [gexpandgroup](#)

### Examples

```
if(interactive()) {
  w <- gwindow("Box containers")
  g <- gvbox(cont=w) # ggroup(horizontal=FALSE, ...)
  nb <- gnobook(cont=g); gbutton("one", label="one", cont=nb)
  gframe("Frame", cont=g)
  pg <- gpanedgroup(cont=g); gbutton("one", cont=pg); gbutton("two", cont=pg)
}
```

ghtml

*Widget for HTML display***Description**

This widget, when supported by the toolkit (not **gWidgets2RGtk2** and **gWidgets2tcltk**) provides a simple means to display HTML formatted text.

Use to update displayed content. Value is HTML fragment or url

**Usage**

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

```
.ghtml(toolkit, x, container = NULL, ...)
```

```
## S3 replacement method for class 'GHtml'
svalue(obj, index=TRUE, ...) <- value
```

**Arguments**

x	url or character vector of HTML formatted text. URLs marked by "http://" prefix
container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <b>gWidgets2tcltk</b> or <b>gWidgets2WWW2</b> .)
...	These values are passed to the add method of the parent container. Examples of values are <code>expand</code> , <code>fill</code> , and <code>anchor</code> , although they're not always supported by a given widget. For more details see <a href="#">add</a> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with <code>gaction</code> and the dialogs.
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <code>guiToolkit</code> .
obj	object of method call
index	NULL or logical. If TRUE and widget supports it an index, instead of a value will be returned.
value	value to assign for selection or property

**Value**

a GHtml instance.

**Author(s)**

john verzani

---

gimage	<i>A widget for displaying an image file</i>
--------	--

---

**Description**

A widget for displaying an image file  
generic for toolkit dispatch

**Usage**

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

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

**Arguments**

filename	basename of file
dirname	dirname of file
stock.id	stock id of icon (if non NULL)
size	size of icon when a stock id (toolkit dependent)
handler	handler if image is clicked on.
action	passed to handler
container	parent container
...	passed to add method of parent
toolkit	toolkit

ginput

*Constructor for modal dialog to collect a line of text*

---

**Description**

Constructor for modal dialog to collect a line of text  
generic for toolkit dispatch

**Usage**

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

**Arguments**

msg	Character. Message to display.
text	Character. Initial text
title	Character. Title of window
icon	which icon to display
parent	gives hint as to where to place dialog
...	ignored
toolkit	toolkit

**Value**

value typed into box or character(0)

**See Also**

[gmessage](#), [gconfirm](#), [gbasicdialog](#), [galert](#)

---

glabel	<i>Basic label widget</i>
--------	---------------------------

---

**Description**

The basic label widget allows one to label areas of a GUI using text. The most common use would be to label fields in a form. For **gWidgets2** labels may be editable or responsive to mouse clicks, although it is the author's experience that such uses are not expected by the end user.

The `svalue` methods refer to the main property of the label, its text.

**Usage**

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

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

```
## S3 replacement method for class 'GLabel'
svalue(obj, index=TRUE, ...) <- value
```

**Arguments**

text	character. Collapsed using a newline to a single string.
markup	logical. If toolkit supports markup, this indicates it will be used. It is suggested that the <code>font&lt;-</code> method be used, though for <b>gWidgets2Qt</b> markup is more convenient.

editable	If TRUE, then clicking on label will enable user-editing of the text.
handler	optional handler. If given, added through addHandlerChanged. Overridden if editable=TRUE.
action	passed to handler through action component of first argument of handler. For buttons, this may also be a GAction instance.
container	parent container (Optional for some toolkits, but not all).
...	passed to add method of parent container
toolkit	toolkit instance
obj	object of method call
index	NULL or logical. If TRUE and widget supports it an index, instead of a value will be returned.
value	value to assign for selection or property

**Value**

a GLabel instance. While this object has its own (reference) methods, one primarily interacts with it through S3 methods defined within the package.

**Author(s)**

john verzani

**Examples**

```
## Not run:
w <- gwindow("gformlayout", visible=FALSE)
g <- gvbox(container=w)
g$set_borderwidth(10)

l1 <- glabel("static label", container=g)
l2 <- glabel("bold label", container=g)
font(l2) <- list(weight="bold")
l3 <- glabel("editable label. Click me", editable=TRUE, container=g)

visible(w) <- TRUE

## End(Not run)
```

**Description**

The grid layout container uses matrix notation to position its child components. This allows one to align widgets both horizontally and vertically, as desired. There is some support for matrix methods, such as `dim` and `[]` to reference the child objects.

The `[]` method for the grid layout allows one to reference the child objects by index. The return value is non standard. It may be the item, a list (if one dimensional) or an array. The list format is convenient to refer to all the child objects in a column.

The matrix notation allows for spanning of multiple rows and or columns, but no holes. The `...` argument is used to pass in values for `expand`, `fill`, `anchor` (see the `add` method of `ggroup`) for their meaning).

**Usage**

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

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

```
## S3 method for class 'GLayout'
x[i, j, ..., drop = TRUE]
```

```
## S3 replacement method for class 'GLayout'
x[i ,j, ...] <- value
```

**Arguments**

homogeneous	are cells all the same size
spacing	between cell spacing
container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <b>gWidgets2tcltk</b> or <b>gWidgets2WWW2</b> .)
...	These values are passed to the <code>add</code> method of the parent container. Examples of values are <code>expand</code> , <code>fill</code> , and <code>anchor</code> , although they're not always supported by a given widget. For more details see <a href="#">add</a> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with <code>gaction</code> and the <code>dialogs</code> .
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will look up the toolkit through <code>guiToolkit</code> .
x	object

i	row index
j	column index
drop	drop return type?
value	constructor for a widget using this object as the parent container

### See Also

[gformlayout](#) for a more convenient means to layout forms.

### Examples

```
## Not run:

w <- gwindow("glayout example", visible=FALSE)
g <- gvbox(container=w)
lyt <- glayout(container=g)
lyt[1,1] <- "a label"
lyt[1,2] <- gedit("A widget", container=lyt)
lyt[2, 1:2] <- gcombobox(state.name, cont=lyt)
g1 <- gggroup(container=g)
addSpring(g1)
gbutton("values", container=g1, handler=function(h, ...) {
  print(sapply(lyt[,2], svalue))
})
visible(w) <- TRUE

## End(Not run)
```

---

gmenu

*menu constructor, main and popup*

---

### Description

A menu may be viewed as a heirarchical collection of buttons, each invoked by clicking on the button. These buttons are exposed through submenus. More generally, a widget may replace the button. This widget intends to support buttons (gactions), separators (gseparator), radio button (gradio) and checkbutton (gcheckbox), but this may be toolkit independent. When using a radio button or checkbox, one should pass in a parent argument to the constructor – not a container.

For a menubar, svalue returns the list of action items etc. that defined the menubar. This can be useful to access the underlying item being proxied. (For gaction items the enabled<- method may be used on the item, but this may not extend to gradio and gcheckbox items)

for a menubar, svalue<- replaces the menubar items with new ones specified by value.

**Usage**

```

gmenu(menu.list, popup = FALSE, container = NULL, ..., toolkit = guiToolkit())

.gmenu(toolkit, menu.list = list(), popup = FALSE, container = NULL, ...)

## S3 method for class 'GMenuBar'
add(obj, child, expand = FALSE, fill = NULL, anchor = NULL, ...)

## S3 method for class 'GMenuBar'
svalue(obj, index = NULL, drop = NULL, ...)

## S3 replacement method for class 'GMenuBar'
svalue(obj, index=NULL, ...) <- value

```

**Arguments**

menu.list	A list defining the menu structure. Named sub lists determine the submenu titles and structure. The list may have components of class: GAction, mapped to a button; GSeparator, mapped to a horizontal separator; GRadio, mapped to linked buttons; or GCheckbox, mapped to a checkbox button.
popup	logical. If true, make a popup window to be added through a handler call
container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <b>gWidgets2tcltk</b> or <b>gWidgets2WWW2</b> .)
...	These values are passed to the add method of the parent container. Examples of values are expand, fill, and anchor, although they're not always supported by a given widget. For more details see <a href="#">add</a> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with gaction and the dialogs.
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will look up the toolkit through <a href="#">guiToolkit</a> .
obj	parent object
child	list. a menubar list or gmenu instance.
expand	NULL or logical. For box containers controls whether a child will expand to fill the allocated space.
fill	NULL or character. For box containers. The value of fill (not always respected) is used to control if expansion happens vertically (y), horizontally (x) or both (both or TRUE). For vertically filled box containers, children always fill horizontally (atleast) and for horizontally filled box containers, children always fill vertically (atleast). This is important to realize when trying to size buttons, say.
anchor	NULL or integer. For box containers. The anchor argument is used to position the child within the parent when there is more space allocated than the child requests. This is specified with a Cartesian pair in $\{-1,0,1\} \times \{-1, 0, 1\}$ .

index	ignored
drop	ignored
value	a list or menu bar specifying the new menubar

---

gmessage	<i>Constructor for modal message dialog</i>
----------	---

---

### Description

Constructor for modal message dialog  
 generic for toolkit dispatch

### Usage

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

.gmessage(toolkit, msg, title = "message", icon = "", parent = NULL, ...)
```

### Arguments

msg	Character. message to display.
title	Character. Title
icon	What icon to show
parent	Hint as to where to display
...	ignored
toolkit	toolkit

### See Also

[gmessage](#), [gconfirm](#), [gbasicdialog](#), [galert](#)

gnotebook

*Constructor for a tabbed notebook container***Description**

The tabbed notebook container allows one to hold many different pages with a mechanism – tabs – to switch between them. In gWidgets2 new pages are added through the add method. This is usually called implicitly in the child object's constructor. One passes in the tab label through the extra label argument. Labels may be subsequently changed through names<-.

Children added to notebooks need a label, a position and optionally a close button (if supported). The arguments expand, fill, anchor are not specified – children expand and fill the allocated space.

Dispose deletes the current page, not the entire notebook object. To delete a specific page, a combination of svalue<- and dispose may be used.

The names of a notebook are the page tab labels. These may be retrieved and set through the names method.

The notebook object contains pages referenced by index. This allows access to underlying page.

The change handler for the notebook is called when the page changes. The new page number is passed back in the page.no component of 'h', which in some cases may differ from the value given by svalue within the handler call.

Dispose deletes the current page, not the entire notebook object. To delete a specific page, a combination of svalue<- and dispose may be used.

**Usage**

```
gnotebook(tab.pos = 3, container = NULL, ..., toolkit = guiToolkit())
```

```
.gnotebook(toolkit, tab.pos = 3, container = NULL, ...)
```

```
## S3 method for class 'GNotebook'
add(obj, child, expand, fill, anchor, ...)
```

```
## S3 method for class 'GNotebook'
dispose(obj, ...)
```

```
## S3 method for class 'GNotebook'
names(x)
```

```
## S3 replacement method for class 'GNotebook'
svalue(obj, index=TRUE, ...) <- value
```

```
## S3 method for class 'GNotebook'
x[i, j, ..., drop = TRUE]
```

```
## S3 method for class 'GNotebook'
addHandlerChanged(obj, handler, action = NULL, ...)
```

```
## S3 method for class 'GStackWidget'
dispose(obj, ...)
```

### Arguments

tab.pos	integer. Position of tabs, 1 on bottom, 2 left, 3 top, 4 right. (If supported)
container	parent container
...	passed to add method for container
toolkit	underlying toolkit
obj	gnotebook object
child	some child component to add
expand	NULL or logical. For box containers controls whether a child will expand to fill the allocated space.
fill	NULL or character. For box containers. The value of fill (not always respected) is used to control if expansion happens vertically (y), horizontally (x) or both (both or TRUE). For vertically filled box containers, children always fill horizontally (atleast) and for horizontally filled box containers, children always fill vertically (atleast). This is important to realize when trying to size buttons, say.
anchor	NULL or integer. For box containers. The anchor argument is used to position the child within the parent when there is more space allocated than the child requests. This is specified with a Cartesian pair in $\{-1,0,1\} \times \{-1, 0, 1\}$ .
x	notebook object svalue method Set the currently raised tab by index (the default) or name
index	TRUE refer to tab by 1-based index; FALSE allows reference by tab label.
value	assignment value
i	row index. Either integer or character
j	ignored
drop	ignored
handler	handler
action	passed along to handler via <code>h[["action"]]</code> .

### Value

none. called for its side effect.

### Note

In **gWidgets2** the button arguments of the `gWidgets` constructor are removed. One passes the close button request to the `add` method.

To keep the signature the same as the generic, several arguments are passed in via `...`:

**label** A character. Label text for tab

**i** An integer in 0 to length(obj) indicating the position to insert child. The new page is inserted to the right of page number i. When i=0, the page appears at the front, when i is not specified it appears at the end.

**close.button** A logical. If TRUE – and the toolkit supports it – the page tab will include a close button.

### See Also

[gstackwidget](#) for a similar widget without tabs.

### Examples

```
## Not run:

w <- gwindow("notebook example", visible=FALSE)
nb <- gnotebook(container=w)
gbutton("Page one", label="tab 1", container=nb) ## note label argument
gbutton("Page two", label="tab 2", container=nb)
svalue(nb) <- 1
addHandlerChanged(nb, handler=function(h,...) {
  message(sprintf("On page %s", h$page.no)) ## svalue(h$obj) not always right
})
svalue(nb) <- 2 ## or use "Page two"
dispose(nb)
length(nb)

## End(Not run)
```

---

gpanedgroup

*constructor for a two-paned container*

---

### Description

A container for holding two child widgets where the space allocated to each can be manipulated by the user with a pane between the widgets, or programmatically via `svalue<-`. The value specified to `svalue<-` can be a number in  $[0,1]$ , in which case it is a proportion or an integer, in which case it is a pixel size (from the left or the top). The ambiguous case 1 or 1L is determined by class. The value of `svalue` is in proportion units.

### Usage

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

.gpanedgroup(toolkit, horizontal = TRUE, container = NULL, ...)
```

**Arguments**

horizontal	direction of layout
container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <b>gWidgets2tcltk</b> or <b>gWidgets2WWW2</b> .)
...	These values are passed to the add method of the parent container. Examples of values are expand, fill, and anchor, although they're not always supported by a given widget. For more details see <a href="#">add</a> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with gaction and the dialogs.
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will look up the toolkit through <a href="#">guiToolkit</a> .

**Details**

Child widgets are added in the usual way, typically through the container argument of a constructor. Only two children may be added. Children expand and fill the allocated space.

**Note**

Setting the size is often only possible after the container has been realized on the screen. In the example, this call of `svalue<-` is done after the parent window is made visible for this reason. There were arguments to specify the children at construction, but these have been removed.

**Examples**

```
## Not run:
w <- gwindow("gpanedgroup", visible=FALSE)
pg <- gpanedgroup(cont=w)
gbutton("left", cont=pg)
gbutton("right", cont=pg)

visible(w) <- TRUE
svalue(pg) <- 0.33

## End(Not run)
```

---

gprogressbar

*Basic progress bar widget*


---

**Description**

Basic progress bar widget

S3 generic whose methods are implemented in the toolkit packages

**Usage**

```
progressbar(value = 10, container = NULL, ..., toolkit = guiToolkit())
  .progressbar(toolkit, value, container, ...)
```

**Arguments**

value	Initial value, between 0 and 100. A value of NULL will make pulsing bar with indeterminate state. For some toolkits, this must be called periodically to pulse the bar.
container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <b>gWidgets2tcltk</b> or <b>gWidgets2WWW2</b> .)
...	These values are passed to the add method of the parent container. Examples of values are expand, fill, and anchor, although they're not always supported by a given widget. For more details see <a href="#">add</a> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with gaction and the dialogs.
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <a href="#">guiToolkit</a> .

**Value**

a GButton instance. While this object has its own (reference) methods, one primarily interacts with it through S3 methods defined within the package.

**Examples**

```
## Not run:
w <- gwindow("progress bar example")
pb <- gprogressbar(cont=w)
for(i in 10:100) {Sys.sleep(.1); svalue(pb) <- i}

## End(Not run)
```

---

gradio

---

*Constructor for radio button widget*


---

**Description**

A radio button group allows a user to select one from many items. In **gWidgets2** the radio button widget shows 2 or more items. The items are coerced to characters, usually by the underlying toolkit. Use the `coerce_with` property to set a function, such as `as.numeric`, to coerce the return value during the `svalue` code. The items are referred to with the `[` method, the selected one with `svalue`.

The `svalue` method returns the radio button label or its index if `index=TRUE`. Labels are coerced to character by many of the toolkits. To be sure to return a numeric value, one can assign to the `coerce_with` property, e.g., `obj$coerce_with <- as.numeric`. For all widgets, if a function is specified to `coerce_with` it will be called on the value returned by `svalue`.

For a radio button group, for `svalue` the value can be referred to by index or label.

Check for repeated items before passing on to `set_items`

### Usage

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

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

## S3 method for class 'GRadio'
svalue(obj, index = NULL, drop = TRUE, ...)

## S3 replacement method for class 'GRadio'
svalue(obj, index=NULL, drop=TRUE, ...) <- value

## S3 replacement method for class 'GRadio'
x[i, j, ...] <- value
```

### Arguments

<code>items</code>	items to select from
<code>selected</code>	index of initially selected item
<code>horizontal</code>	layout direction
<code>handler</code>	A handler assigned to the default change signal. Handlers are called when some event triggers a widget to emit a signal. For each widget some default signal

is assumed, and handlers may be assigned to that through `addHandlerChanged` or at construction time. Handlers are functions whose first argument, `h` in the documentation, is a list with atleast two components `obj`, referring to the object emitting the signal and `action`, which passes in user-specified data to parameterize the function call.

Handlers may also be added via `addHandlerXXX` methods for the widgets, where `XXX` indicates the signal, with a default signal mapped to `addHandlerChanged` (cf. [addHandler](#) for a listing). These methods pass back a handler ID that can be used with `blockHandler` and `unblockHandler` to suppress temporarily the calling of the handler.

<code>action</code>	User supplied data passed to the handler when it is called
<code>container</code>	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <code>gWidgets2tcltk</code> or <code>gWidgets2WWW2</code> .)
<code>...</code>	These values are passed to the <code>add</code> method of the parent container. Examples of values are <code>expand</code> , <code>fill</code> , and <code>anchor</code> , although they're not always supported by a given widget. For more details see <a href="#">add</a> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with <code>gaction</code> and the dialogs.
<code>toolkit</code>	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <code>guiToolkit</code> .
<code>obj</code>	object of method call
<code>index</code>	NULL or logical. If TRUE and widget supports it an index, instead of a value will be returned.
<code>drop</code>	NULL or logical. If widget supports it, <code>drop</code> will work as it does in a data frame or perhaps someother means.
<code>value</code>	items to assigns a choices for the buttons
<code>x</code>	GRadio object
<code>i</code>	button index. Leave as missing to replace items to select from.
<code>j</code>	ignored

### Examples

```
if(interactive()) {
  w <- gwindow("Selection widgets")
  g <- gvbox(cont=w)

  f1 <- gformlayout(cont=g)
  gcheckbox("checkbox", checked=TRUE, cont=f1, label="checkbox")
  gradio(state.name[1:4], selected=2, horizontal=TRUE, cont=f1, label="gradio")
  gcheckboxgroup(state.name[1:4], horizontal=FALSE, cont=f1, label="checkbox group")

  bg <- gggroup(cont=g)
  gbutton("ok", cont=bg, handler=function(h,...) print(sapply(f1$children, svalue)))
}
```

```
}

```

---

gseparator	<i>constructor providing a widget for displaying a line in a GUI</i>
------------	--

---

### Description

The `gseparator` widget provides a horizontal or vertical line to visually divide child components of its parent container. In addition to box containers this can be used within toolbars (where one uses `parent` and not `container`).

### Usage

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

### Arguments

<code>horizontal</code>	Logical. Is separator drawn horizontally?
<code>container</code>	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <code>gWidgets2tcltk</code> or <code>gWidgets2WWW2</code> .)
<code>...</code>	These values are passed to the <code>add</code> method of the parent container, and occasionally have been used to sneak in hidden arguments to toolkit implementations.
<code>toolkit</code>	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <code>guiToolkit</code> .

### Examples

```
## Not run:

w <- gwindow("Within page", visible=FALSE)
g <- gvbox(container=w)
glabel("Lorem ipsum ...", cont=g)
gseparator(cont=g)
bg <- ggrouop(cont=g); addSpring(bg)
gbutton("dismiss", container=bg, handler=function(h,...) dispose(w))
visible(w) <- TRUE

w1 <- gwindow("within layout", visible=FALSE)
lyt <- glayout(container=w1)
lyt[1,1] <- "label"
lyt[2,1:2] <- gseparator(container=lyt)
lyt[3,2] <- "asdf"
```

```
visible(w1) <- TRUE

w2 <- gwindow("Within toolbar", visible=FALSE)
l <- list(file=gaction("File", parent=w2),
          sep=gseparator(parent=w2),
          quit=gaction("quit", parent=w2))
gtoolbar(l, cont=w2)
glabel("Lorem ipsum ...", container=w2)
visible(w2) <- TRUE

## End(Not run)
```

---

gslider

*slider widget constructor*

---

## Description

A slider widgets allows a selection from a range of numeric values. The widget presents the user with a quick to adjust, but relatively difficult to adjust precisely way to to pick a number.

## Usage

```
gslider(
  from = 0,
  to = 100,
  by = 1,
  length.out = NULL,
  along.with = NULL,
  value = from[1],
  horizontal = TRUE,
  handler = NULL,
  action = NULL,
  container = NULL,
  ...,
  toolkit = guiToolkit()
)

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

**Arguments**

from	If a number of length one then a starting point, in which case to, by are passed to seq. Otherwise a sequence of values for which <code>sort(unique(from))</code> will order
to	ending point when from is starting point
by	step size if not specified by from
length.out	in place of by
along.with	in place of length.out
value	initial value
horizontal	Logical. Is separator drawn horizontally?
handler	A handler assigned to the default change signal. Handlers are called when some event triggers a widget to emit a signal. For each widget some default signal is assumed, and handlers may be assigned to that through <code>addHandlerChanged</code> or at construction time. Handlers are functions whose first argument, <code>h</code> in the documentation, is a list with atleast two components <code>obj</code> , referring to the object emitting the signal and <code>action</code> , which passes in user-specified data to parameterize the function call.  Handlers may also be added via <code>addHandlerXXX</code> methods for the widgets, where <code>XXX</code> indicates the signal, with a default signal mapped to <code>addHandlerChanged</code> (cf. <a href="#">addHandler</a> for a listing). These methods pass back a handler ID that can be used with <code>blockHandler</code> and <code>unblockHandler</code> to suppress temporarily the calling of the handler.
action	User supplied data passed to the handler when it is called
container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <code>gWidgets2tcltk</code> or <code>gWidgets2WWW2</code> .)
...	These values are passed to the <code>add</code> method of the parent container. Examples of values are <code>expand</code> , <code>fill</code> , and <code>anchor</code> , although they're not always supported by a given widget. For more details see <a href="#">add</a> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with <code>gaction</code> and the dialogs.
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will look up the toolkit through <code>guiToolkit</code> .

**See Also**

[gspinbutton](#)

**Examples**

```
if(interactive()) {
  ## a range widget uses either a slider or a linked spinbutton to select a value
  w <- gwindow("Range widget", visible=FALSE)
  g <- ggroup(cont=w, horizontal=TRUE)
```

```

sl <- gslider(from=0, to=100, by=1, value=0, cont=g, expand=TRUE, fill="both")
sp <- gspinbutton(from=0, to=100, by=1, value=0, cont=g)

## Two ways to do this:
## addHandlerChanged(sl, function(...) svalue(sp) <- svalue(sl))
## addHandlerChanged(sp, function(...) svalue(sl) <- svalue(sp))

f <- function(h, ...) svalue(h$action) <- svalue(h$obj)
addHandlerChanged(sl, f, action=sp)
addHandlerChanged(sp, f, action=sl)

visible(w) <- TRUE
}

```

---

gspinbutton

*Spinbutton constructor*


---

### Description

A spinbutton allows the user to select from a pre-selected range of numbers. Similar to a slider, but with more precision, but slower to adjust. The basic arguments mirror that of `seq.int`.

### Usage

```

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

```

```

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

```

```

        container = NULL,
        ...
    )

```

### Arguments

from	from value
to	to value
by	step length
length.out	number of steps. Only one of by or length.out is used.
along.with	Take from
value	initial value
digits	number of digits to display, should the toolkit support it
handler	<p>A handler assigned to the default change signal. Handlers are called when some event triggers a widget to emit a signal. For each widget some default signal is assumed, and handlers may be assigned to that through <code>addHandlerChanged</code> or at construction time. Handlers are functions whose first argument, <code>h</code> in the documentation, is a list with atleast two components <code>obj</code>, referring to the object emitting the signal and <code>action</code>, which passes in user-specified data to parameterize the function call.</p> <p>Handlers may also be added via <code>addHandlerXXX</code> methods for the widgets, where <code>XXX</code> indicates the signal, with a default signal mapped to <code>addHandlerChanged</code> (cf. <a href="#">addHandler</a> for a listing). These methods pass back a handler ID that can be used with <code>blockHandler</code> and <code>unblockHandler</code> to suppress temporarily the calling of the handler.</p>
action	User supplied data passed to the handler when it is called
container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <code>gWidgets2tcltk</code> or <code>gWidgets2WWW2</code> .)
...	<p>These values are passed to the <code>add</code> method of the parent container. Examples of values are <code>expand</code>, <code>fill</code>, and <code>anchor</code>, although they're not always supported by a given widget. For more details see <a href="#">add</a>. Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with <code>gaction</code> and the dialogs.</p>
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <a href="#">guiToolkit</a> .

### See Also

[gslider](#)

**Examples**

```

if(interactive()) {
  ## a range widget uses either a slider or a linked spinbutton to select a value
  w <- gwindow("Range widget", visible=FALSE)
  g <- ggroup(cont=w, horizontal=TRUE)
  sl <- gslider(from=0, to=100, by=1, value=0, cont=g, expand=TRUE, fill="both")
  sp <- gspinbutton(from=0, to=100, by=1, value=0, cont=g)

  ## Two ways to do this:
  ## addHandlerChanged(sl, function(...) svalue(sp) <- svalue(sl))
  ## addHandlerChanged(sp, function(...) svalue(sl) <- svalue(sp))

  f <- function(h, ...) svalue(h$action) <- svalue(h$obj)
  addHandlerChanged(sl, f, action=sp)
  addHandlerChanged(sp, f, action=sl)

  visible(w) <- TRUE
}

```

---

gstackwidget

*Constructor for a stack of widgets*


---

**Description**

This widget is like a notebook – it holds a stack of pages, but does not provide the tabs to work with. Most methods are inherited from `gnotebook`'s.

**Usage**

```
gstackwidget(container = NULL, ..., toolkit = guiToolkit())
```

```
.gstackwidget(toolkit, container = NULL, ...)
```

**Arguments**

container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <code>gWidgets2tcltk</code> or <code>gWidgets2WWW2</code> .)
...	These values are passed to the <code>add</code> method of the parent container, and occasionally have been used to sneak in hidden arguments to toolkit implementations.
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will look up the toolkit through <code>guiToolkit</code> .

**Examples**

```
## Not run:
w <- gwindow("stack widget", visible=FALSE)
add_page <- function(cont, i) {
  g <- gvbox(container=cont)
  glabel(sprintf("page %s",i), container=g)
  bg <- ggroup(container=g); addSpring(bg)
  lb <- gbutton("Previous", container=bg, handler=function(h,...) {
    svalue(cont) <- max(1, i - 1)
  })
  rb <- gbutton("Next", container=bg, handler=function(h,...) {
    svalue(cont) <- min(i + 1, length(cont))
  })
}
sw <- gstackwidget(cont=w)
sapply(1:5, add_page, cont=sw)
visible(w) <- TRUE

## End(Not run)
```

---

gstatusbar

*constructor to add a status bar to main window*


---

**Description**

constructor to add a status bar to main window

generic for toolkit dispatch

**Usage**

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

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

**Arguments**

text	inital status text
container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <b>gWidgets2tcltk</b> or <b>gWidgets2WWW2</b> .)
...	These values are passed to the add method of the parent container, and occasionally have been used to sneak in hidden arguments to toolkit implementations.
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <a href="#">guiToolkit</a> .

**Examples**

```
## Not run:
w <- gwindow("Statusbar", visible=FALSE)
sb <- gstatusbar("status text", container=w)
g <- gybox(container=w)
gbutton("update", container=g, handler=function(...) svalue(sb) <- "new value")
visible(w) <- TRUE

## End(Not run)
```

gtable

*A constructor for displaying tabular data for selection***Description**

The tabular widget allows a user to select one (or more) row(s) using the mouse or keyboard selection. The selected rows are the main property and are returned by `svalue` through their key (from the column specified by `chosen.col`), or by index. The change handler changes on double-click event. Use `add.handler.click` to respond to a change in selection.

For `gtable` one can pass in row(s) to select by index (when `index=TRUE`) or by match among the values in the chosen column. For setting by index, a value of `0L` or `integer(0)` will clear the current selection

For `GTable` objects the `[]` and `[<-` methods are (mostly) implemented. The `[]` method allows one to access the object using the regular matrix notation (but there is no list notation, e.g. `$` or `[[`, defined). The `[<-` method is available, but for most toolkits is restricted: one can not add columns, add rows, remove columns, remove rows, or change the type of the column. For all of these actions, one can reassign the items being displayed through the idiom `obj[] <- new_items`. This allows the widget to resize or redo the column renderers.

The change handler for `GTable` is called when the selection changes. This is often the result of a click event (but need not be), although we alias to `addHandlerClicked`. For double click events, see `addHandlerDoubleClick`.

Double clicking is used to activate an item (single click is selection). We also bind pressing the Return key on an item to initiate this signal

For `GTable`, `visibility` refers to which rows are currently shown, not whether the widget itself is shown or hidden. (For the latter, place the widget into a container and adjust that). One can use this method to perform filtering by adjusting which rows are displayed according to some criteria that returns a logical.

For `GTable` the `size<-` method is overridden to allow one to specify the column widths. To do so, pass in the values for `width`, `height` or `column.widths` as named components of a list. The value of `column.widths` should be a numeric vector of pixel widths of length matching the number of columns.

**Usage**

```

gtable(
  items,
  multiple = FALSE,
  chosen.col = 1,
  icon.col = NULL,
  tooltip.col = NULL,
  handler = NULL,
  action = NULL,
  container = NULL,
  ...,
  toolkit = guiToolkit()
)

.gtable(
  toolkit,
  items,
  multiple = FALSE,
  chosen.col = 1,
  icon.col = NULL,
  tooltip.col = NULL,
  handler = NULL,
  action = NULL,
  container = NULL,
  ...
)

## S3 method for class 'GTable'
svalue(obj, index = NULL, ..., value)

## S3 method for class 'GTable'
x[i, j, ..., drop = TRUE]

## S3 method for class 'GTable'
addHandlerChanged(obj, handler, action = NULL, ...)

## S3 method for class 'GTable'
addHandlerDoubleClick(obj, handler, action = NULL, ...)

## S3 method for class 'GTable'
visible(obj, ...)

## S3 replacement method for class 'GTable'
size(obj) <- value

```

**Arguments**

`items`                    `data.frame` specifies items for selection. May be a vector, matrix or data frame

multiple	logical allow multiple selection
chosen.col	which value from the row is returned by selection
icon.col	NULL or integer. If latter, specifies column containing stock icon
tooltip.col	NULL or integer. If latter, specifies column containing tooltip
handler	<p>A handler assigned to the default change signal. Handlers are called when some event triggers a widget to emit a signal. For each widget some default signal is assumed, and handlers may be assigned to that through <code>addHandlerChanged</code> or at construction time. Handlers are functions whose first argument, <code>h</code> in the documentation, is a list with atleast two components <code>obj</code>, referring to the object emitting the signal and <code>action</code>, which passes in user-specified data to parameterize the function call.</p> <p>Handlers may also be added via <code>addHandlerXXX</code> methods for the widgets, where <code>XXX</code> indicates the signal, with a default signal mapped to <code>addHandlerChanged</code> (cf. <code>addHandler</code> for a listing). These methods pass back a handler ID that can be used with <code>blockHandler</code> and <code>unblockHandler</code> to suppress temporarily the calling of the handler.</p>
action	User supplied data passed to the handler when it is called
container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <code>gWidgets2tcltk</code> or <code>gWidgets2WWW2</code> .)
...	These values are passed to the <code>add</code> method of the parent container. Examples of values are <code>expand</code> , <code>fill</code> , and <code>anchor</code> , although they're not always supported by a given widget. For more details see <code>add</code> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with <code>gaction</code> and the dialogs.
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <code>guiToolkit</code> .
obj	object of method call
index	NULL or logical. If TRUE and widget supports it an index, instead of a value will be returned.
value	value to assign for selection or property
x	GTable object
i	row index
j	column index
drop	do we drop when subsetting

### Details

Many generic methods for data frames are implemented for `gtable`. These include `[`, `[<-`, `length`, `names`, and `names<-`

**Value**

Returns an object of class GTable

**Examples**

```
## Not run:
w <- gwindow("gtable example", visible=FALSE)
g <- gvbox(cont=w)
tbl <- gtable(mtcars, cont=g, expand=TRUE, fill=TRUE)
addHandlerClicked(tbl, handler=function(h,...) sprintf("You selected %s", svalue(h$obj)))
visible(w) <- TRUE

## End(Not run)
```

---

gtext

*Multiline text edit constructor*


---

**Description**

The multiline text widget has its main property the text contained within.

- The `svalue` will return a string (length-1 character vector) with embedded newlines
- The "change" handler is `addHandlerKeystroke`
- Use `addHandlerSelectionChanged` to monitor the selection

The `svalue` method for a `gtext` object returns a) the buffers content; b) the selected text (if `drop=TRUE`, but not `NULL`), this can be used to set the selected value, as well; c) the index of the selection if `index=TRUE`.

**Usage**

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

.gtext(
  toolkit,
```

```

    text = NULL,
    width = NULL,
    height = 300,
    font.attr = NULL,
    wrap = TRUE,
    handler = NULL,
    action = NULL,
    container = NULL,
    ...
)

insert(
  obj,
  value,
  where = c("end", "beginning", "at.cursor"),
  font.attr = NULL,
  do.newline = TRUE,
  ...
)

## S3 method for class 'GText'
insert(
  obj,
  value,
  where = c("end", "beginning", "at.cursor"),
  font.attr = NULL,
  do.newline = TRUE,
  ...
)

## S3 method for class 'GText'
dispose(obj, ...)

## S3 method for class 'GText'
svalue(obj, index = NULL, drop = NULL, ...)

```

### Arguments

text	initial text
width	width of widget
height	height of widget (when width is specified)
font.attr	font attributes for text buffer. One can also specify font attributes for insertion. The font attributes are specified with a list with named components, with names and values coming from: <b>weight</b> in c("light", "normal", "bold", "heavy") <b>style</b> in c("normal", "oblique", "italic") <b>family</b> in c("sans", "helvetica", "times", "monospace")

	<b>size</b> in c("xx-small", "x-small", "small", "medium", "large", "x-large", "xx-large")
	<b>foreground</b> a value in colors()
	<b>background</b> a value in colors()
wrap	logical do lines wrap
handler	<p>A handler assigned to the default change signal. Handlers are called when some event triggers a widget to emit a signal. For each widget some default signal is assumed, and handlers may be assigned to that through <code>addHandlerChanged</code> or at construction time. Handlers are functions whose first argument, <code>h</code> in the documentation, is a list with atleast two components <code>obj</code>, referring to the object emitting the signal and <code>action</code>, which passes in user-specified data to parameterize the function call.</p> <p>Handlers may also be added via <code>addHandlerXXX</code> methods for the widgets, where <code>XXX</code> indicates the signal, with a default signal mapped to <code>addHandlerChanged</code> (cf. <a href="#">addHandler</a> for a listing). These methods pass back a handler ID that can be used with <code>blockHandler</code> and <code>unblockHandler</code> to suppress temporarily the calling of the handler.</p>
action	User supplied data passed to the handler when it is called
container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <code>gWidgets2tcltk</code> or <code>gWidgets2WWW2</code> .)
...	These values are passed to the <code>add</code> method of the parent container. Examples of values are <code>expand</code> , <code>fill</code> , and <code>anchor</code> , although they're not always supported by a given widget. For more details see <a href="#">add</a> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with <code>gaction</code> and the dialogs.
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <code>guiToolkit</code> .
obj	object
value	text to insert
where	position of insertion
do.newline	logical add a newline at end
index	NULL or logical. If TRUE and widget supports it an index, instead of a value will be returned.
drop	NULL or logical. If widget supports it, drop will work as it does in a data frame or perhaps someother means.

## Value

called for side effect

**Note**

with **gWidgetstcltk** the allocation of size to the widget may be incorrect. It is best to wait until the widget is added before displaying its parent window. See the `visible` argument for `gwindow`.

**Examples**

```
## Not run:
w <- gwindow("gtext example", visible=FALSE)
g <- gvbox(cont=w)
t1 <- gtext("initial text", container=g)
t2 <- gtext("monospace", font.attr=list(family="monospace"), container=g)
insert(t2, "new text in bold", font.attr=list(weight="bold"))
visible(w) <- TRUE

## End(Not run)
```

---

gtimer

*Basic timer widget*


---

**Description**

Calls FUN every ms/1000 seconds. A timer is stopped through its `stop_timer` method which is called using OO style: `obj$stop_timer()`.

**Usage**

```
gtimer(
  ms,
  FUN,
  data = NULL,
  one.shot = FALSE,
  start = TRUE,
  toolkit = guiToolkit()
)

.gtimer(toolkit, ms, FUN, data = NULL, one.shot = FALSE, start = TRUE)
```

**Arguments**

<code>ms</code>	interval in milliseconds
<code>FUN</code>	FUnction to call. Has one argument, data passed in
<code>data</code>	passed to function
<code>one.shot</code>	logical. If TRUE, called just once, else repeats
<code>start</code>	logical. If FALSE, started by <code>start_timer</code> OO method. (Call <code>obj\$start_time()</code> ).
<code>toolkit</code>	gui toolkit to dispatch into

**Examples**

```
## Not run:
i <- 0
FUN <- function(data) {i <- i + 1; if(i > 10) a$stop_timer(); print(i)}
a <- gtimer(1000, FUN)
##
## a one shot timer is run only once
FUN <- function(data) message("Okay, I can breathe now")
hold_breath <- gtimer(1000*60, FUN, one.shot=TRUE)

## End(Not run)
```

---

gtoolbar

*A toolbar constructor*


---

**Description**

A toolbar can be added to a main window to proxy various actions. Toolbars can also contain various widgets, such as buttons, checkboxes, radio buttons, etc. These should be constructed using a parent argument – not a container argument. In **gWidgets2** a toolbar is specified by a list of toolbar items. The `svalue` and `svalue<-` methods may be used to get or set the items.

A toolbar item is a list of action items or a toolbar instance

for a toolbar, `svalue<-` replaces the toolbar items with new ones specified by value.

**Usage**

```
gtoolbar(
  toolbar.list = list(),
  style = c("both", "icons", "text", "both-horiz"),
  container = NULL,
  ...,
  toolkit = guiToolkit()
)

.gtoolbar(
  toolkit,
  toolbar.list = list(),
  style = c("both", "icons", "text", "both-horiz"),
  container = NULL,
  ...
)

## S3 method for class 'GToolBar'
add(obj, child, expand = FALSE, fill = NULL, anchor = NULL, ...)

## S3 replacement method for class 'GToolBar'
svalue(obj, index=NULL, ...) <- value
```

**Arguments**

toolbar.list	list. A one-level list of gaction items, gseparator items or possibly other widgets. In the latter cases the container argument is not specified prior. (XXX Need to work this out with gWidgetstcltk)
style	style for icon or text.
container	a GWindow instance
...	ignored
toolkit	toolkit
obj	parent object
child	child widget
expand	NULL or logical. For box containers controls whether a child will expand to fill the allocated space.
fill	NULL or character. For box containers. The value of fill (not always respected) is used to control if expansion happens vertically (y), horizontally (x) or both (both or TRUE). For vertically filled box containers, children always fill horizontally (atleast) and for horizontally filled box containers, children always fill vertically (atleast). This is important to realize when trying to size buttons, say.
anchor	NULL or integer. For box containers. The anchor argument is used to position the child within the parent when there is more space allocated than the child requests. This is specified with a Cartesian pair in $\{-1,0,1\} \times \{-1, 0, 1\}$ .
index	NULL or logical. If TRUE and widget supports it an index, instead of a value will be returned.
value	value to assign for selection or property

---

gtoolkit

*Which toolkit are we using?*


---

**Description**

Which toolkit are we using?

**Usage**

```
gtoolkit()
```

**Value**

string of toolkit (RGtk2, tcltk, Qt, ???)

gtree

*constructor for widget to display heirarchical data***Description**

The gtree widget is used to present structured heirarchical data. This data may be specified through a data frame with some accompanying columns by which to split the data, or dynamically through a function (offspring).

For a GTree object, svalue refers to the path specified as a vector of keys or (if INDEX=TRUE) by an integer vector of offspring positions. The drop argument is used to indicate if the terminus of the path is returned or the entire path, defaults=TRUE. To get the data associated with a row, use the [ method.

For a GTree object, svalue refers to the path specified as a vector of keys . For the assignment method, one assigns by index. That is svalue(tr, index=TRUE) <- svalue(tr, index=TRUE) should not change the state of the widget. (The index=TRUE argument is the case for setting, but not getting.)

The [ method is used to return the data associated with a selected row. The svalue method returns the path or its endpoint, the [ method returns the row data associated with the path.

The update method for GTree recomputes the base nodes, then reopens the given node if still available

**Usage**

```
gtree(
  x = NULL,
  INDICES = NULL,
  offspring = x,
  offspring.data = NULL,
  chosen.col = 1,
  offspring.col = 2,
  icon.col = NULL,
  tooltip.col = NULL,
  multiple = FALSE,
  handler = NULL,
  action = NULL,
  container = NULL,
  ...,
  toolkit = guiToolkit()
)
```

```
.gtree(
  toolkit,
  offspring = NULL,
  offspring.data = NULL,
  chosen.col = 1,
```

```

    offspring.col = 2,
    icon.col = NULL,
    tooltip.col = NULL,
    multiple = FALSE,
    handler = NULL,
    action = NULL,
    container = NULL,
    ...
)

## S3 method for class 'GTree'
svalue(obj, index = FALSE, drop = TRUE, ...)

## S3 replacement method for class 'GTree'
svalue(obj, index=TRUE, ...) <- value

## S3 method for class 'GTree'
x[i, j, ..., drop = FALSE]

## S3 method for class 'GTree'
update(object, ...)

```

### Arguments

x	Data frame. Optional, if given specify INDICES value to split data into heirarchical data structure
INDICES	Integers or column names, referring to columns of x. Used to form heirarchical structure. Order is important.
offspring	function. A function passed values path and data, the latter from offspring.data. The path is the current position of the parent item using the named keys from the chosen column.
offspring.data	Passed to second argument of offspring function. Used to parameterize a function call.
chosen.col	integer or one of column names of data frame returned by offspring. The chosen column gives the key and value of the path.
offspring.col	integer or column name. Points to column containing logical values indicating if a row has offspring.
icon.col	integer of one of the column names of the data frame. If provided (non-NULL), then this column should provide a stock icon name to be placed in the row for the given data.
tooltip.col	integer or one of the column names of the data frame. If provided (non-NULL), then the row for this item will have a tooltip given by the pointed to value.
multiple	logical. Is multiple selection allowed?
handler	A handler assigned to the default change signal. Handlers are called when some event triggers a widget to emit a signal. For each widget some default signal is assumed, and handlers may be assigned to that through addHandlerChanged

or at construction time. Handlers are functions whose first argument, `h` in the documentation, is a list with at least two components `obj`, referring to the object emitting the signal and `action`, which passes in user-specified data to parameterize the function call.

Handlers may also be added via `addHandlerXXX` methods for the widgets, where `XXX` indicates the signal, with a default signal mapped to `addHandlerChanged` (cf. [addHandler](#) for a listing). These methods pass back a handler ID that can be used with `blockHandler` and `unblockHandler` to suppress temporarily the calling of the handler.

<code>action</code>	User supplied data passed to the handler when it is called
<code>container</code>	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <code>gWidgets2tcltk</code> or <code>gWidgets2WWW2</code> .)
<code>...</code>	passed to update method
<code>toolkit</code>	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <code>guiToolkit</code> .
<code>obj</code>	object
<code>index</code>	index
<code>drop</code>	do we return tip or path
<code>value</code>	vector of indices
<code>i</code>	ignored
<code>j</code>	ignored
<code>object</code>	object to update

### Details

In the former case, the data frame is split up by the columns specified by `INDICES`. The first index is used to give the initial branches, the second index the second, etc. The end leaves are the data associated with a given path, with key given by that column specified by `chosen.col`

For the latter case, the "path" of the current node (the node and its ancestors) is passed to the `offspring` function which computes the next level in the heirarchy. This level is specified through a data frame. This data frame has special columns. The `chosen.col` specifies which column is used as the key in the path, the `icon.col` (when given) points to a stock icon name to decorate the column. Similarly, the `tooltip.columns`. The fact that a row in the data frame has offspring is specified through the `offspring.col` column, again specified by index or column name.

### Examples

```
#####
## This tree reads a list
offspring <- function(path=character(0), lst, ...) {
  if(length(path))
    obj <- lst[[path]]
  else
    obj <- lst
  nms <- names(obj)
```

```

hasOffspring <- sapply(nms, function(i) {
  newobj <- obj[[i]]
  is.recursive(newobj) && !is.null(names(newobj))
})
data.frame(comps=nms, hasOffspring=hasOffspring, ## fred=nms,
           stringsAsFactors=FALSE)
}
l <- list(a="1", b= list(a="21", b="22", c=list(a="231")))

## Not run:
w <- gwindow("Tree test")
t <- gtree(offspring=offspring, offspring.data=l, cont=w)

## End(Not run)

#####
## This tree looks at recursive objects
describe <- function(x) UseMethod("describe")
describe.default <- function(x) sprintf("An object with class %s", class(x)[1])
describe.integer <- function(x) sprintf("An integer with %s value%s", length(x),
  ifelse(length(x) > 1, "s", ""))
describe.numeric <- function(x) sprintf("A numeric with %s value%s", length(x),
  ifelse(length(x) > 1, "s", ""))
describe.factor <- function(x) sprintf("A factor with %s level%s", length(levels(x)),
  ifelse(length(levels(x)) > 1, "s", ""))

offspring <- function(path, obj) {
  if(length(path) > 0)
    x <- obj[[path]]
  else
    x <- obj

  nms <- names(x)
  recursive <- sapply(x, function(i) {
    is.recursive(i) &&
    !is.null(attr(i, "names")) &&
    length(i) > 0
  })
  descr <- sapply(x, describe)

  data.frame(Variable=nms, offspring=recursive, Description=descr, stringsAsFactors=FALSE)
}

l <- lm(mpg ~ wt, mtcars)
## Not run:
w <- gwindow("test")
gtree(offspring=offspring, offspring.data=l, cont=w)

## End(Not run)

```

---

guiToolkit	<i>set or get the current toolkit for gWidgets</i>
------------	--

---

**Description**

set or get the current toolkit for gWidgets

**Usage**

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

**Arguments**

name	name of toolkit (e.g. "tcltk", "RGtk2", "Qt" (not qtbase)). If NULL, then we search for it in a) an inherited toolkit object b) the "guiToolkit" option (which can be set via <code>options("guiToolkit"="RGtk2")</code> ), say. If that fails, we prompt for a selection for any installed toolkit. In the typical usage, this all happens in the background, except perhaps once.
------	---

In design this is to allow different toolkits to be used with different GUIs, but due to differences in event loops, this often leads to lockups, so is not recommended.

**Value**

an instance of guiWidgetsToolkit sub class.

---

guiWidgetsToolkit-class	<i>A class to record the toolkit a gui object uses</i>
-------------------------	--

---

**Description**

An observer can be observed

This interface is inherited by the base GComponent classes in the toolkit implementations. The methods defined here are referenced by the S3 methods. For example, `svalue` dispatches to `get_value` or `get_index`.

Class for commands. Has methods `do`, `redo`, `undo`

A list with ptr. delegates call of `do` or `undo` to appropriate command

A reference class to create a model that monitors the global workspace. The class has method `update_state` and the "getting" methods `get_by_class`, `get_by_function` (filter), `get_changes`. Use with a `gtimer` instance to keep up to date with changes to the workspace.

**Arguments**

...	passed to constructor
-----	-----------------------

**Details**

We combine both widget and container methods here. It isn't perfect, but they do share quite a bit. Perhaps, we could make the container class subclass the basic interface.

**Methods**

update(...) Call self.

update(...) Call self.

add\_observer(o, signal = "DEFAULT") Add an observer. Return id for block/remove/...

block\_observer(id) Block observers. If o missing, block all

block\_observers() Block all observers

notify\_observers(..., signal = "DEFAULT") Call each non-blocked observer

remove\_observer(id) Remove observer

unblock\_observer(id) Unblock observer. If id missing, unblock global block

unblock\_observers() Remove block of all observers. Keeps count, so may need to call again

add\_handler(signal, handler, action, ...) Add a handler to be called for the event indicated by signal

get\_enabled() is widget sensitive to user input

get\_index(drop = NULL, ...) svalue; index=TRUE

get\_value(drop = NULL, ...) Get main value of widget. From 'svalue' when index = FALSE or NULL

set\_enabled(value, ...) specify with logical if widget is sensitive to user input

set\_value(value, ..., drop = NULL) for 'svalue<-' when index = FALSE or NULL

any\_changes(...) Report if any changes

filter\_names(f) Filter the names by f

get\_by\_class(classes = character(0)) Return objects matching any of classes

get\_by\_function(f) Filter objects by function f

get\_changes() Return list of changes

initialize(...) Initialize state of cached objects

pop\_changes() pop changes, reset

update\_state(...) update cache of names/digests, then notify observers if there are changes

gvarbrowser

*Constructor for workspace variable browser***Description**

A workspace browser widget. The workspace browser displays values in the global environment. Displayed objects are shown in categories.

Return selected objects a string (when drop=TRUE) with recursive values separated by \$, or the objects themselves (when drop=FALSE).

**Usage**

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

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

## S3 method for class 'GVarBrowser'
svalue(obj, index = FALSE, drop = TRUE, ...)
```

**Arguments**

handler	A handler assigned to the default change signal. Handlers are called when some event triggers a widget to emit a signal. For each widget some default signal is assumed, and handlers may be assigned to that through <code>addHandlerChanged</code> or at construction time. Handlers are functions whose first argument, <code>h</code> in the documentation, is a list with at least two components <code>obj</code> , referring to the object emitting the signal and <code>action</code> , which passes in user-specified data to parameterize the function call.  Handlers may also be added via <code>addHandlerXXX</code> methods for the widgets, where <code>XXX</code> indicates the signal, with a default signal mapped to <code>addHandlerChanged</code> (cf. <a href="#">addHandler</a> for a listing). These methods pass back a handler ID that can be used with <code>blockHandler</code> and <code>unblockHandler</code> to suppress temporarily the calling of the handler.
action	User supplied data passed to the handler when it is called

container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <b>gWidgets2tcltk</b> or <b>gWidgets2WWW2</b> .)
...	These values are passed to the add method of the parent container. Examples of values are expand, fill, and anchor, although they're not always supported by a given widget. For more details see <a href="#">add</a> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with gaction and the dialogs.
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <a href="#">guiToolkit</a> .
obj	object of method call
index	NULL or logical. If TRUE and widget supports it an index, instead of a value will be returned.
drop	NULL or logical. If widget supports it, drop will work as it does in a data frame or perhaps someother means.

### Details

For defining the categories, the reference method `set_filter_classes` takes a named list, the names defining the categories, the values being the classes belonging to that category. Non categorized values appear separately. The default is defined in `gWidgets2::gvarbrowser_default_classes`.

The variable `browser` uses an instance of `WSWatcherModel` to monitor the global workspace. This instance may be useful for other purposes. (For example, one may add an observer that is called to listen for changes to the set of available data frames.). The instance is available through the `ws_model` property.

The `svalue` method returns the selected variable names. If `drop=FALSE` is given, the objects are returned.

The widget should support dragging from without needing to specify a `drag_source`, though this may be overridden.

Use `addHandlerChanged` to listen to activation of a variable (double clicking). Use `addHandlerSelectionChanged` to monitor change of selection.

### Description

Used as template for documentation

**Usage**

```

gwidget(
    handler = NULL,
    action = NULL,
    container = NULL,
    ...,
    toolkit = guiToolkit()
)

```

**Arguments**

handler	<p>A handler assigned to the default change signal. Handlers are called when some event triggers a widget to emit a signal. For each widget some default signal is assumed, and handlers may be assigned to that through <code>addHandlerChanged</code> or at construction time. Handlers are functions whose first argument, <code>h</code> in the documentation, is a list with at least two components <code>obj</code>, referring to the object emitting the signal and <code>action</code>, which passes in user-specified data to parameterize the function call.</p> <p>Handlers may also be added via <code>addHandlerXXX</code> methods for the widgets, where <code>XXX</code> indicates the signal, with a default signal mapped to <code>addHandlerChanged</code> (cf. <a href="#">addHandler</a> for a listing). These methods pass back a handler ID that can be used with <code>blockHandler</code> and <code>unblockHandler</code> to suppress temporarily the calling of the handler.</p>
action	User supplied data passed to the handler when it is called
container	A parent container. When a widget is created it can be incorporated into the widget heirarchy by passing in a parent container at construction time. (For some toolkits this is not optional, e.g. <code>gWidgets2tcltk</code> or <code>gWidgets2WWW2</code> .)
...	These values are passed to the <code>add</code> method of the parent container. Examples of values are <code>expand</code> , <code>fill</code> , and <code>anchor</code> , although they're not always supported by a given widget. For more details see <a href="#">add</a> . Occasionally the variable arguments feature has been used to sneak in hidden arguments to toolkit implementations. For example, when using a widget as a menubar object one can specify a parent argument to pass in parent information, similar to how the argument is used with <code>gaction</code> and the dialogs.
toolkit	Each widget constructor is passed in the toolkit it will use. This is typically done using the default, which will lookup the toolkit through <code>guiToolkit</code> .

---

gwindow

*gwindow*


---

**Description**

top-level window object

Dispatches on type of child (menubar, toolbar, statusbar, widget)

The `dispose` method destroys the top-level window and its children.

**Usage**

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

```
.gwindow(
  toolkit,
  title,
  visible,
  name,
  width,
  height,
  parent,
  handler,
  action,
  ...
)
```

```
## S3 method for class 'GWindow'
add(obj, child, expand = NULL, fill = NULL, anchor = NULL, ...)
```

```
## S3 method for class 'GWindow'
dispose(obj, ...)
```

**Arguments**

title	title for window's title bar. This is the main property and is accessed via <code>svalue</code> or <code>svalue&lt;-</code> .
visible	logical. If TRUE window is drawn when constructed. Otherwise, window can be drawn later using <code>visible&lt;-</code> . This value can default to FALSE by setting the option: <code>options("gWidgets:gwindow-default-visible-is-false"=TRUE)</code> . There are advantages: windows can draw slowly when adding many items. With <b>gWidgets2RGtk2</b> , the <code>ggraphics</code> widget can like to be added to an undrawn widget as this avoids sizing issue.
name	Name for registry of windows
width	initial width of window
height	initial height of window. This sets height before window manager manages the window

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	action passed t handler
...	ignored
toolkit	toolkit
obj	parent object
child	child widget
expand	NULL or logical. For box containers controls whether a child will expand to fill the allocated space.
fill	NULL or character. For box containers. The value of fill (not always respected) is used to control if expansion happens vertically (y), horizontally (x) or both (both or TRUE). For vertically filled box containers, children always fill horizontally (atleast) and for horizontally filled box containers, children always fill vertically (atleast). This is important to realize when trying to size buttons, say.
anchor	NULL or integer. For box containers. The anchor argument is used to position the child within the parent when there is more space allocated than the child requests. This is specified with a Cartesian pair in $\{-1,0,1\} \times \{-1, 0, 1\}$ .

**Value**

a GWindow instance

**Author(s)**

john verzani

---

installing\_gWidgets\_toolkits

*blurb about installation*

---

**Description**

put in so can be updated easily

**Usage**

installing\_gWidgets\_toolkits()

---

isExtant	<i>Check if widget is extant.</i>
----------	-----------------------------------

---

**Description**

Widgets can be destroyed, but their R object is still present. This is FALSE in that case.

**Usage**

```
isExtant(obj)

## Default S3 method:
isExtant(obj)
```

**Arguments**

obj	object
-----	--------

---

is_empty	<i>is value missing, null, 0-length or NA length 1</i>
----------	--

---

**Description**

is value missing, null, 0-length or NA length 1

**Usage**

```
is_empty(x)
```

**Arguments**

x	object to test
---	----------------

**Value**

logical

---

is_MacOSX	<i>Return logical indicating if we are on a macintosh machine</i>
-----------	---

---

**Description**

Return logical indicating if we are on a macintosh machine

**Usage**

```
is_MacOSX()
```

**Value**

logical

---

is_Windows	<i>Return logical indicating if we are on a Windows machine</i>
------------	---

---

**Description**

Return logical indicating if we are on a Windows machine

**Usage**

```
is_Windows()
```

**Value**

logical

---

observer	<i>constructor for handler object</i>
----------	---------------------------------------

---

**Description**

constructor for handler object

**Usage**

```
observer(receiver, handler, action = NULL)
```

**Arguments**

receiver	object receiving event
handler	function to call
action	used to parametrize handler call not exported, call using :::

---

redo	<i>Redo past action.</i>
------	--------------------------

---

**Description**

Some widgets support redo actions

**Usage**

```
redo(obj, ...)
```

```
## S3 method for class 'GComponent'  
redo(obj, ...)
```

**Arguments**

obj	object to redo
...	ignored

---

short_summary	<i>Provide a short summary for an object</i>
---------------	--

---

**Description**

Provide a short summary for an object

method for generic

method for generic

method for generic

method for generic

method for generic

method for generic

method for generic

method for generic

method for generic

**Usage**

```
short_summary(x)

## Default S3 method:
short_summary(x)

## S3 method for class 'numeric'
short_summary(x)

## S3 method for class 'character'
short_summary(x)

## S3 method for class 'logical'
short_summary(x)

## S3 method for class 'data.frame'
short_summary(x)

## S3 method for class 'matrix'
short_summary(x)

## S3 method for class 'list'
short_summary(x)

## S3 method for class 'lm'
short_summary(x)

## S3 method for class '`function`'
short_summary(x)
```

**Arguments**

x	object
---	--------

---

size	<i>Return size (width and height) of widget</i>
------	---

---

**Description**

The size is specified in pixels (integers). Some toolkits allow -1 as a default, but not all.

**Usage**

```
size(obj)

## Default S3 method:
size(obj)
```

```
size(obj) <- value
```

```
size(obj) <- value
```

### Arguments

obj	object
value	size in pixels

---

svalue	<i>svalue</i>
--------	---------------

---

### Description

This returns the "selected" value in a widget (if applicable), or its main property. Selection varies from widget to widget, but should generally be 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.

Calls `coerce_with` when available. This value is a function and may be set as any property if the constructor does not explicitly provide it.

This method sets the selected value of, or main property of the widget.

For `gformlayout` the `svalue` assignment method takes a named list and calls `svalue<-` on the children with matching names.

### Usage

```
svalue(obj, index = FALSE, drop = NULL, ...)
```

```
## Default S3 method:
```

```
svalue(obj, index = NULL, drop = NULL, ...)
```

```
svalue(obj, index=NULL, ...) <- value
```

```
svalue(obj, index=NULL, ...) <- value
```

```
## S3 replacement method for class 'GFormLayout'
```

```
svalue(obj, index=NULL, ...) <- value
```

### Arguments

obj	object of method call
index	NULL or logical. If TRUE and widget supports it an index, instead of a value will be returned.

drop	NULL or logical. If widget supports it, drop will work as it does in a data frame or perhaps someother means.
...	passed on to call
value	value to assign for selection or property

### Value

The return value varies, depending if the widget is a "selection" widget or not. For non-selection widgets, the main property is loosely defined (the title of a window, text of a label or button, spacing of box containers, ...). For selection widgets the return value is the currently selected value. If no selection is made, this will be a 0-length vector with the expected class, if possible. For selection widgets, when `index=TRUE`, the value is an integer, possible 0-length when non selection is made.

---

tag	<i>get a persistent attribute for an object</i>
-----	---

---

### Description

Unlike `attr<-`, this method (essentially) stores the attribute in a reference to the object, not a copy. As such it can be used within function call (handlers) to assign values outside the scope of the function call.

### Usage

```
tag(obj, key)

## Default S3 method:
tag(obj, key)

tag(obj, key) <- value

tag(obj, key) <- value
```

### Arguments

obj	object
key	character. Values are stored by key. If missing, all keys are returned.
value	to assign to key

---

tooltip	<i>Get a tooltip for the widget</i>
---------	-------------------------------------

---

**Description**

Get a tooltip for the widget  
Basic S3 method for tooltip<-  
Set a tooltip for the widget  
Basic S3 method for tooltip<-

**Usage**

```
tooltip(obj)

## Default S3 method:
tooltip(obj)

tooltip(obj) <- value

tooltip(obj) <- value
```

**Arguments**

obj	object
value	character tooltip value

---

undo	<i>Undo past action.</i>
------	--------------------------

---

**Description**

Some widgets support undo actions. See reference class method `can_undo` as well.

**Usage**

```
undo(obj, ...)
```

## S3 method for class 'GComponent'  
undo(obj, ...)

**Arguments**

obj	object to call undo on
...	ignored

---

visible	<i>Controls whether widget is visible or not</i>
---------	--

---

**Description**

For most – but not all – widgets, a widget is visible if it is shown. For others, parts of the widget may be controlled by visible. If the former state is desired, simply place widget into a box container.

**Usage**

```
visible(obj, ...)  
  
## Default S3 method:  
visible(obj, ...)  
  
visible(obj) <- value  
  
visible(obj) <- value
```

**Arguments**

obj	object
...	ignored
value	logical. Set visible state.

---

XXX	<i>Functions to message something needs doing. Easy to search for</i>
-----	---

---

**Description**

Functions to message something needs doing. Easy to search for

**Usage**

```
XXX(msg)
```

**Arguments**

msg	optional message to emit
-----	--------------------------

---

[.GDefaultWidget      *Return items*

---

## Description

Names are used in many different contexts.

We use the extraction operator, [, typically to refer to the underlying items from which a selection can be made. As well, we overload this to containers to refer to the child components.

The update method will cause a widget to recompute itself, if it is necessary.

The current items for a gdf object are both the visible and non-visible items. To retrieve just the currently visible items, use the idiom obj[visible(obj), ].

The underlying widget may allow autocompletion, if this is the case then this method is used to set the list of candidates.

## Usage

```
## S3 method for class 'GDefaultWidget'
x[i, j, ...]

## S3 method for class 'GComponent'
length(x)

## S3 replacement method for class 'GComponent'
length(x) <- value

## S3 method for class 'GComponent'
dim(x)

## S3 method for class 'GComponent'
names(x)

## S3 replacement method for class 'GComponent'
names(x) <- value

## S3 method for class 'GComponent'
dimnames(x)

## S3 replacement method for class 'GComponent'
dimnames(x) <- value

## S3 method for class 'GComponent'
x[i, j, ..., drop = TRUE]

## S3 method for class 'GContainer'
x[i, j, ..., drop = TRUE]
```

```
## S3 replacement method for class 'GComponent'  
x[i, j, ...] <- value  
  
## S3 method for class 'GComponent'  
update(object, ...)  
  
## S3 method for class 'GComponent'  
str(object, ...)  
  
## S3 method for class 'GDf'  
x[i, j, ..., drop = TRUE]  
  
## S3 method for class 'GEdit'  
x[i, j, ..., drop = TRUE]  
  
## S3 replacement method for class 'GFrame'  
names(x) <- value
```

### Arguments

x	component
i	index or row index if applicable
j	column index if applicable
...	dots argument
value	value to assign
drop	logical. Does return value get "dropped" down to something easier?
object	object to update

### Value

length of object

# Index

- \* **package**
  - gWidgets2-package, 3
- .addStockIcons (addStockIcons), 15
- .gaction (gaction), 22
- .galert (galert), 23
- .gbasicdialog (gbasicdialog), 24
- .gbutton (gbutton), 26
- .gcalendar (gcalendar), 28
- .gcheckbox (gcheckbox), 29
- .gcheckboxgroup (gcheckboxgroup), 31
- .gcombobox (gcombobox), 34
- .gconfirm (gconfirm), 36
- .gdf (gdf), 37
- .gdfnotebook (gdfnotebook), 39
- .gdfnotebook.default, 7
- .gedit (gedit), 39
- .getStockIconByName (addStockIcons), 15
- .getStockIcons (addStockIcons), 15
- .gexpandgroup (gexpandgroup), 43
- .gfile (gfile), 45
- .gfilebrowse (gfile), 45
- .gfilter (gfilter), 47
- .gformlayout (gformlayout), 50
- .gframe (gframe), 51
- .ggraphics (ggraphics), 53
- .ggraphicsnotebook (ggraphicsnotebook), 54
- .ggroup (ggroup), 56
- .ghtml (ghtml), 58
- .gimage (gimage), 59
- .ginput (ginput), 60
- .glabel (glabel), 61
- .glayout (glayout), 62
- .gmenu (gmenu), 64
- .gmessage (gmessage), 66
- .gnotebook (gnotebook), 67
- .gpanedgroup (gpanedgroup), 69
- .gprogressbar (gprogressbar), 70
- .gradio (gradio), 71
- .gseparator (gseparator), 74
- .gslider (gslider), 75
- .gspinbutton (gspinbutton), 77
- .gstackwidget (gstackwidget), 79
- .gstatusbar (gstatusbar), 80
- .gtable (gtable), 81
- .gtext (gtext), 84
- .gtimer (gtimer), 87
- .gtoolbar (gtoolbar), 88
- .gtree (gtree), 90
- .gvarbrowser (gvarbrowser), 96
- .gwindow (gwindow), 98
- .make\_gcombobox\_items (gcombobox), 34
- .stockIconFromClass (addStockIcons), 15
- .stockIconFromObject (addStockIcons), 15
- [.GComponent ([.GDefaultWidget), 109
- [.GContainer ([.GDefaultWidget), 109
- [.GDefaultWidget, 109
- [.GDf ([.GDefaultWidget), 109
- [.GEdit ([.GDefaultWidget), 109
- [.GFilter (gfilter), 47
- [.GLayout (glayout), 62
- [.GNotebook (gnotebook), 67
- [.GTable (gtable), 81
- [.GTree (gtree), 90
- [<- .GCheckbox (gcheckbox), 29
- [<- .GComboBox (gcombobox), 34
- [<- .GComponent ([.GDefaultWidget), 109
- [<- .GLayout (glayout), 62
- [<- .GRadio (gradio), 71
- add, 6, 8, 27, 33, 46, 50, 54, 55, 57, 58, 63, 65, 70, 71, 73, 76, 78, 83, 86, 97, 98
- add.GMenuBar (gmenu), 64
- add.GNotebook (gnotebook), 67
- add.GToolBar (gtoolbar), 88
- add.GWindow (gwindow), 98
- add3rdMousePopupMenu (addHandlerChanged.GGraphics), 9

- add3rdmousePopupMenu  
(addHandlerChanged.GGraphics),  
9
- addDragMotion, 7
- addDragMotion  
(addHandlerChanged.GGraphics),  
9
- addDropSource, 7
- addDropSource  
(addHandlerChanged.GGraphics),  
9
- addDropTarget, 7
- addDropTarget  
(addHandlerChanged.GGraphics),  
9
- addHandler, 7, 26, 32, 47, 49, 54, 73, 76, 78,  
83, 86, 92, 96, 98
- addHandler  
(addHandlerChanged.GGraphics),  
9
- addHandlerBlur, 6
- addHandlerBlur  
(addHandlerChanged.GGraphics),  
9
- addHandlerChanged, 6
- addHandlerChanged  
(addHandlerChanged.GGraphics),  
9
- addHandlerChanged.GButton (gbutton), 26
- addHandlerChanged.GCalendar  
(gcalendar), 28
- addHandlerChanged.GCheckbox  
(gcheckbox), 29
- addHandlerChanged.GCheckboxGroup  
(gcheckboxgroup), 31
- addHandlerChanged.GComboBox  
(gcombobox), 34
- addHandlerChanged.GDf (gdf), 37
- addHandlerChanged.GEdit (gedit), 39
- addHandlerChanged.GExpandGroup  
(gexpandgroup), 43
- addHandlerChanged.GGraphics, 9
- addHandlerChanged.GNotebook  
(gnotebook), 67
- addHandlerChanged.GTable (gtable), 81
- addHandlerClicked, 6
- addHandlerClicked  
(addHandlerChanged.GGraphics),  
9
- addHandlerColumnClicked, 6
- addHandlerColumnClicked  
(addHandlerChanged.GGraphics),  
9
- addHandlerColumnDoubleClicked, 6
- addHandlerColumnDoubleClicked  
(addHandlerChanged.GGraphics),  
9
- addHandlerColumnRightClicked, 6
- addHandlerColumnRightClicked  
(addHandlerChanged.GGraphics),  
9
- addHandlerControlClick  
(addHandlerChanged.GGraphics),  
9
- addHandlerDestroy, 6
- addHandlerDestroy  
(addHandlerChanged.GGraphics),  
9
- addHandlerDoubleClick, 6
- addHandlerDoubleClick  
(addHandlerChanged.GGraphics),  
9
- addHandlerDoubleClick.GTable (gtable),  
81
- addHandlerExpose, 7
- addHandlerExpose  
(addHandlerChanged.GGraphics),  
9
- addHandlerFocus, 6
- addHandlerFocus  
(addHandlerChanged.GGraphics),  
9
- addHandlerIdle, 7
- addHandlerIdle  
(addHandlerChanged.GGraphics),  
9
- addHandlerKeystroke, 7
- addHandlerKeystroke  
(addHandlerChanged.GGraphics),  
9
- addHandlerMouseMotion, 7
- addHandlerMouseMotion  
(addHandlerChanged.GGraphics),  
9
- addHandlerRightClick, 6
- addHandlerRightClick

- (addHandlerChanged.GGraphics),  
9
- addHandlerSelect, 6
- addHandlerSelect  
(addHandlerChanged.GGraphics),  
9
- addHandlerSelectionChanged  
(addHandlerChanged.GGraphics),  
9
- addHandlerShiftclick  
(addHandlerChanged.GGraphics),  
9
- addHandlerUnrealize, 6, 14
- addHandlerUnrealize  
(addHandlerChanged.GGraphics),  
9
- addPopupMenu, 7
- addPopupMenu  
(addHandlerChanged.GGraphics),  
9
- addRightclickPopupMenu, 7
- addRightclickPopupMenu  
(addHandlerChanged.GGraphics),  
9
- addSpace (addSpring), 14
- addSpring, 14
- addStockIcons, 15
  
- BasicToolkitInterface  
(guiWidgetsToolkit-class), 94
- BasicToolkitInterface-class  
(guiWidgetsToolkit-class), 94
- blockHandler, 7, 14
- blockHandler  
(addHandlerChanged.GGraphics),  
9
- blockHandlers, 7, 14
- blockHandlers  
(addHandlerChanged.GGraphics),  
9
  
- call\_meth, 17
- check\_deprecated, 18
- check\_return\_class, 18
- Command (guiWidgetsToolkit-class), 94
- Command-class  
(guiWidgetsToolkit-class), 94
- CommandList (guiWidgetsToolkit-class),  
94
  
- CommandList-class  
(guiWidgetsToolkit-class), 94
- CommandStack (guiWidgetsToolkit-class),  
94
- CommandStack-class  
(guiWidgetsToolkit-class), 94
  
- delete, 6
- delete (add), 8
- dim.GComponent ([.GDefaultWidget]), 109
- dimnames.GComponent ([.GDefaultWidget),  
109
- dimnames<- .GComponent  
([.GDefaultWidget), 109
- dispose, 6, 19
- dispose.GBasicDialog (gbasicdialog), 24
- dispose.GNotebook (gnotebook), 67
- dispose.GStackWidget (gnotebook), 67
- dispose.GText (gtext), 84
- dispose.GWindow (gwindow), 98
  
- editable, 5, 19
- editable<-, 5
- editable<- (editable), 19
- enabled, 5, 20
- enabled<-, 5
- enabled<- (enabled), 20
  
- flatten, 20
- focus, 5, 21
- focus<-, 5
- focus<- (focus), 21
- font, 5, 21
- font<-, 5
- font<- (font), 21
  
- gaction, 4, 22
- galert, 5, 23, 24, 25, 36, 61, 66
- gbasicdialog, 5, 24, 24, 25, 36, 61, 66
- gbutton, 4, 26
- gcalendar, 4, 28
- gcheckbox, 4, 29
- gcheckboxgroup, 4, 31
- gcombobox, 4, 34
- gconfirm, 5, 24, 25, 36, 36, 61, 66
- gcontainer, 37
- gdf, 4, 37
- gdfnotebook, 39
- gdroplist (gcombobox), 34

- gedit, 4, 29, 39
- get\_index\_in\_list, 42
- get\_object\_from\_string, 43
- getBlock (getToolkitWidget), 41
- getStockIconByName (addStockIcons), 15
- getStockIcons (addStockIcons), 15
- getToolkitWidget, 6, 41
- getTopLevel (getToolkitWidget), 41
- getWidget (getToolkitWidget), 41
- getWithDefault, 42
- gexpandgroup, 5, 43, 52, 57
- gfile, 5, 45
- gfilebrowse (gfile), 45
- gfilter, 47
- gformlayout, 50, 64
- gframe, 5, 44, 51, 57
- ggraphics, 4, 53
- ggraphicsnotebook, 54
- ggroup, 5, 44, 52, 56
- ghtml, 58
- gimage, 4, 59
- ginput, 5, 60
- glabel, 4, 61
- glayout, 5, 62
- gmenu, 4, 64
- gmessage, 5, 24, 25, 36, 61, 66, 66
- gnotebook, 5, 67
- gpanedgroup, 5, 69
- gprogressbar, 70
- gradio, 4, 71
- gseparator, 4, 74
- gslider, 4, 75, 78
- gspinbutton, 4, 76, 77
- gstackwidget, 5, 69, 79
- gstatusbar, 4, 80
- gtable, 4, 81
- gtext, 4, 84
- gtimer, 4, 7, 9, 87
- gtoolbar, 4, 88
- gtoolkit, 89
- gtree, 4, 90
- guiToolkit, 23, 27, 33, 37, 47, 49, 50, 54, 55, 57, 58, 63, 65, 70, 71, 73, 74, 76, 78–80, 83, 86, 92, 94, 97, 98
- guiWidgetsToolkit-class, 94
- gvarbrowser, 4, 96
- gvbox (ggroup), 56
- gwidget, 97
- gWidgets2-package, 3
- GWidgets2Icons
  - (guiWidgetsToolkit-class), 94
- GWidgets2Icons-class
  - (guiWidgetsToolkit-class), 94
- gwindow, 5, 98
- Handler (guiWidgetsToolkit-class), 94
- Handler-class
  - (guiWidgetsToolkit-class), 94
- insert (gtext), 84
- installing\_gWidgets\_toolkits, 100
- is\_empty, 101
- is\_MacOSX, 102
- is\_Windows, 102
- isExtant, 5, 101
- length.GComponent ([.GDefaultWidget), 109
- length<- .GComponent ([.GDefaultWidget), 109
- names.GComponent ([.GDefaultWidget), 109
- names.GNotebook (gnotebook), 67
- names<- .GComponent ([.GDefaultWidget), 109
- names<- .GFrame ([.GDefaultWidget), 109
- Observable (guiWidgetsToolkit-class), 94
- Observable-class
  - (guiWidgetsToolkit-class), 94
- Observer (guiWidgetsToolkit-class), 94
- observer, 102
- Observer-class
  - (guiWidgetsToolkit-class), 94
- redo, 5, 103
- removeHandler, 7, 14
- removeHandler
  - (addHandlerChanged.GGraphics), 9
- short\_summary, 103
- size, 5, 104
- size<-, 5
- size<- (size), 104
- size<- .GTable (gtable), 81
- stockIconFromClass (addStockIcons), 15
- stockIconFromObject (addStockIcons), 15

- str.GComponent ([.GDefaultWidget), 109
- svalue, 5, 105
- svalue.GButton (gbutton), 26
- svalue.GCalendar (gcalendar), 28
- svalue.GCheckboxGroup (gcheckboxgroup), 31
- svalue.GComboBox (gcombobox), 34
- svalue.GDf (gdf), 37
- svalue.GEdit (gedit), 39
- svalue.GFilter (gfilter), 47
- svalue.GFormLayout (gformlayout), 50
- svalue.GMenuBar (gmenu), 64
- svalue.GRadio (gradio), 71
- svalue.GTable (gtable), 81
- svalue.GText (gtext), 84
- svalue.GTree (gtree), 90
- svalue.GVarBrowser (gvarbrowser), 96
- svalue<-, 5
- svalue<- (svalue), 105
- svalue<- .GCheckbox (gcheckbox), 29
- svalue<- .GGroup (ggroup), 56
- svalue<- .GHtml (ghtml), 58
- svalue<- .GLabel (glabel), 61
- svalue<- .GMenuBar (gmenu), 64
- svalue<- .GNotebook (gnotebook), 67
- svalue<- .GRadio (gradio), 71
- svalue<- .GToolBar (gtoolbar), 88
- svalue<- .GTree (gtree), 90
  
- tag, 6, 106
- tag<-, 6
- tag<- (tag), 106
- tooltip, 5, 107
- tooltip<-, 5
- tooltip<- (tooltip), 107
  
- unblockHandler, 7, 14
- unblockHandler
  - (addHandlerChanged.GGraphics), 9
- unblockHandlers, 7, 14
- unblockHandlers
  - (addHandlerChanged.GGraphics), 9
- undo, 5, 107
- update.GComponent ([.GDefaultWidget), 109
- update.GTree (gtree), 90
  
- visible, 5, 108
- visible.GBasicDialog (gbasicdialog), 24
- visible.GTable (gtable), 81
- visible<-, 5
- visible<- (visible), 108
- visible<- .GDf (gdf), 37
- visible<- .GExpandGroup (gexpandgroup), 43
  
- WSWatcherModel
  - (guiWidgetsToolkit-class), 94
- WSWatcherModel-class
  - (guiWidgetsToolkit-class), 94
  
- XXX, 108