Package 'periscope'

September 21, 2020

Type Package

Title Enterprise Streamlined 'Shiny' Application Framework

Version 0.5.2

Description An enterprise-targeted scalable and UI-standardized 'shiny' framework including a variety of developer convenience functions with the goal of both streamlining robust application development while assisting with creating a consistent user experience regardless of application or developer.

URL https://github.com/cb4ds/periscope, http://periscopeapps.org:3838, https://www.canvasxpress.org

BugReports https://github.com/cb4ds/periscope/issues

Repository CRAN

License GPL-3

Encoding UTF-8

Language en-US

LazyData true

Depends R (>= 3.4)

Imports shiny (>= 1.1), shinydashboard (>= 0.5), shinydashboardPlus (>= 0.5), shinyBS (>= 0.61), lubridate (>= 1.6), DT (>= 0.2), writexl (>= 1.3), ggplot2 (>= 2.2), methods, utils

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, testthat, openxlsx (>= 3.0)

VignetteBuilder knitr

NeedsCompilation no

Author Constance Brett [aut, cre],
Isaac Neuhaus [aut] (canvasXpress JavaScript Library Maintainer),
Ger Inberg [ctb],
Bristol-Meyers Squibb (BMS) [cph]

Maintainer Constance Brett <connie@aggregate-genius.com>

Date/Publication 2020-09-21 17:50:02 UTC

R topics documented:

`add_left_sidebar` *Add the left sidebar to an existing application.*

Description

Add the left sidebar to an existing application.

Usage

```
add_left_sidebar(location)
```

Arguments

location	path of the existing application.

---

add_reset_button *Add the reset button to an existing application.*

Description

Add the reset button to an existing application.

Usage

```
add_reset_button(location)
```

Arguments

location	path of the existing application.

---

add_right_sidebar *Add the right sidebar to an existing application.*

Description

Add the right sidebar to an existing application.

Usage

```
add_right_sidebar(location)
```

Arguments

location	path of the existing application.

add_ui_body *Add UI Elements to the Body area*

Description

This function registers UI elements to the body of the application (the right side). Items are added in the order given.

Usage

```
add_ui_body(elementlist = NULL, append = FALSE)
```

Arguments

elementlist	list of UI elements to add to the body
append	whether to append the elementlist to the currently registered elements or replace the currently registered elements completely

Shiny Usage

Call this function after creating elements in program/ui_body.R to register them to the application framework and show them on the body area of the dashboard application

See Also

add_ui_sidebar_basic

add_ui_sidebar_advanced

Examples

```
require(shiny)

body1 <- htmlOutput("example1")
body2 <- actionButton("exButton", label = "Example")

add_ui_body(list(body1, body2))
```

add_ui_sidebar_advanced *Add UI Elements to the Sidebar (Advanced Tab)*

Description

This function registers UI elements to the secondary (rear-most) tab on the dashboard sidebar. The default name of the tab is **Advanced** but can be renamed using the tabname argument.

Usage

```
add_ui_sidebar_advanced(
  elementlist = NULL,
  append = FALSE,
  tabname = "Advanced"
)
```

Arguments

`elementlist`	list of UI elements to add to the sidebar tab
`append`	whether to append the `elementlist` to the currently registered elements or replace the currently registered elements completely
`tabname`	change the label on the UI tab (default = "Advanced")

Shiny Usage

Call this function after creating elements in `program/ui_sidebar.R` to register them to the application framework and show them on the Advanced tab in the dashboard sidebar

See Also

add_ui_sidebar_basic

add_ui_body

Examples

```
require(shiny)

s1 <- selectInput("sample1", "A Select", c("A", "B", "C"))
s2 <- radioButtons("sample2", NULL, c("A", "B", "C"))

add_ui_sidebar_advanced(list(s1, s2), append = FALSE)
```

`add_ui_sidebar_basic` *Add UI Elements to the Sidebar (Basic Tab)*

Description

This function registers UI elements to the primary (front-most) tab on the dashboard sidebar. The default name of the tab is **Basic** but can be renamed using the tabname argument. This tab will be active on the sidebar when the user first opens the shiny application.

Usage

```
add_ui_sidebar_basic(elementlist = NULL, append = FALSE, tabname = "Basic")
```

Arguments

`elementlist`	list of UI elements to add to the sidebar tab
`append`	whether to append the `elementlist` to currently registered elements or replace the currently registered elements.
`tabname`	change the label on the UI tab (default = "Basic")

Shiny Usage

Call this function after creating elements in `ui_sidebar.R` to register them to the application framework and show them on the Basic tab in the dashboard sidebar

See Also

add_ui_sidebar_advanced

add_ui_body

Examples

```
require(shiny)

s1 <- selectInput("sample1", "A Select", c("A", "B", "C"))
s2 <- radioButtons("sample2", NULL, c("A", "B", "C"))

add_ui_sidebar_basic(list(s1, s2), append = FALSE)
```

add_ui_sidebar_right *Add UI Elements to the Right Sidebar*

Description

This function registers UI elements at the right dashboard sidebar. The UI element should be of type rightSidebarTabContent.

Usage

```
add_ui_sidebar_right(elementlist = NULL, append = FALSE)
```

Arguments

elementlist	list of UI elements to add to the sidebar tab
append	whether to append the elementlist to the currently registered elements or replace the currently registered elements completely

Shiny Usage

Call this function after creating elements in program/ui_sidebar_right.R to register them to the application framework and show them on the right dashboard sidebar

See Also

add_ui_sidebar_basic

add_ui_body

rightSidebarTabContent

Examples

```
require(shiny)
require(shinydashboardPlus)

s1 <- rightSidebarTabContent(id = 1, icon = "desktop", title = "Tab 1 - Plots", active = TRUE,
                             div(helpText(align = "center", "Sample UI Text"),
                                 selectInput("sample1", "A Select", c("A", "B", "C")) ))

add_ui_sidebar_right(list(s1), append = FALSE)
```

bootstrapping *Bootstrapping the logging package.*

Description

basicConfig and logReset provide a way to put the logging package in a know initial state.

Usage

```
basicConfig(level = 20)

logReset()
```

Arguments

level	The logging level of the root logger. Defaults to INFO. Please do notice that this has no effect on the handling level of the handler that basicConfig attaches to the root logger.

Details

basicConfig creates the root logger, attaches a console handler(by *basic.stdout* name) to it and sets the level of the handler to level. You must not call basicConfig to for logger to work any more: then root logger is created it gets initialized by default the same way as basicConfig does. If you need clear logger to fill with you own handlers use logReset to remove all default handlers.

logReset reinitializes the whole logging system as if the package had just been loaded except it also removes all default handlers. Typically, you would want to call basicConfig immediately after a call to logReset.

create_new_application *Create a new templated framework application*

Description

Creates ready-to-use templated application files using the periscope framework. The application can be created either empty (default) or with a sample/documented example application.

A running instance of the exact sample application that will be created is hosted here if you would like to see the sample application before creating your own copy.

Usage

```
create_new_application(
  name,
  location,
  sampleapp = FALSE,
  resetbutton = TRUE,
  rightsidebar = FALSE,
  leftsidebar = TRUE,
  style = list(skin = "blue")
)
```

Arguments

name	name for the new application and directory
location	base path for creation of name
sampleapp	whether to create a sample shiny application
resetbutton	whether the reset button should be added on the Advanced (left) sidebar.
rightsidebar	parameter to set the right sidebar. It can be TRUE/FALSE or a character containing the name of a shiny::icon().
leftsidebar	whether the left sidebar should be enabled.
style	list containing application styling properties. By default the skin is blue.

Name

The name directory must not exist in location. If the code detects that this directory exists it will abort the creation process with a warning and will not create an application template.

Use only filesystem-compatible characters in the name (ideally w/o spaces)

Directory Structure

```
name
 -- www (supporting shiny files)
 -- program (user application)
 -- -- data (user application data)
 -- -- fxn  (user application function)
 -- log (log files)
```

File Information

All user application creation and modifications will be done in the **program** directory. The names & locations of the framework-provided .R files should not be changed or the framework will fail to work as expected.

***name/program*/ui_body.R** :
Create body UI elements in this file and register them with the framework using a call to add_ui_body

***name/program*/ui_sidebar.R** :

Create sidebar UI elements in this file and register them with the framework using a call to add_ui_sidebar_basic or add_ui_sidebar_advanced

***name/program*/ui_sidebar_right.R** :
Create right sidebar UI elements in this file and register them with the framework using a call to add_ui_sidebar_right

name/program/data directory :
Use this location for data files. There is a **.gitignore** file included in this directory to prevent accidental versioning of data

***name/program*/global.R** :
Use this location for code that would have previously resided in global.R and for setting application parameters using set_app_parameters. Anything placed in this file will be accessible across all user sessions as well as within the UI context.

***name/program*/server_global.R** :
Use this location for code that would have previously resided in server.R above (i.e. outside of) the call to `shinyServer(...)`. Anything placed in this file will be accessible across all user sessions.

***name/program*/server_local.R** :
Use this location for code that would have previously resided in server.R inside of the call to `shinyServer(...)`. Anything placed in this file will be accessible only within a single user session.

Do not modify the following files:

```
name\global.R
name\server.R
name\ui.R
name\www\img\loader.gif
name\www\img\tooltip.png
```

Right Sidebar

```
value
FALSE   --- no sidebar
TRUE    --- sidebar with default icon ('gears').
"table" --- sidebar with table icon. The character string should be a valid "font-awesome" icon.
```

See Also

shiny:icon()

shinydashboard:dashboardPage()

Examples

```
# sample app named 'mytestapp' created in a temp dir
create_new_application(name = 'mytestapp', location = tempdir(), sampleapp = TRUE)

# sample app named 'mytestapp' with a right sidebar using a custom icon created in a temp dir
create_new_application(name = 'mytestapp', location = tempdir(), sampleapp = TRUE,
rightsidebar = "table")

# blank app named 'myblankapp' created in a temp dir
create_new_application(name = 'myblankapp', location = tempdir())
# blank app named 'myblankapp' with a green skin created in a temp dir
create_new_application(name = 'myblankapp', location = tempdir(), style = list(skin = "green"))
# blank app named 'myblankapp' without a left sidebar created in a temp dir
create_new_application(name = 'myblankapp', location = tempdir(), leftsidebar = FALSE)
```

---

downloadablePlot *downloadablePlot Module*

Description

Server-side function for the downloadablePlotUI. This is a custom plot output paired with a linked downloadFile button.

Usage

```
downloadablePlot(
  input,
  output,
  session,
  logger,
  filenameroot,
  aspectratio = 1,
  downloadfxns = list(),
  visibleplot
)
```

Arguments

input	provided by shiny::callModule
output	provided by shiny::callModule
session	provided by shiny::callModule
logger	logger to use
filenameroot	the base text used for user-downloaded file - can be either a character string or a reactive expression returning a character string

aspectratio	the downloaded chart image width:height ratio (ex: 1 = square, 1.3 = 4:3, 0.5 = 1:2). Where not applicable for a download type it is ignored (e.g. data, html downloads)
downloadfxns	a **named** list of functions providing download images or data tables as return values. The names for the list should be the same names that were used when the plot UI was created.
visibleplot	function or reactive expression providing the plot to display as a return value. This function should require no input parameters.

Notes

When there are no values to download in any of the linked downloadfxns the button will be hidden as there is nothing to download.

Shiny Usage

This function is not called directly by consumers - it is accessed in server.R using the same id provided in `downloadablePlotUI`:

```
callModule(downloadablePlot, id, logger, filenameroot, downloadfxns, visibleplot)
```

See Also

downloadablePlotUI

callModule

Examples

```
# Inside server_local.R

# callModule(downloadablePlot,
#            "object_id1",
#            logger = ss_userAction.Log,
#            filenameroot = "mydownload1",
#            aspectratio = 1.33,
#            downloadfxns = list(png = myplotfxn, tsv = mydatafxn),
#            visibleplot = myplotfxn)
```

---

downloadablePlotUI *downloadablePlot UI*

Description

Creates a custom plot output that is paired with a linked downloadFile button. This module is compatible with ggplot2, grob and lattice produced graphics.

Usage

```
downloadablePlotUI(
  id,
  downloadtypes = c("png"),
  download_hovertext = NULL,
  width = "100%",
  height = "400px",
  btn_halign = "right",
  btn_valign = "bottom",
  btn_overlap = TRUE,
  clickOpts = NULL,
  hoverOpts = NULL,
  brushOpts = NULL
)
```

Arguments

id	character id for the object
downloadtypes	vector of values for download types
download_hovertext	download button tooltip hover text
width	plot width (any valid css size value)
height	plot height (any valid css size value)
btn_halign	horizontal position of the download button ("left", "center", "right")
btn_valign	vertical position of the download button ("top", "bottom")
btn_overlap	whether the button should appear on top of the bottom of the plot area to save on vertical space *(there is often a blank area where a button can be overlayed instead of utilizing an entire horizontal row for the button below the plot area)*
clickOpts	NULL or an object created by the clickOpts function
hoverOpts	NULL or an object created by the hoverOpts function
brushOpts	NULL or an object created by the brushOpts function

Example

```
downloadablePlotUI("myplotID",c("png","csv"),"Download Plot or Data","300px")
```

Notes

When there is nothing to download in any of the linked downloadfxns the button will be hidden as there is nothing to download. The linked downloadfxns are set in the paired callModule (see the **Shiny Usage** section)

This module is NOT compatible with the built-in (base) graphics *(such as basic plot, etc.)* because they cannot be saved into an object and are directly output by the system at the time of creation.

Shiny Usage

Call this function at the place in ui.R where the plot should be placed.

Paired with a call to `shiny::callModule(downloadablePlot,id,...)` in server.R

See Also

downloadablePlot

downloadFileButton

clickOpts

hoverOpts

brushOpts

Examples

```
# Inside ui_body.R or ui_sidebar.R
downloadablePlotUI("object_id1",
                   downloadtypes = c("png", "csv"),
                   download_hovertext = "Download the plot and data here!",
                   height = "500px",
                   btn_halign = "left")
```

---

downloadableTable *downloadableTable Module*

Description

Server-side function for the downloadableTableUI. This is a custom high-functionality table paired with a linked downloadFile button.

Usage

```
downloadableTable(
  input,
  output,
  session,
  logger,
  filenameroot,
  downloaddatafxns = list(),
  tabledata,
  rownames = TRUE,
  caption = NULL,
  selection = NULL
)
```

Arguments

`input`	provided by `shiny::callModule`
`output`	provided by `shiny::callModule`
`session`	provided by `shiny::callModule`
`logger`	logger to use
`filenameroot`	the base text used for user-downloaded file - can be either a character string or a reactive expression returning a character string
`downloaddatafxns`	a **named** list of functions providing the data as return values. The names for the list should be the same names that were used when the table UI was created.
`tabledata`	function or reactive expression providing the table display data as a return value. This function should require no input parameters.
`rownames`	whether or not to show the rownames in the table
`caption`	table caption
`selection`	function or reactive expression providing the row_ids of the rows that should be selected.

Value

Reactive expression containing the currently selected rows in the display table

Notes

When there are no rows to download in any of the linked downloaddatafxns the button will be hidden as there is nothing to download.

Shiny Usage

This function is not called directly by consumers - it is accessed in server.R using the same id provided in `downloadableTableUI`:

```
callModule(downloadableTable, id, logger, filenameroot, downloaddatafxns, tabledata,
rownames, caption, selection)
```

Note: callModule returns the reactive expression containing the currently selected rows in the display table.

See Also

downloadableTableUI

callModule

Examples

```
# Inside server_local.R

# selectedrows <- callModule(downloadableTable,
#                            "object_id1",
#                            logger = ss_userAction.Log,
#                            filenameroot = "mydownload1",
#                           downloaddatafxns = list(csv = mydatafxn1, tsv = mydatafxn2),
#                            tabledata = mydatafxn3,
#                            rownames = FALSE,
#                            caption = "This is a great table!  By: Me",
#                            selection = mydataRowIds)

# selectedrows is the reactive return value, captured for later use
```

`downloadableTableUI` *downloadableTable UI*

Description

Creates a custom high-functionality table paired with a linked downloadFile button. The table has search and highlight functionality, infinite scrolling, sorting by columns and returns a reactive dataset of selected items.

Usage

```
downloadableTableUI(
  id,
  downloadtypes = c("csv"),
  hovertext = NULL,
  contentHeight = "200px",
  singleSelect = FALSE
)
```

Arguments

`id`	character id for the object
`downloadtypes`	vector of values for data download types
`hovertext`	download button tooltip hover text
`contentHeight`	viewable height of the table (any valid css size value)
`singleSelect`	whether the table should only allow a single row to be selected at a time (FALSE by default allows multi-select).

Table Features

- Consistent styling of the table
- downloadFile module button functionality built-in to the table
- Ability to show different data from the download data
- Table is automatically fit to the window size with infinite y-scrolling
- Table search functionality including highlighting built-in
- Multi-select built in, including reactive feedback on which table items are selected

Example

```
downloadableTableUI("mytableID",c("csv","tsv"),"Click Here","300px")
```

Notes

When there are no rows to download in any of the linked downloaddatafxns the button will be hidden as there is nothing to download. The linked downloaddatafxns are set in the paired callModule (see the **Shiny Usage** section)

Shiny Usage

Call this function at the place in ui.R where the table should be placed.

Paired with a call to `shiny::callModule(downloadableTable,id,...)` in server.R

See Also

downloadableTable

downloadFileButton

Examples

```
# Inside ui_body.R or ui_sidebar.R
downloadableTableUI("object_id1",
                    downloadtypes = c("csv", "tsv"),
                    hovertext = "Download the data here!",
                    contentHeight = "300px",
                    singleSelect = FALSE)
```

`downloadFile` *downloadFile Module*

Description

Server-side function for the downloadFileButton. This is a custom high-functionality button for file downloads supporting single or multiple download types. The server function is used to provide the data for download.

Usage

```
downloadFile(
  input,
  output,
  session,
  logger,
  filenameroot,
  datafxns = list(),
  aspectratio = 1
)
```

Arguments

`input`	provided by `shiny::callModule`
`output`	provided by `shiny::callModule`
`session`	provided by `shiny::callModule`
`logger`	logger to use
`filenameroot`	the base text used for user-downloaded file - can be either a character string or a reactive expression that returns a character string
`datafxns`	a **named** list of functions providing the data as return values. The names for the list should be the same names that were used when the button UI was created.
`aspectratio`	the downloaded chart image width:height ratio (ex: 1 = square, 1.3 = 4:3, 0.5 = 1:2). Where not applicable for a download type it is ignored (e.g. data downloads).

Shiny Usage

This function is not called directly by consumers - it is accessed in server.R using the same id provided in `downloadFileButton`:

```
callModule(downloadFile, id, logger, filenameroot, datafxns)
```

See Also

downloadFileButton

downloadFile_ValidateTypes

downloadFile_AvailableTypes

callModule

Examples

```
# Inside server_local.R

#single download type
# callModule(downloadFile,
#            "object_id1",
#            logger = ss_userAction.Log,
#            filenameroot = "mydownload1",
#            datafxns = list(csv = mydatafxn1),
#            aspectratio = 1)

#multiple download types
# callModule(downloadFile,
#            "object_id2",
#            logger = ss_userAction.Log,
#            filenameroot = "mytype2",
#            datafxns = list(csv = mydatafxn1, xlsx = mydatafxn2),
#            aspectratio = 1)
```

---

downloadFileButton *downloadFileButton UI*

Description

Creates a custom high-functionality button for file downloads with two states - single download type or multiple-download types. The button image and pop-up menu (if needed) are set accordingly. A tooltip can also be set for the button.

Usage

```
downloadFileButton(id, downloadtypes = c("csv"), hovertext = NULL)
```

Arguments

id	character id for the object
downloadtypes	vector of values for data download types
hovertext	tooltip hover text

Button Features

- Consistent styling of the button, including a hover tooltip
- Single or multiple types of downloads
- Ability to download different data for each type of download

Example

`downloadFileUI("mybuttonID1",c("csv","tsv"),"Click Here")` `downloadFileUI("mybuttonID2","csv","Click to download")`

Shiny Usage

Call this function at the place in ui.R where the button should be placed.

It is paired with a call to `shiny::callModule(downloadFile,id,...)` in server.R

See Also

downloadFile

downloadFile_ValidateTypes

downloadFile_AvailableTypes

Examples

```
# Inside ui_body.R or ui_sidebar.R

#single download type
downloadFileButton("object_id1",
                   downloadtypes = c("csv"),
                   hovertext = "Button 1 Tooltip")

#multiple download types
downloadFileButton("object_id2",
                   downloadtypes = c("csv", "tsv"),
                   hovertext = "Button 2 Tooltip")
```

`downloadFile_AvailableTypes` *downloadFile Helper*

Description

Returns a list of all supported types

Usage

```
downloadFile_AvailableTypes()
```

Value

a vector of all supported types

See Also

downloadFileButton

downloadFile

`downloadFile_ValidateTypes` *downloadFile Helper*

Description

Checks a given list of file types and warns if an invalid type is included

Usage

```
downloadFile_ValidateTypes(types)
```

Arguments

`types`	list of types to test

Value

the list input given in types

Example

```
downloadFile_ValidateTypes(c("csv","tsv"))
```

See Also

downloadFileButton

downloadFile

getHandler *Retrieves a handler from a logger.*

Description

Handlers are not uniquely identified by their name. Only within the logger to which they are attached is their name unique. This function is here to allow you grab a handler from a logger so you can examine and alter it.

Typical use of this function is in `setLevel(newLevel,getHandler(...))`.

Usage

```
getHandler(handler, logger = "")
```

Arguments

`handler`	The name of the handler, or its action.
`logger`	Optional: the name of the logger. Defaults to the root logger.

Value

The retrieved handler object. It returns NULL if handler is not registered.

getLogger *Set defaults and get the named logger.*

Description

Make sure a logger with a specific name exists and return it as a *Logger* S4 object. if not yet present, the logger will be created and given the values specified in the ... arguments.

Usage

```
getLogger(name = "", ...)
```

Arguments

`name`	The name of the logger
`...`	Any properties you may want to set in the newly created logger. These have no effect if the logger is already present.

Value

The logger retrieved or registered.

get_url_parameters *Get URL Parameters*

Description

This function returns any url parameters passed to the application as a named list. Keep in mind url parameters are always user-session scoped

Usage

```
get_url_parameters(session)
```

Arguments

session	shiny session object

Value

named list of url parameters and values. List may be empty if no URL parameters were passed when the application instance was launched.

handlers-management *Add a handler to or remove one from a logger.*

Description

Use this function to maintain the list of handlers attached to a logger.

addHandler and removeHandler are also offered as methods of the *Logger* S4 class.

Usage

```
addHandler(handler, ..., logger = "")

removeHandler(handler, logger = "")
```

Arguments

handler	The name of the handler, or its action
...	Extra parameters, to be stored in the handler list ... may contain extra parameters that will be passed to the handler action. Some elements in the ... will be interpreted here.
logger	the name of the logger to which to attach the new handler, defaults to the root logger.

Details

Handlers are implemented as environments. Within a logger a handler is identified by its *name* and all handlers define at least the three variables:

level all records at level lower than this are skipped.

formatter a function getting a record and returning a string

`action(msg, handler)` a function accepting two parameters: a formatted log record and the handler itself. making the handler a parameter of the action allows us to have reusable action functions.

Being an environment, a handler may define as many variables as you think you need. keep in mind the handler is passed to the action function, which can check for existence and can use all variables that the handler defines.

`inbuilt-actions` *Predefined(sample) handler actions*

Description

When you define a handler, you specify its name and the associated action. A few predefined actions described below are provided.

Usage

```
writeToConsole(msg, handler, ...)

writeToFile(msg, handler, ...)
```

Arguments

`msg`	A formatted message to handle.
`handler`	The handler environment containing its options. You can register the same action to handlers with different properties.
`...`	parameters provided by logger system to interact with the action.

Details

A handler action is a function that accepts a formatted message and handler configuration.

Messages passed are filtered already regarding loglevel.

...parameters are used by logging system to interact with the action. ...can contain *dry* key to inform action that it meant to initialize itself. In the case action should return TRUE if initialization succeeded.

If it's not a dry run ...contain the whole preformatted *logging.record*. A *logging.record* is a named list and has following structure:

msg contains the real formatted message

level message level as numeric

levelname message level name

logger name of the logger that generated it

timestamp formatted message timestamp

`writeToConsole` detects if crayon package is available and uses it to color messages. The coloring can be switched off by means of configuring the handler with *color_output* option set to FALSE.

`writeToFile` action expects file path to write to under *file* key in handler options.

Examples

```
## define your own function and register it with a handler.
## author is planning a sentry client function.  please send
## any interesting function you may have written!
```

`logging-entrypoints` *Entry points for logging actions*

Description

Generate a log record and pass it to the logging system.

Usage

```
logdebug(msg, ..., logger = "")

logfinest(msg, ..., logger = "")

logfiner(msg, ..., logger = "")

logfine(msg, ..., logger = "")

loginfo(msg, ..., logger = "")

logwarn(msg, ..., logger = "")

logerror(msg, ..., logger = "")

levellog(level, msg, ..., logger = "")
```

Arguments

msg	the textual message to be output, or the format for the ... arguments
...	if present, msg is interpreted as a format and the ... values are passed to it to form the actual message.
logger	the name of the logger to which we pass the record
level	The logging level

Details

A log record gets timestamped and will be independently formatted by each of the handlers handling it.

Leading and trailing whitespace is stripped from the final message.

---

loglevels *The logging levels, names and values*

Description

This list associates names to values and vice versa.
Names and values are the same as in the python standard logging module.

Usage

```
loglevels
```

Format

An object of class numeric of length 11.

---

periscope *Periscope Shiny Application Framework*

Description

This package supports a ui-standardized environment as well as a variety of convenience functions for shiny applications. Base reusable functionality as well as UI paradigms are included to ensure a consistent user experience regardless of application or developer.

Details

A gallery of example apps is hosted at http://periscopeapps.org

Function Overview

Create a new framework application instance:
create_new_application

Set application parameters in program/global.R:
set_app_parameters

Get any url parameters passed to the application:
get_url_parameters

Register user-created UI objects to the requisite application locations:
add_ui_sidebar_basic
add_ui_sidebar_advanced
add_ui_sidebar_right
add_ui_body

Included shiny modules with a customized UI:
downloadFileButton
downloadableTableUI
downloadablePlotUI

High-functionality standardized tooltips:
ui_tooltip

More Information

```
browseVignettes(package = 'periscope')
```

`remove_reset_button` *Remove the reset button from an existing application.*

Description

Remove the reset button from an existing application.

Usage

```
remove_reset_button(location)
```

Arguments

`location`	path of the existing application.

resetMsgComposer *Resets previously set message composer.*

Description

Resets previously set message composer.

Usage

```
resetMsgComposer(container = "")
```

Arguments

container	name of logger to reset message composer for (type: character)

setLevel *Set logging.level for the object.*

Description

Alter an existing logger or handler, setting its *logging.level* to a new value. You can access loggers by name, while you must use `getHandler` to get a handler.

Usage

```
setLevel(level, container = "")
```

Arguments

level	The new level for this object. Can be numeric or character.
container	a logger, its name or a handler. Default is root logger.

setMsgComposer *Sets message composer for logger.*

Description

Message composer is used to compose log message out of formatting string and arguments. It is function with signature `function(msg,...)`. Formatting message is passed under msg and formatting arguments are passed as `....`

Usage

```
setMsgComposer(composer_f, container = "")
```

Arguments

`composer_f`	message composer function (type: function(msg, ...))
`container`	name of logger to reset message composer for (type: character)

Details

If message composer is not set default is in use (realized with `sprintf`). If message composer is not set for sub-logger, parent's message composer will be used.

set_app_parameters *Set Application Parameters*

Description

This function sets global parameters customizing the shiny application.

Usage

```
set_app_parameters(
  title,
  titleinfo = NULL,
  loglevel = "DEBUG",
  showlog = TRUE,
  app_version = "1.0.0"
)
```

Arguments

`title`	application title text
`titleinfo`	character string, HTML value or NULL • A **character** string will be used to set a link target. This means the user will be able to click on the application title and be redirected in a new window to whatever value is given in the string. Any valid URL, File, or other script functionality that would normally be accepted in an <a href=...> tag is allowed. • An **HTML** value will be used to as the HTML content for a modal pop-up window that will appear on-top of the application when the user clicks on the application title. • Supplying **NULL** will disable the title link functionality.
`loglevel`	character string designating the log level to use for the userlog (default = 'DEBUG')
`showlog`	enable or disable the visible userlog at the bottom of the body on the application. Logging will still take place, this disables the visible functionality only.
`app_version`	character string designating the application version (default = '1.0.0').

Shiny Usage

Call this function from `program/global.R` to set the application parameters.

`ui_tooltip` *Insert a standardized tooltip*

Description

This function inserts a standardized tooltip image, label (optional), and hovertext into the application UI

Usage

```
ui_tooltip(id, label = "", text = "")
```

Arguments

`id`	character id for the tooltip object
`label`	text label to appear to the left of the tooltip image
`text`	tooltip text shown when the user hovers over the image

`updateOptions` *Changes settings of logger or handler.*

Description

Changes settings of logger or handler.

Usage

```
updateOptions(container, ...)

## S3 method for class 'character'
updateOptions(container, ...)

## S3 method for class 'environment'
updateOptions(container, ...)

## S3 method for class 'Logger'
updateOptions(container, ...)
```

Arguments

`container`	a logger, its name or a handler.
`...`	options to set for the container.

Methods (by class)

- `character`: Update options for logger identified by name.
- `environment`: Update options of logger or handler passed by reference.
- `Logger`: Update options of logger or handler passed by reference.

Index