

Package ‘traitr’

February 15, 2012

Type Package

Title An interface for creating GUIs modeled in part after traits UI module for python.

Version 0.13

Author John Verzani <jverzani@gmail.com>

Maintainer John Verzani <jverzani@gmail.com>

URL https://r-forge.r-project.org/R/?group_id=761

Description An interface for creating GUIs modeled in part after the traits UI module for python. The basic design takes advantage of the model-view-controller design pattern. The interface makes basic dialogs quite easy to produce.

Depends digest, proto, gWidgets

Suggests RGtk2, testthat

Extends gWidgetsRGtk2 (>= 0.0-63), gWidgetsQt, gWidgetsctcltk

License GPL (>= 2)

Collate

‘helpers.R’ ‘base.R’ ‘model.R’ ‘view.R’ ‘controller.R’ ‘container.R’ ‘editor.R’ ‘itemgroup.R’ ‘dialog.R’ ‘items.R’ ‘itemlist.R’
package.R’

Repository CRAN

Date/Publication 2012-01-26 07:43:47

R topics documented:

traitR-package	3
aContainer	4
aContext	5
aController	5
Adapter	6

aDialog	6
aFrame	8
aGroup	9
aModel	10
anExpandGroup	11
anItemGroup	12
aNotebook	13
aNotebookPage	14
aPanedGroup	15
aTableLayout	16
aView	17
BaseTrait	17
BooleanEditor	17
ButtonEditor	18
buttonItem	18
choiceItem	19
Container	21
Controller	21
dataframeItem	21
DateEditor	22
dateItem	22
desc	23
dfEditItem	24
Dialog	24
dialogMaker	25
Editor	26
EntryEditor	26
expressionItem	26
FileBrowseEditor	27
fileItem	27
formulaItem	28
get_with_default	29
GraphEditor	29
graphicDeviceItem	29
ImageEditor	30
imageItem	31
integerItem	32
Item	33
ItemGroup	33
itemList	34
ItemListEditor	35
LabelEditor	35
labelItem	36
loadingAnimation	36
merge.list	37
Model	38
numericItem	38
ObjectWithValuesEditor	39

traitR-package 3

param	39
RangeEditor	40
rangItem	40
returns	41
SeparatorEditor	41
separatorItem	42
stringItem	42
TableEditor	43
tableItem	44
trueFalseItem	45
ul	46
variableSelectorItem	46
View	47
wrap_in_tag	48

Index 49

traitR-package *An interface for GUI creation using gWidgets...*

Description

An interface for GUI creation using gWidgets

Details

This package provides an alternate interface for creating graphical user interfaces. The design was inspired by the Traits UI module for python developed by enthought.com. The implementation uses the MVC design pattern in the background, although the user need not be aware of this.

For basic use, the user creates a bunch of items (the model), specifies how these will be layed out in a simple manner (the view), specifies actions to happen (the controller) and then creates a dialog. See [aDialog](#) for examples.

Creating basic dialogs requires no actual GUI programming knowledge. One specifies the items by type of variable (numericItem or stringItem, say) and the "action" through a method call.

The package uses the **proto** package so at some level, the R user must use that OO syntax. In particular, methods calls are done with the notation `obj$method_name` and method definitions have an initial argument `.` for passing in a reference to the proto object. (The name `.` is a convention, but can be changed to `self` or `this`, if that naming convention is preferred. Methods for proto objects are documented (to some degree anyways). Their help page is shown through the method `show_help`, as in `obj$show_help()`.

aContainer	<i>A container to give a different context than the default for a set of items...</i>
------------	---

Description

A container to give a different context than the default for a set of items

Usage

```
aContainer(..., context, attr=list(), enabled_when, visible_when)
```

Arguments

context	ItemGroup or item to get context from. Typically just NULL.
attr	gWidget values passed to constructor
enabled_when	Method to determine when items in container should be enabled
visible_when	Method to determine when items in container should be visible
...	children items specified by character strings

Value

Returns a proto object. Call `obj$show_help()` to view its methods and properties.

See Also

[Container](#)

Examples

```
i <- aDialog(items=list(x=numericItem(1), y=stringItem("a")))
lay <- aContainer("x","y")
## Not run: i$make_gui(gui_layout=lay)
## how to do enabled when
lay <- aContainer("x",
aContainer("y", enabled_when=function(.) .$get_x() > 1))
j <- i$instance()
## Not run: j$make_gui(gui_layout=lay)
## visible can be used to hide values if not needed
i <- aDialog(items=list(x=numericItem(1), y=stringItem("a")))
lay <- aContainer("x","y")
## Not run: i$make_gui(gui_layout=lay)
## how to do enabled when
lay <- aContainer("x",
aContainer("y", visible_when=function(.) .$get_x() > 1))
k <- i$instance()
## Not run: k$make_gui(gui_layout=lay)
```

aContext	<i>A container to give a different context than the default for a set of items...</i>
----------	---

Description

A container to give a different context than the default for a set of items

Usage

```
aContext(..., context, attr=list(), enabled_when, visible_when)
```

Arguments

context	ItemGroup or item to get context from. Typically just NULL.
attr	gWidget values passed to constructor
enabled_when	Method to determine when items in container should be enabled
visible_when	Method to determine when items in container should be visible
...	children items specified by character strings

Details

The basic container uses the calling model (a dialog or item group) as its context. This allows the context to be overridden, which might be desirable if the items are in more than one dialog.

Value

Returns a proto object. Call obj\$show_help() to view its methods and properties.

See Also

[Container](#)

aController	<i>Constructor for a Controller proto objects...</i>
-------------	--

Description

Constructor for a Controller proto objects

Usage

```
aController(...)
```

Arguments

... passed to proto method for

Details

Simply provides a more typical calling interface for the Controller proto object

Value

returns the Controller object

Adapter	<i>Trait for Adapter object...</i>
---------	------------------------------------

Description

Trait for Adapter object An adapter is a simple controller connecting one model property with a widget in a view by default the adapter synchronizes changes

aDialog	<i>Create a Dialog instance...</i>
---------	------------------------------------

Description

Create a Dialog instance

Usage

```
aDialog(items=list(), title="", help_string="", buttons=c("OK",
"SPACE", "Cancel", "Help"), ...)
```

Arguments

items	List of item instances to create the model for the dialog object. May also be an item group (anItemGroup).
title	Title of dialog
help_string	String for default Help button
buttons	Character vector of button names. "OK","Cancel","Help","Undo","Redo" are some standard ones. "SPACE" and "SPRING" adjust the layout.
...	How to pass in other properties and methods of the dialog object. For example OK_handler.

Details

A dialog is like an item group, in that it combines items into a model. However, an item group is meant to be incorporated into other GUIs, whereas a dialog creates its own window and decorations. A dialog has default buttons, and options for adding in menubars, toolbars, and statusbars. The choice of buttons can be specified at construction.

Methods:

The main method that a dialog has is its `OK_handler` which is a method called when the "OK" button is clicked (one of the default buttons).

The getters and setters for the main value for an item are `get_NAME` and `set_NAME`, where `NAME` is the item name. The name is specified when the item is constructed (through its `name` property) or more conveniently, taken from the name of the component in the `items` list that defines the items for dialog or item group.

The method `to_R` returns the items' values as a list (useful in combination with `do.call`).

The method `get_item_by_name` returns an item object by its name. (Names should be unique.) This is useful if more properties than the main one for an item are needed to be set. (The main value is set via the setter.) The example shows how the `validate` property of some items can be set.

The method `is_valid` is `TRUE` if all items validate and `FALSE` otherwise.

The method `model_value_changed(.)` is called whenever an item property is changed either through the GUI. A dialog observes itself.

For each item one can listen for changes with the method `property_NAME_value_changed(. , value, old_value)`.

Properties that are of interest:

1. `status_text` If non-NULL, when GUI is drawn, a status bar will be made with this text. The method `set_status_text` can be used to update the status
2. `menu_list` A menu list to specify a menubar. (See [gmenu](#).)
3. `toolbar_list` A menu list to specify a toolbar. (See [gtoolbar](#).)
4. `buttons` A list of buttons names. The default is `c("OK", "SPACE", "Cancel", "Help")`. The special names `SPACE` and `SPRING` adjust their positioning, otherwise the values are button names. When a button is clicked, the handler `buttonname_handler` is called, where the `buttonname` is stripped on non-alphanumeric characters. The basic buttons and Redo and Undo have default handlers. Likely, only `OK_handler` will need redefining. The property `default_button` can be specified to make a button the default one (so that it is activated when a user presses the enter key).

Value

Returns a proto object. See its `show_help` method for details.

Examples

```
##
## a Basic example
dlg <- aDialog(items=list(
```

```

a = numericItem(0),
b = stringItem("a")
),
title="The title",
help_string="Help on this dialog"
)
## Not run: dlg$make_gui()
##
##
## example with model_value_changed
plotIt <- function(n, mean, sd, ...) hist(rnorm(n, mean, sd))
dlg <- aDialog(items=list(
n = integerItem(10),
mean = numericItem(0),
sd = numericItem(1),
out=graphicDeviceItem()
),
buttons="Cancel",
model_value_changed=function(.) if(!.$is_valid()) do.call("plotIt", .$to_R())
)
##
## validation for n, sd
n <- dlg$get_item_by_name("n")
n$validate <- function(., rawvalue) if(rawvalue <= 1) stop("n must be positive integer") else rawvalue
sd <- dlg$get_item_by_name("sd")
sd$validate <- function(., rawvalue) if(rawvalue <= 0) stop("sd must be positive") else rawvalue
## Not run: dlg$make_gui()
##
##
## subtle point about scope. Proto methods can be defined via $<- or [[<- but there is a difference.
## $<- does not have lexical scope whereas [[<- does. The $<- might be more natural to type,
## but [[<- might be more natural to use. In this example, The "b" button does not work, as it can't find the
## function a -- the frame of evaluation is the environment dlg (not its enclosing frame).
## Thanks to Gabor for his help with this.
scope_example <- function() {
a <- function(...) print("hi")
dlg <- aDialog(items=list(),
buttons=c("a","b","c"),
a_handler=function(.) a(), ## like [[<-, not $<-
title="a, c work; b doesn't"
)
dlg$b_handler <- function(.) a() ## $<- has unbound variables found in dlg
dlg[['c_handler']] <- a ## [[<- uses lexical scope for unbound variables
}
## Not run: scope_example()
## See ?anItemGroup for an example of a modal dialog

```

Description

Box container with label and visual separator to indicate grouping

Usage

```
aFrame(..., label="frame label", horizontal=FALSE, spacing=10, context,
        attr=list(), enabled_when, visible_when)
```

Arguments

label	label for frame
horizontal	If TRUE left to right, if FALSE top to bottom
spacing	Space in pixels between items
context	ItemGroup or item to get context from. Typically just NULL.
attr	gWidget values passed to constructor
enabled_when	Method to determine when items in container should be enabled
visible_when	Method to determine when items in container should be visible
...	children items specified by character strings

Value

Returns a proto object. Call `obj$show_help()` to view its methods and properties.

See Also

[Container](#)

Examples

```
i <- aDialog(items=list(x=numericItem(1), y=stringItem("a")))
lay <- aFrame(label="label frame",
             aContainer("x", "y"))
## Not run: i$make_gui(gui_layout=lay)
```

aGroup

A box container.

Description

A box container. Packs in items left to right or top to bottom

Usage

```
aGroup(..., horizontal=TRUE, spacing=10, context, attr=list(),
        enabled_when, visible_when)
```

Arguments

horizontal	If TRUE left to right, if FALSE top to bottom
spacing	Space in pixels between items
context	ItemGroup or item to get context from. Typically just NULL.
attr	gWidget values passed to constructor
enabled_when	Method to determine when items in container should be enabled
visible_when	Method to determine when items in container should be visible
...	children items specified by character strings

Value

Returns a proto object. Call `obj$show_help()` to view its methods and properties.

See Also

[Container](#)

Examples

```
i <- aDialog(items=list(xlong=numericItem(1), y=stringItem("a")))
lay <- aGroup("xlongname","y", horizontal=FALSE) # not in nice layout
## Not run: i$make_gui(gui_layout=lay)
```

aModel

Constructor for a Model proto objects...

Description

Constructor for a Model proto objects

Usage

```
aModel(...)
```

Arguments

... passed to proto method for

Details

Simply provides a more typical calling interface for the Model proto object

Value

returns the Model object. Call `obj$show_help()` to view its methods and properties.

anExpandGroup	<i>Expanding group.</i>
---------------	-------------------------

Description

Expanding group. Has trigger to show/hide its children

Usage

```
anExpandGroup(..., label="", horizontal=FALSE, expanded=TRUE, context,  
  attr=list(), enabled_when, visible_when)
```

Arguments

label	label for trigger
horizontal	If TRUE left to right, if FALSE top to bottom
expanded	Initial state of children. Set to TRUE to show
context	ItemGroup or item to get context from. Typically just NULL.
attr	gWidget values passed to constructor
enabled_when	Method to determine when items in container should be enabled
visible_when	Method to determine when items in container should be visible
...	children items specified by character strings

Value

Returns a proto object. Call `obj$show_help()` to view its methods and properties.

See Also

[Container](#)

Examples

```
i <- aDialog(items=list(x=numericItem(1), y=stringItem("a")))
lay <- anExpandGroup(label="label frame",
  aContainer("x", "y"))
## Not run: i$make_gui(gui_layout=lay)
```

 anItemGroup

Constructor for ItemGroup instances...

Description

Constructor for ItemGroup instances

Usage

```
anItemGroup(items=list(), name, ...)
```

Arguments

items	List of Item instances or ItemGroup instances
name	Name of ItemGroup.
...	Passed to ItemGroup proto trait

Details

An ItemGroup creates a model with properties given by the items and a default layout for its items. This can also be specified when the layout is drawn through `make_gui`.

An item group bundles a list of items into a model. When the model is initialized, constructors to access the model values are created. These getters/setters use the item names, so that `get_name` will get the main value for the item with name attribute "name".

An item group has the useful methods `to_R` to return the values in the model as a named list and `get_item_by_name` to get the item from the list of items matching the name.

Value

A proto object. Call `obj$show_help()` to view its methods and properties.

See Also

[aContainer](#) for specifying a layout

Examples

```
## Not run:
## make a simple item group, show in non-default layout
i <- anItemGroup(items=list(
  numericItem(0,"x"),
  numericItem(0,"y"),
  stringItem("", "z")
))
lay <- aContainer("x","y", aFrame("z", label="z in a box"))
## some proto methods:
i$make_gui(cont=gwindow("Example of itemGroup"), gui_layout=lay)
```

```

i$get_x()    # get x value
i$set_x(10) # set x value to 10
i$to_R()    # get list of x,y,z values

## End(Not run)

## example of using an item group and gbasicdialog to make a modal GUI
ig <- anItemGroup(items=list(
  x=numericItem(2)
)
)

## using gbasicdialog from gWidgets
## Not run:
w <- gbasicdialog("testing", handler=function(h,...) {
  . <- h$action          # action passes in itemgroup
  .$output <- sin(.$get_x())
},
action=ig)
ig$make_gui(container=w)
visible(w, TRUE) ## modal now
print(ig$output)

## End(Not run)

```

aNotebook

A notebook container.

Description

A notebook container.

Usage

```

aNotebook(..., close_buttons=FALSE, initial_page=1, context,
  attr=list(expand = TRUE), enabled_when, visible_when)

```

Arguments

close_buttons	Logical indicating if close buttons should be added (RGtk2 only)
initial_page	Which page to open on
context	ItemGroup or item to get context from. Typically left as NULL.
attr	gWidget values passed to constructor
enabled_when	Method to determine when items in container should be enabled
visible_when	Method to determine when items in container should be visible
...	children items specified by character strings

Details

Pages of notebook are made with aNotebookPage container, which in turn can hold other items.

Value

Returns a proto object. Call obj\$show_help() to view its methods and properties.

See Also

[Container](#)

Examples

```
## Not run:
i <- aDialog(items=list(x=numericItem(1), y=stringItem("a")))
lay <- aNotebook(
  aNotebookPage(label="page 1", "x"),
  aNotebookPage(label="page 2", "y")
)
i$make_gui(gui_layout=lay)

## End(Not run)
```

aNotebookPage	<i>A page in a notebook...</i>
---------------	--------------------------------

Description

A page in a notebook

Usage

```
aNotebookPage(..., label, context, attr=list(), enabled_when,
  visible_when)
```

Arguments

label	Tab label for notebook page
context	ItemGroup or item to get context from. Typically just NULL.
attr	gWidget values passed to constructor
enabled_when	Method to determine when items in container should be enabled
visible_when	Method to determine when items in container should be visible
...	children items specified by character strings

Details

Container to hold a page within a notebook container

Value

Returns a proto object. Call `obj$show_help()` to view its methods and properties.

aPanedGroup	<i>A two panel paned group container.</i>
-------------	---

Description

A two panel paned group container.

Usage

```
aPanedGroup(..., horizontal=TRUE, context, attr=list(), enabled_when,
             visible_when)
```

Arguments

horizontal	If TRUE left to right, if FALSE top to bottom
context	ItemGroup or item to get context from. Typically just NULL.
attr	gWidget values passed to constructor
enabled_when	Method to determine when items in container should be enabled
visible_when	Method to determine when items in container should be visible
...	children items specified by character strings

Value

Returns a proto object. Call `obj$show_help()` to view its methods and properties.

See Also

[Container](#)

Examples

```
## Not run:
i <- aDialog(items=list(x=numericItem(1), y=stringItem("a")))
lay <- aPanedGroup("x","y") ## just two children,
i$make_gui(gui_layout=lay)
## can put other children into a container to make just two children for aPanedGroup instance
j <- aDialog(items=list(x=numericItem(1), y=stringItem("a"), z=trueFalseItem(TRUE, label="check me")))
lay <- aPanedGroup("x", aContainer("y", "z"))
j$make_gui(gui_layout=lay)

## End(Not run)
```

aTableLayout	<i>A container for tabular layout...</i>
--------------	--

Description

A container for tabular layout

Usage

```
aTableLayout(..., no_cols=1, context, attr=list(), enabled_when,  
             visible_when)
```

Arguments

no_cols	Number of columns. Fills in row by row.
context	ItemGroup or item to get context from. Typically just NULL.
attr	gWidget values passed to constructor
enabled_when	Method to determine when items in container should be enabled
visible_when	Method to determine when items in container should be visible
...	children items specified by character strings

Details

The basic container has one column for the item's labels and one column for the item's editors.

Value

Returns a proto object. Call `obj$show_help()` to view its methods and properties.

See Also

[aContainer](#) constructor, [Container](#) base trait

Examples

```
## simple example  
i <- aDialog(items=list(x=numericItem(1), y=stringItem("a")))  
lay <- aTableLayout("x","y", no_cols=2)  
## Not run: i$make_gui(gui_layout=lay)
```

aView	<i>Constructor for a View proto object...</i>
-------	---

Description

Constructor for a View proto object

Usage

aView(...)

Arguments

... passed to proto method for

Details

Simply provides a more typical calling interface for the View proto object

Value

Returns the View object. Call obj\$show_help() to view its methods and properties.

BaseTrait	<i>Base Trait to place common properties and methods...</i>
-----------	---

Description

Base Trait to place common properties and methods

BooleanEditor	<i>Trait for Editor for TRUE/FALSE selection...</i>
---------------	---

Description

Trait for Editor for TRUE/FALSE selection

Details

Editor has regular or compact style

ButtonEditor	<i>Trait for button editor...</i>
--------------	-----------------------------------

Description

Trait for button editor

buttonItem	<i>Button item to initiate an action...</i>
------------	---

Description

Button item to initiate an action

Usage

```
buttonItem(value="button label", action, name, label=name, help="",
           tooltip="", attr, model, editor, ...)
```

Arguments

value	Default value for the model
action	function to call when clicked. Signature is <code>function(., h, ...) {}</code> (like <code>gWidgets</code> with extra leading <code>.</code>). The <code>"."</code> is the button item, not the itemgroup or dialog that this item may be a part of. When that is the case, <code>.\$parent</code> refers to the parent itemgroup or dialog. The evaluation environment is not that where the action is defined. This can lead to unexpected sources of error.
name	Required name for object. Names should be unique within a group of items
label	Optional label, default value is the name. Use <code>""</code> to have not label text.
help	Optional help string
tooltip	Optional tooltip to display
attr	A list of attributes to pass to widget on construction. Eg. <code>attr=list(size=c(100,200))</code>
model	Optional model. Useful if one wishes to use same model for multiple views
editor	Specification of editor (a view) to override default
...	Passed to parent proto object during call to proto

Details

While dialogs have a `buttons` property for the main buttons, this item allows other buttons to be used within a dialog. One must define an action (a callback) to call when the button is clicked. There are some issues with how this method is defined and where it is evaluated.

Value

A proto object. Call `obj$show_help()` to view its methods and properties.

See Also

[Item](#)

Examples

```
## basic button. Note the extra "." compared to gWidgets handler
b <- buttonItem("click me", action=function(.,h,...) {
  print("hi")
})
## An example within a dialog
dlg <- aDialog(items=list(
  a = stringItem(""),
  b = buttonItem("Click me", label="", action=function(., h, ...) {
    galert(sprintf("Item a is %s\n", .$parent$get_a()))
  })
),
  title="A dialog with a button item",
  buttons=c()           # no standard buttons
)
## Not run: dlg$make_gui()
```

choiceItem

Item for choosing one of several values...

Description

Item for choosing one of several values

Usage

```
choiceItem(value="", values="", by_index=FALSE, multiple=FALSE,
  editable=FALSE, name, label=name, help="", tooltip="", attr, model,
  editor, editor_type=c("", "gradio", "gcombobox", "gtable", "gedit",
  "gcheckboxgroup"), ...)
```

Arguments

value	Default value for the model. This is specified by index (or indices if <code>multiple=TRUE</code>)
values	Values that one can select from. May be a data frame or vector. The editor depends on the size of this: small will be radio button or checkboxes; medium is combobox; large is a table. One can override the behaviour by passing in a value to <code>editor_type</code> .
by_index	Do we get and set the main value by index or by value?

multiple	Multiple selection is allowed? If so, then only checkboxes or table widget is used
editable	Can user edit value to be selected? If so, the combobox is used
name	Required name for object. Names should be unique within a group of items
label	Optional label, default value is the name
help	Optional help string
tooltip	Optional tooltip to display
attr	A list of attributes to pass to widget on construction. Eg. attr=list(size=c(100,200)). The expand=TRUE value is a default for this.
model	Optional model. Useful if one wishes to use same model for multiple views
editor	Specification of editor (a view) to override default
editor_type	override choice of editor by heuristic based on the number of possible values. Must set attr to match desired.
...	Passed to parent proto object during call to proto

Value

A proto object. Call obj\$show_help() to view its methods and properties.

See Also

[Item](#)

Examples

```
## default is to get/set by value
a <- choiceItem("a", letters, name="x")
a$get_x()
a$set_x("b")
a$get_x()
## or by index, which can be easier to do
b <- choiceItem("a", letters, name="x", by_index=TRUE)
b$get_x()
b$set_x(2)
b$get_x()
## Size determines widget, unless you set editor_type
## a radio group
rg <- choiceItem("a", letters[1:3], name="x")
## a combobox
cb <- choiceItem("a", letters[1:8], name="x")
## a table
tb <- choiceItem("a", letters[1:26], name="x")
## adjust size of table widget
tb <- choiceItem("a", letters[1:26], name="x", attr=list(size=c(width=300,height=400)))
## Multiple and size determines widget type
## smaller uses checkboxgroup
cbg <- choiceItem("a", letters[1:5], multiple=TRUE)
## larger uses table
```

```
tbl <- choiceItem("a", letters[1:15], multiple=TRUE)
## place values in data frame to avoid generic header
tbl <- choiceItem("a", data.frame("Column header"=letters[1:15]), multiple=TRUE)
```

Container	<i>Base Trait for Container objects.</i>
-----------	--

Description

Base Trait for Container objects. Containers are used to make views.

Details

Basic container is a layout object for tabular layout There are various types of layouts. The most basic, and default view, is simply `aContainer(...names of items...)` which simply uses a table to display the item's label and editor. Other containers can be used to adjust this.

Containers have a few methods, notably the `is_visible` and `is_enabled` methods which can be used to hide or set sensitive to user input the container's components.

Controller	<i>Trait for Controller objects...</i>
------------	--

Description

Trait for Controller objects

Details

A controller connects a model and an associated view to synchronize changes in one with another This implementation rests on the controller having some suitably named methods

dataframeItem	<i>Item to select a data frame from the available data frames in...</i>
---------------	---

Description

Item to select a data frame from the available data frames in `.GlobalEnv`

Usage

```
dataframeItem(value="", name, label=name, help="", tooltip="", attr,
              model, editor, ...)
```

Arguments

value	Default data frame for the model, defaults to .GlobalEnv
name	Required name for object. Names should be unique within a group of items
label	Optional label, default value is the name
help	Optional help string
tooltip	Optional tooltip to display
attr	A list of attributes to pass to widget on construction. Eg. attr=list(size=c(100,200)) This widget uses a gtable instance and specifying the size is suggested
model	Optional model. Useful if one wishes to use same model for multiple views
editor	Specification of editor (a view) to override default
...	Passed to parent proto object during call to proto

Details

This widget checks every second or so for new data frames and updates selection accordingly

Value

A proto object. Call obj\$show_help() to view its methods and properties.

DateEditor	<i>Trait for data selection editor..</i>
------------	--

Description

Trait for data selection editor

dateItem	<i>A calendar date selection item..</i>
----------	---

Description

A calendar date selection item

Usage

```
dateItem(value="", format_string, name, label=name, help="",
         tooltip="", attr, model, editor, ...)
```

Arguments

value	Default data frame for the model
format_string	String to specify format of date to return. See strftime for codes. default value is '%Y-%m-%d'
name	Required name for object. Names should be unique within a group of items
label	Optional label, default value is the name
help	Optional help string
tooltip	Optional tooltip to display
attr	A list of attributes to pass to widget on construction. Eg. <code>attr=list(size=c(100,200))</code>
model	Optional model. Useful if one wishes to use same model for multiple views
editor	Specification of editor (a view) to override default
...	Passed to parent proto object during call to proto

Value

A proto object. Call `obj$show_help()` to view its methods and properties.

See Also

[Item](#)

Examples

```
d <- dateItem(name="d") ## basic usage, no initial date.
# specify initial date and reformat -- can't start in that format, it is ambiguous
d <- dateItem('2000-12-25', format_string='%m-%d-%Y', name='d')
```

desc	<i>specify a description for documentation...</i>
------	---

Description

specify a description for documentation

Usage

```
desc(...)
```

Arguments

... Values pasted in return a string

Details

internal function for writing proto docs.

dfEditItem	<i>data frame editor item.</i>
------------	--------------------------------

Description

data frame editor item. Needs writing

Usage

```
dfEditItem(...)
```

Arguments

... to be replaced with actual arguments

Details

Write me

Value

A proto object. Call `obj$show_help()` to view its methods and properties.

Dialog	<i>A Dialog wraps a top-level window around a collection of items which may be item groups...</i>
--------	---

Description

A Dialog wraps a top-level window around a collection of items which may be item groups

Details

One can specify the parent when making the UI Buttons are specified through the buttons property

dialogMaker	<i>Automatically create a dialog for a function...</i>
-------------	--

Description

Automatically create a dialog for a function

Usage

```
dialogMaker(f, title="Dialog", help_string="", make_gui=TRUE,
            add_graphic_device=FALSE, ...)
```

Arguments

f	function to make dialog for. Its arguments must be specified in a certain way.
title	Title for dialog window
help_string	String for help information
make_gui	If TRUE or add_graphic_device=TRUE then call dialogs make_gui method
add_graphic_device	If TRUE add an graphicDeviceItem to dialog
...	passed to make_gui when no graphic device asked for

Details

Function must have a special markup for its argument. A named argument a..b is interpreted with b determining the type of item to use. We support numeric, string, choice, range, ??? Within the body of the function, the variable a..b should be referred to by a. The idea is that you write and debug the function as usual, then simply modify the argument list to include the types. This function will not work for functions whose arguments use lazy evaluation referring to other argument's values.

All arguments should have a default A choice items should have its default with a vector. The first argument is the selected one A range item is specified with values c(from=., to=..., by=..., [value=from]). If value not give, then from is used. The OK_handler will call f.

Value

Returns an instance of aDialog.

Examples

```
f <- function(x..numeric=1, y..string="a") print(list(x,y))
## Not run: dialogMaker(f)
## can have missing arguments
f <- function(x, y..numeric=1) print(list(x,y))
## Not run: dialogMaker(f)
## a choice item. Sizing is funny for tables
f <- function(x..choice=letters) print(x)
```

```
## Not run: dialogMaker(f)
## range items
f <- function(x..numeric=0, mu..numeric=0,
  alternative..choice=c("two.sided","less","greater"),
  conf.level..range=c(.80,1.00, .01, .95)) {
  out <- capture.output(t.test(x, alt=alternative, conf.level=conf.level))
  print(out)
}
## Not run: dialogMaker(f, title="CI from t.test with summarized values")
```

 Editor

Base Trait for Editor.

Description

Base Trait for Editor.

Details

An editor is a basic view for a widget, essentially a map from gedit, say, to a view.

 EntryEditor

A Base Trait for an editor using the entry widget...

Description

A Base Trait for an editor using the entry widget

 expressionItem

Item for typing in R expressions.

Description

Item for typing in R expressions. These are eval-parsed in .GlobalEnv prior to return

Usage

```
expressionItem(value="", name, label=name, help="", tooltip="", attr,
  model, editor, ...)
```

Arguments

value	Default value for the model
name	Required name for object. Names should be unique within a group of items
label	Optional label, default value is the name
help	Optional help string
tooltip	Optional tooltip to display
attr	A list of attributes to pass to widget on construction. Eg. attr=list(size=c(100,200))
model	Optional model. Useful if one wishes to use same model for multiple views
editor	Specification of editor (a view) to override default
...	Passed to parent proto object during call to proto

Value

A proto object. Call obj\$show_help() to view its methods and properties.

See Also

[numericItem](#), [integerItem](#), [stringItem](#), as these are similar, but also validate the final results

FileBrowseEditor	<i>Trait for making File browser editor...</i>
------------------	--

Description

Trait for making File browser editor

fileItem	<i>A file selection item...</i>
----------	---------------------------------

Description

A file selection item

Usage

```
fileItem(value="", name, label=name, help="", tooltip="", attr, model,
         editor, ...)
```

Arguments

value	Default data frame for the model
name	Required name for object. Names should be unique within a group of items
label	Optional label, default value is the name
help	Optional help string
tooltip	Optional tooltip to display
attr	A list of attributes to pass to widget on construction. Eg. attr=list(size=c(100,200))
model	Optional model. Useful if one wishes to use same model for multiple views
editor	Specification of editor (a view) to override default
...	Passed to parent proto object during call to proto

Value

A proto object. Call obj\$show_help() to view its methods and properties.

formulaItem	<i>A formula Item...</i>
-------------	--------------------------

Description

A formula Item

Usage

```
formulaItem(value="", dataframeItem, name, label=name, help="",
            tooltip="", attr, model, editor, ...)
```

Arguments

value	Default data frame for the model
dataframeItem	A required dataframeItem instance. This need not be in same display, or even displayed
name	Required name for object. Names should be unique within a group of items
label	Optional label, default value is the name
help	Optional help string
tooltip	Optional tooltip to display
attr	A list of attributes to pass to widget on construction. Eg. attr=list(size=c(100,200)) This widget uses a gtable instance and specifying the size is suggested
model	Optional model. Useful if one wishes to use same model for multiple views
editor	Specification of editor (a view) to override default
...	Passed to parent proto object during call to proto

Value

A proto object. Call obj\$show_help() to view its methods and properties.

get_with_default	<i>Function to return value or an object (or default if value is null, NA or "")...</i>
------------------	---

Description

Function to return value or an object (or default if value is null, NA or "")

Usage

```
get_with_default(x, default)
```

Arguments

x	object
default	default value

Value

Returns default if x is NA, null or "", otherwise x

GraphEditor	<i>Trait for embedding graphics (Qt, RGtk2 only)...</i>
-------------	---

Description

Trait for embedding graphics (Qt, RGtk2 only)

graphicDeviceItem	<i>A graphic device item.</i>
-------------------	-------------------------------

Description

A graphic device item. (Only with RGtk2 and cairoDevice!)

Usage

```
graphicDeviceItem(value="", name, label=name, help="", tooltip="",
  attr=list(size = c(480, 480)), model, editor, ...)
```

Arguments

value	ingored
name	Required name for object. Names should be unique within a group of items
label	Optional label, default value is the name
help	Optional help string
tooltip	Optional tooltip to display
attr	A list of attributes to pass to widget on construction. Eg. attr=list(size=c(100,200)).
model	ignored
editor	ignored
...	Passed to parent proto object during call to proto

Details

This device will become the current one if the mouse clicks in the window, This isn't perfect, but should be easy enough to get used to. This only works with gWidgetsRGtk2, gWidgetsQt

Value

A proto object. Call obj\$show_help() to view its methods and properties.

Note

With **gWidgetsRGtk2**, there is some thing odd that causes a display to pop up before the cairo Device if no devices are open.

See Also

[Item](#)

Examples

```
graphIt <- function(n, ...) hist(rnorm(n))
dlg <- aDialog(items=list(n=integerItem(10), out=graphicDeviceItem()),
model_value_changed=function(.) do.call("graphIt", .$to_R()) ## ... allows out to pass in unnoticed
)
## Not run: dlg$make_gui()
graphIt(dlg$get_n()) ## initial graphic

## End(Not run)
```

ImageEditor

Trait for embedding an image file...

Description

Trait for embedding an image file

imageItem	<i>Display an image specified by its filename.</i>
-----------	--

Description

Display an image specified by its filename.

Usage

```
imageItem(value="", name, label=name, help="", tooltip="", attr=list(),
          model, editor, ...)
```

Arguments

value	name of file
name	Required name for object. Names should be unique within a group of items
label	ignored
help	Optional help string
tooltip	Optional tooltip to display
attr	A list of attributes to pass to widget on construction. Eg. attr=list(size=c(100,200)).
model	ignored
editor	ignored
...	Passed to parent proto object during call to proto

Value

A proto object. Call obj\$show_help() to view its methods and properties.

See Also

[Item](#)

Examples

```
img <- system.file("images/plot.gif", package="gWidgets") ## some image
i <- imageItem(img) ## constructor
## Not run: i$make_ui(container=gwindow("Image")) ## show item directly
```

integerItem	<i>Item for integers...</i>
-------------	-----------------------------

Description

Item for integers

Usage

```
integerItem(value=integer(0), name, label=name, help="", tooltip="",
            eval_first=FALSE, attr, model, editor, ...)
```

Arguments

value	Default value for the model
name	Required name for object. Names should be unique within a group of items
label	Optional label, default value is the name
help	Optional help string
tooltip	Optional tooltip to display
eval_first	Should value be run through eval/parse before coercion.
attr	A list of attributes to pass to widget on construction. Eg. attr=list(size=c(100,200))
model	Optional model. Useful if one wishes to use same model for multiple views
editor	Specification of editor (a view) to override default
...	Passed to parent proto object during call to proto

Value

A proto object. Call obj\$show_help() to view its methods and properties.

See Also

[numericItem](#)

Item

Base Trait for an Item...

Description

Base Trait for an Item

Details

An Item combines a model, view and controller interface into one convenient package. Items may be combined into an ItemGroup or a Dialog to be shown.

The `make_ui` method creates the user interface, initializes the model and the controller. The `init_model`, `init_controller` and `init_ui` do the work.

The model may be shared with different items. See `set_model_from_item` or the `instance` proto method.

Items implement the observer interface, so one can add observers to listen for changes to the properties. (Properties are listed in the property "properties".)

Items use the Adapter interface to link the model with the view (an Editor). The "properties" property lists the names of model properties. One should use "value" for the special one to be returned by the method `to_R`. (This method gathers values from the items after coercion)

When an item's user interface is made, the method `on_realized` is called.

ItemGroup

Base Trait to group items together to form a model.

Description

Base Trait to group items together to form a model. ItemGroups may be viewed as a model, view and controller bundled together in a tidy package.

Details

An item group is a collection of Item instances. These are specified through the `items` property as a list.

ItemGroups implement the observer pattern, so are models and can have observers listen for changes. ItemGroups observe themselves to update the user interface on model changes. One can add other observers if desired. See the `init` method for an example.

If a method `model_value_changed` is defined, then it will be called with the ItemGroup instance being in whenever a property of the model has a new value. The handlers `property_NAME_value_changed` is called when the main value in item NAME is changed. (An item can have several properties, one of which is the main one.)

ItemGroups have a `make_gui` method to make a view of the model. The layout of this GUI can be specified through its `gui_layout` argument, or by default have a simple table layout. ItemGroup instances are meant to be embedded into a GUI, so the `cont` argument is needed to pass in the desired container.

itemList	<i>An itemList is used to store a list of similar items or itemgroups with a means to edit individually...</i>
----------	--

Description

An itemList is used to store a list of similar items or itemgroups with a means to edit individually

Usage

```
itemList(items=list(), items_name="", item_factory, name, label=name,
         help="", tooltip="", attr=list(), model, editor, ...)
```

Arguments

items	list of similar items, may be empty list
items_name	Header name on top of table displaying item list
item_factory	function to call to produce a new item, e.g. function(.) numericItem(1)
name	name of itemList object
label	label for itemList object
help	help string
tooltip	tooltip
attr	attributes passed to make_ui constructor
model	optional model to pass in
editor	optional editor to pass in
...	passed along to Item\$proto() call

Value

A proto object. Call obj\$show_help() to view its methods and properties.

Note

This item's model is a list storing child items or item groups. To create new items, the `item_factory` method should be provided. It provides a template for a new item, the editor allows the user to modify its values. When a child item is edited the "done" button is clicked to close. The method `post_process` is called. (The edited changes may already have been sent back to the model.) The child items `to_string` method is called to make the label in the table that allows the user to select the child item to edit. This should be a character vector of length 1. The table can display an icon. Simply set the `icon` property of the icon to a **gWidgets** stock icon name.

The child items can be returned via the `get_value` method or the `get_NAME` method, where NAME is that passed into the name argument of the constructor. The `to_R` method can be modified to manipulate the return value. The vignette has an example where the output is coerced into a data frame. The default is a list with each child items `to_R` method called to form the numbered components.

Examples

```
## Not run:
## make icons
imagedir <- system.file("images",package="traitr")
addStockIcons(gsub("\\.png","", list.files(path=imagedir)),
list.files(path=imagedir, full.names=TRUE))
## make item
item <- itemList(items=list(),
items_name="Personnel",
item_factory = function(.) {
a <- anItemGroup(items=list(
name=stringItem(""),
rank=choiceItem("Scout", values=c("Scout","Captain","General")),
serial.number = stringItem("", label="Serial number")))
a$post_process <- function(.) {
.$icon <- tolower(.$get_rank())
}
a$to_string <- function(., drop=TRUE) .$to_R()$name
return(a)
},
name="itemlist")

item$make_ui(container=gwindow("itemList test"))

## End(Not run)
```

ItemListEditor	<i>trait for editor for itemList...</i>
----------------	---

Description

trait for editor for itemList

LabelEditor	<i>Trait for a label...</i>
-------------	-----------------------------

Description

Trait for a label

labelItem	<i>Simple label item...</i>
-----------	-----------------------------

Description

Simple label item

Usage

```
labelItem(value="label", name, label, help="", tooltip="", attr, model,
          editor, ...)
```

Arguments

value	Default value for the label
name	Required name for object. Names should be unique within a group of items
label	Same as value. Here for consistency, but needn't be used
help	Optional help string
tooltip	Optional tooltip to display
attr	A list of attributes to pass to widget on construction. Eg. attr=list(size=c(100,200))
model	Optional model. Useful if one wishes to use same model for multiple views
editor	Specification of editor (a view) to override default
...	Passed to parent proto object during call to proto

Details

Useful to adding text to a dialog. Has no interactivity.

Value

A proto object. Call obj\$show_help() for its methods and properties

loadingAnimation	<i>A window to show a loading animation...</i>
------------------	--

Description

A window to show a loading animation

Usage

```
loadingAnimation(message="<b>Loading...</b>")
```

Arguments

message A message to display along with graphic while loading. PANGO markup is okay.

Value

An item group instance with a close method to call to dismiss window

Examples

```
## we call, something happens, then we close
## Not run:
w <- loadingAnimation()
## ... something long, like dlg$make_gui() ...
w$close()

## End(Not run)
```

merge.list	<i>merge two lists, possibly overwriting...</i>
------------	---

Description

merge two lists, possibly overwriting

Usage

```
merge.list(x, y, overwrite=TRUE)
```

Arguments

x a list

y a list. Named values of y are assigned to x and then x is returned.

overwrite If TRUE named values in y clobber similarly named values in x

Value

Returns a list

Model	<i>Trait for a model object.</i>
-------	----------------------------------

Description

Trait for a model object.

Details

Model objects consist of properties and the methods that manipulate them Models are initialized by `init` so that getter and setter pairs are made When setter functions are called, the model's observers are notified

numericItem	<i>Item for numbers...</i>
-------------	----------------------------

Description

Item for numbers

Usage

```
numericItem(value=numeric(0), name, label=name, help="", tooltip="",
  eval_first=FALSE, attr, model, editor, ...)
```

Arguments

<code>value</code>	Default value for the model
<code>name</code>	Required name for object. Names should be unique within a group of items
<code>label</code>	Optional label, default value is the name
<code>help</code>	Optional help string
<code>tooltip</code>	Optional tooltip to display
<code>eval_first</code>	Should value be run through eval/parse before coercion.
<code>attr</code>	A list of attributes to pass to widget on construction. Eg. <code>attr=list(size=c(100,200))</code>
<code>model</code>	Optional model. Useful if one wishes to use same model for multiple views
<code>editor</code>	Specification of editor (a view) to override default
<code>...</code>	Passed to parent proto object during call to proto

Value

A proto object. Call `obj$show_help()` to view its methods and properties.

See Also[Item](#)**Examples**

```
## basic use
a <- numericItem(0, name="x")
a$set_x(10)
a$get_x()
## eval can be instructed
a <- numericItem(0, name="x", eval_first=TRUE)
a$set_x("1:5")
a$get_x()
a$to_R()
```

ObjectWithValuesEditor*Base trait for editor where there are underlying values to choose from...*

Description

Base trait for editor where there are underlying values to choose from

Details

editor has regular or compact style

param	<i>specify a method paramter..</i>
-------	------------------------------------

Description

specify a method paramter

Usage

```
param(value, ...)
```

Arguments

value	name of parameter
...	Values pasted in return a string

Details

internal function for writing proto docs.

RangeEditor	<i>Trait for making a range editor (slider, spinbox)...</i>
-------------	---

Description

Trait for making a range editor (slider, spinbox)

rangeItem	<i>A range selection item...</i>
-----------	----------------------------------

Description

A range selection item

Usage

```
rangeItem(value="", from=0, to=10, by=1, name, label=name, help="",
          tooltip="", attr, model, editor, ...)
```

Arguments

value	Default data frame for the model
from	Starting value of range
to	Ending value of range
by	Step size to step through range. If an integer, a spinbutton is also displayed
name	Required name for object. Names should be unique within a group of items
label	Optional label, default value is the name
help	Optional help string
tooltip	Optional tooltip to display
attr	A list of attributes to pass to widget on construction. Eg. attr=list(size=c(100,200))
model	Optional model. Useful if one wishes to use same model for multiple views
editor	Specification of editor (a view) to override default
...	Passed to Item trait

Details

Editor is a slider (with spinbutton when by value is an integer).

Value

A proto object. Call obj\$show_help() to view its methods and properties.

Examples

```
i <- rangeItem(value=5, from=0, to=10, by=1, name="rng")
i$get_rng()
i$set_rng(10)
i$get_rng()
```

returns	<i>Document return value...</i>
---------	---------------------------------

Description

Document return value

Usage

```
returns(...)
```

Arguments

... Values pasted in return a string

Details

internal function for writing proto docs.

SeparatorEditor	<i>Trait for making a visual separator...</i>
-----------------	---

Description

Trait for making a visual separator

separatorItem	<i>Visual separator item...</i>
---------------	---------------------------------

Description

Visual separator item

Usage

```
separatorItem(name=".separator", attr=list(horizontal = TRUE), ...)
```

Arguments

name	name of widget
attr	passed to widget constructor. Use list(horizontal=FALSE) to get vertical
...	ignored

Details

Creates a horizontal (or vertical if done through "attr") line to separate GUI elements.

Value

A proto object. Call obj\$show_help() to view its methods and properties.

stringItem	<i>A string item...</i>
------------	-------------------------

Description

A string item

Usage

```
stringItem(value="", regex, name, label=name, help="", tooltip="",
  eval_first=FALSE, attr, model, editor, ...)
```

Arguments

value	Default value for the model
regex	If non NULL specifies a regular expression for validation.
name	Required name for object. Names should be unique within a group of items
label	Optional label, default value is the name
help	Optional help string
tooltip	Optional tooltip to display
eval_first	Should value be run through eval/parse before coercion.
attr	A list of attributes to pass to widget on construction. Eg. attr=list(size=c(100,200))
model	Optional model. Useful if one wishes to use same model for multiple views
editor	Specification of editor (a view) to override default
...	Passed to parent proto object during call to proto

Value

A proto object. Call obj\$show_help() to view its methods and properties.

See Also

[Item](#)

Examples

```
## basic usage
a <- stringItem("ac", name="x")
a$get_x()
a$set_x("abc213")
a$get_x()
## eval first
a <- stringItem("ac", name="x", eval_first=TRUE)
a$set_x("2 + 2")
a$get_x()
a$to_R()
```

TableEditor

Trait for displaying a table of information...

Description

Trait for displaying a table of information

Details

No selection, just display. For selection use ChoiceItem

tableItem	<i>List editor – list <-> tree, must have special structure to list?</i>
-----------	--

Description

List editor – list <-> tree, must have special structure to list? XXX This needs writing An item to display a table of data (given as a matrix or data.frame)

Usage

```
tableItem(value=data.frame(V1 = "", V2 = ""), name, label=name,
          help="", tooltip="", attr=list(expand = TRUE), model, editor, ...)
```

Arguments

value	Default value of data frame
name	Required name for object. Names should be unique within a group of items
label	ignored
help	Optional help string
tooltip	Optional tooltip to display
attr	A list of attributes to pass to widget on construction. Eg. attr=list(size=c(100,200)).
model	ignored
editor	ignored
...	Passed to parent proto object during call to proto

Value

A proto object. Call obj\$show_help() to view its methods and properties.

See Also

[Item](#)

Examples

```
## to change data frame
i <- tableItem(mtcars, name="a")
i$set_a(mtcars[1:3, 1:3])
```

trueFalseItem	<i>Item for Boolean values...</i>
---------------	-----------------------------------

Description

Item for Boolean values

Usage

```
trueFalseItem(value=TRUE, name, label=name, help="", tooltip="", attr,  
              model, editor, ...)
```

Arguments

value	Default value for the model
name	Required name for object. Names should be unique within a group of items
label	Optional label, default value is the name
help	Optional help string
tooltip	Optional tooltip to display
attr	A list of attributes to pass to widget on construction. Eg. attr=list(size=c(100,200))
model	Optional model. Useful if one wishes to use same model for multiple views
editor	Specification of editor (a view) to override default
...	Passed to parent proto object during call to proto. The value editor_style="compact" will pass this information to the editor causing it to render as a checkbox.

Value

A proto object. Call obj\$show_help() to view its methods and properties.

See Also

[Item](#)

Examples

```
## basic usage  
a <- trueFalseItem(TRUE, name="x")  
a$get_x()  
a$set_x(FALSE)  
a$get_x()
```

ul	<i>write ...</i>
----	----------------------------

Description

write

Usage

ul(values)

Arguments

values	values to form items
--------	----------------------

Details

internal function for writing proto docs.

Value

returns a string

variableSelectorItem	<i>Item to select a variable (or variables) from a selected data frame...</i>
----------------------	---

Description

Item to select a variable (or variables) from a selected data frame

Usage

```
variableSelectorItem(value=NA, multiple=FALSE, dataframeItem, name,
  label=name, help="", tooltip="", attr, model, editor, ...)
```

Arguments

value	Default data frame for the model, defaults to .GlobalEnv
multiple	Allow multiple selection?
dataframeItem	A required dataframeItem instance. This need not be in same display, or even displayed
name	Required name for object. Names should be unique within a group of items
label	Optional label, default value is the name
help	Optional help string

<code>tooltip</code>	Optional tooltip to display
<code>attr</code>	A list of attributes to pass to widget on construction. Eg. <code>attr=list(size=c(100,200))</code> This widget uses a <code>gtable</code> instance and specifying the size is suggested
<code>model</code>	Optional model. Useful if one wishes to use same model for multiple views
<code>editor</code>	Specification of editor (a view) to override default
<code>...</code>	Passed to parent proto object during call to proto

Details

Needs to have a `dataframeItem` specified to be useful.

Value

A proto object. Call `obj$show_help()` to view its methods and properties.

See Also

[dataframeItem](#), [Item](#)

Examples

```
df <- data.frame(a=1:3, b= letters[1:3], c=rnorm(3)) # make a data frame
dfI <- dataframeItem(value="df", name="dfI")
dlg <- aDialog(items=list(
  dfI,
  ## a bit awkward -- can't define dfI in list of items
  variable=variableSelectorItem(dataframeItem=dfI)
)
## Not run: dlg$make_gui()
```

View

Trait for View objects.

Description

Trait for View objects.

Details

A view "displays" the values in an associated model. The association is through a controller the view does not know the controller or the model Views are initialized through the `make_ui` method.

wrap_in_tag	<i>write values in paired tags with optional class...</i>
-------------	---

Description

write values in paired tags with optional class

Usage

```
wrap_in_tag(tag, ..., class="")
```

Arguments

tag	tag to wrap in, eg. "ul"
...	values to be wrapped. Pasted together
class	optional call to add to tag

Details

internal function for writing proto docs

Value

a string

Index

aContainer, 4, 12, 16
aContext, 5
aController, 5
Adapter, 6
aDialog, 3, 6
aFrame, 8
aGroup, 9
aModel, 10
anExpandGroup, 11
anItemGroup, 6, 12
aNotebook, 13
aNotebookPage, 14
aPanedGroup, 15
aTableLayout, 16
aView, 17

BaseTrait, 17
BooleanEditor, 17
ButtonEditor, 18
buttonItem, 18

choiceItem, 19
Container, 4, 5, 9–11, 14–16, 21
Controller, 21

dataframeItem, 21, 47
DateEditor, 22
dateItem, 22
desc, 23
dfEditItem, 24
Dialog, 24
dialogMaker, 25

Editor, 26
EntryEditor, 26
expressionItem, 26

FileBrowseEditor, 27
fileItem, 27
formulaItem, 28

get_with_default, 29
gmenu, 7
GraphEditor, 29
graphicDeviceItem, 29
gtoolbar, 7

ImageEditor, 30
imageItem, 31
integerItem, 27, 32
Item, 19, 20, 23, 30, 31, 33, 39, 43–45, 47
ItemGroup, 33
itemList, 34
ItemListEditor, 35

LabelEditor, 35
labelItem, 36
loadingAnimation, 36

merge.list, 37
Model, 38

numericItem, 27, 32, 38

ObjectWithValuesEditor, 39

param, 39

RangeEditor, 40
rangeItem, 40
returns, 41

SeparatorEditor, 41
separatorItem, 42
strftime, 23
stringItem, 27, 42

TableEditor, 43
tableItem, 44
traitR-package, 3
trueFalseItem, 45

ul, 46

`variableSelectorItem`, [46](#)

`View`, [47](#)

`wrap_in_tag`, [48](#)