Package 'periscope2'

November 14, 2023

Type Package

Title Enterprise Streamlined 'shiny' Application Framework Using 'bs4Dash'

Version 0.1.4

Description A framework for building enterprise, scalable and UI-standardized 'shiny' applications. It brings enhanced features such as 'bootstrap' v4 <https://getbootstrap.com/docs/4.0/getting-started/introduction/>, additional and enhanced 'shiny' modules, customizable UI features, as well as an enhanced application file organization paradigm. This update allows developers to harness the ability to build powerful applications and enriches the 'shiny' developers' experience when building and maintaining applications.

URL https://github.com/Aggregate-Genius/periscope2, http://periscopeapps.org:3838

BugReports https://github.com/Aggregate-Genius/periscope2/issues

License GPL-3

Encoding UTF-8

Language en-US

Depends R (>= 4.0)

Imports shiny (>= 1.7), bs4Dash (>= 2.3), DT, fresh, grDevices, lubridate, methods, miniUI, shinyFeedback, shinyWidgets, utils, writexl, yaml

RoxygenNote 7.2.3

Suggests assertthat, canvasXpress, ggplot2, knitr, lattice, openxlsx, rmarkdown, shinyjs, spelling, testthat, waiter

VignetteBuilder knitr

NeedsCompilation no

Author Mohammed Ali [aut, cre], Constance Brett [ctb], Aggregate Genius Inc [spn]

Maintainer Mohammed Ali <mohammed@aggregate-genius.com>

Repository CRAN

Date/Publication 2023-11-14 08:53:23 UTC

R topics documented:

`add_ui_body` *Add UI elements to dashboard body section*

Description

Builds application body with given configurations and elements. It is called within "ui_body.R". Check example application for detailed example

Usage

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

Arguments

body_elements	- List of UI elements to be displayed in application body
append	- Add elements to current body elements or remove previous body elements (default = FALSE)

Value

list of both shiny UI elements and html div tags for alert and linking app JS and CSS files

Shiny Usage

Call this function from program/ui_body.R to set body parameters

See Also

bs4Dash:bs4DashBody()

periscope2:add_ui_footer()

periscope2:add_ui_left_sidebar()

periscope2:add_ui_header()

periscope2:add_ui_right_sidebar()

periscope2:ui_tooltip()

periscope2:get_url_parameters()

Examples

```
library(shiny)
library(bs4Dash)
# Inside ui_body.R
about_box <- jumbotron(title  = "periscope2: Test Example",
               lead  = p("periscope2 is a scalable and UI-standardized 'shiny' framework
                         including a variety of developer convenience functions"),
                       status = "info",
                       href   = "https://periscopeapps.org/")
# -- Register Elements in the ORDER SHOWN in the UI
add_ui_body(list(about_box))
```

---

add_ui_footer *Add UI elements to dashboard footer section*

Description

Builds application footer with given configurations and elements. It is called within "ui_footer.R". Check example application for detailed example

Usage

```
add_ui_footer(left = NULL, right = NULL, fixed = FALSE)
```

Arguments

left	- Left side UI elements
right	- Right side UI elements
fixed	- Always show footer at page bottom regardless page scroll location (default = FALSE).

Value

list of both shiny UI elements and named footer properties

Shiny Usage

Call this function from `program/ui_footer.R` to set footer parameters

See Also

bs4Dash:bs4DashFooter()

periscope2:add_ui_left_sidebar()

periscope2:add_ui_header()

periscope2:add_ui_body()

periscope2:add_ui_right_sidebar()

periscope2:set_app_parameters()

periscope2:ui_tooltip()

periscope2:get_url_parameters()

Examples

```
library(shiny)
library(bs4Dash)

# Inside ui_footer.R
# Left text
left <- a(href   = "https://periscopeapps.org/",
          target = "_blank",
          "periscope2")
# Right text
right <- "2022"

# -- Register Elements in the ORDER SHOWN in the UI
add_ui_footer(left, right)
```

add_ui_header *Add UI elements to dashboard header section*

Description

Builds application header with given configurations and elements. It is called within "ui_header.R". Check example application for detailed example

Usage

```
add_ui_header(
  left_menu = NULL,
  right_menu = NULL,
  border = TRUE,
  compact = FALSE,
  right_sidebar_icon = shiny::icon("bars"),
  fixed = FALSE,
  left_sidebar_icon = shiny::icon("th"),
  skin = "light",
  status = "white"
)
```

Arguments

left_menu	- Left menu. bs4DropdownMenu object (or similar dropdown menu). Check ?bs4Dash::bs4DropdownMenu()
right_menu	- Right menu. bs4DropdownMenu object (or similar dropdown menu). Check ?bs4Dash::bs4DropdownMenu()
border	- Whether to separate the navbar and body by a border. TRUE by default
compact	- Whether items should be compacted. FALSE by default
right_sidebar_icon	- Right sidebar toggle icon
fixed	- Whether to fix the navbar to the top. FALSE by default
left_sidebar_icon	- Left sidebar toggle icon
skin	- Sidebar skin. "dark" or "light"
status	- Sidebar status. Check ?bs4Dash::bs4DashNavbar() for list of valid values

Value

list of both shiny UI elements and named header properties

Shiny Usage

Call this function from program/ui_header.R to set header parameters

See Also

bs4Dash:bs4DashNavbar()

periscope2:add_ui_footer()

periscope2:add_ui_left_sidebar()

periscope2:add_ui_body()

periscope2:add_ui_right_sidebar()

periscope2:ui_tooltip()

periscope2:get_url_parameters()

Examples

```
library(shiny)
library(bs4Dash)

# Inside ui_header.R
# Custom left UI menu
left_menu <- tagList(dropdownMenu(badgeStatus = "info",
                                  type        = "notifications",
                                   notificationItem(inputId = "triggerAction2",
                                                    text    = "Error!",
                                                    status  = "danger")),
                     dropdownMenu(badgeStatus = "info",
                                   type        = "tasks",
                                   taskItem(inputId = "triggerAction3",
                                            text    = "My progress",
                                            color   = "orange",
                                            value   = 10)))

# Custom right UI menu
right_menu <- dropdownMenu(badgeStatus = "danger",
                           type        = "messages",
                           messageItem(inputId = "triggerAction1",
                                       message = "message 1",
                                       from    = "Divad Nojnarg",
                                       time    = "today",
                                       color   = "lime"))
# -- Register Header Elements in the ORDER SHOWN in the UI
add_ui_header(left_menu = left_menu, right_menu = right_menu)
```

`add_ui_left_sidebar` *Add UI elements to dashboard left sidebar section*

Description

This function adds left sidebar configurations and UI elements. It is called within "ui_left_sidebar.R". Check example application for detailed example

Usage

```
add_ui_left_sidebar(
  sidebar_elements = NULL,
  sidebar_menu = NULL,
  collapsed = FALSE,
  custom_area = NULL,
  elevation = 4,
  expand_on_hover = TRUE,
  fixed = TRUE,
  minified = FALSE,
  status = "primary",
  skin = "light"
)
```

Arguments

sidebar_elements	- List of regular shiny UI elements (inputText, textArea, etc..)
sidebar_menu	- ?bs4Dash::bs4SidebarMenu() object to created a menu inside left sidebar
collapsed	- If TRUE, the sidebar will be collapsed on app start up
custom_area	- List of regular shiny UI elements but for sidebar bottom space area only. Only works if sidebar is fixed
elevation	- A number between 0 and 5, which applies a shadow to the sidebar to add a shadow effect.
expand_on_hover	- When minified is TRUE, if this property is TRUE, the sidebar opens when hovering but re-collapses as soon as the focus is lost (default = TRUE)
fixed	- Whether to see all menus at once without scrolling up and down.(default = TRUE)
minified	- Whether to slightly close the sidebar but still show item icons (default = FALSE)
status	- Determines which color menu items (if exist) will have Check ?bs4Dash::dashboardSidebar() for list of valid values
skin	- Sidebar skin. "dark" or "light" (default = "light")

Value

list of both shiny UI elements and named left sidebar properties

Shiny Usage

Call this function from program/ui_left_sidebar.R to set left sidebar parameters

See Also

bs4Dash:bs4DashSidebar()

periscope2:add_ui_footer()

periscope2:add_ui_header()

periscope2:add_ui_body()

periscope2:add_ui_right_sidebar()

periscope2:ui_tooltip()

periscope2:get_url_parameters()

Examples

```
library(shiny)
library(bs4Dash)
# Inside ui_left_sidebar.R
# sidebar menu items
sidebar_elements <- textInput("text_id", "Test", "Test Data")
sidebar_menu     <- sidebarMenu(sidebarHeader("Main Menu"),
                                menuItem("menu item 1",
                                         tabName = "item_1 page"),
                                menuItem("menu item 2",
                                         tabName = "item_2 page"))
add_ui_left_sidebar(sidebar_elements = sidebar_elements,
                    sidebar_menu     = sidebar_menu)
```

---

add_ui_right_sidebar *Add UI elements to dashboard right sidebar section*

Description

Builds application right sidebar with given configurations and elements. It is called within "ui_right_sidebar.R". Check example application for detailed example

Usage

```
add_ui_right_sidebar(
  sidebar_elements = NULL,
  sidebar_menu = NULL,
  collapsed = TRUE,
  overlay = TRUE,
  pinned = FALSE,
  skin = "light"
)
```

Arguments

sidebar_elements	- List of regular shiny UI elements (inputText, textArea, etc..)
sidebar_menu	- ?bs4Dash::controlbarMenu() object to created a menu inside right sidebar
collapsed	- If TRUE, the sidebar will be collapsed on app startup (default = TRUE)
overlay	- Whether the sidebar covers the content when expanded (default = TRUE)
pinned	- If TRUE, allows right sidebar to remain open even after a click outside (default = FALSE)
skin	- Sidebar skin. "dark" or "light" (default = "light")

Value

list of both shiny UI elements and named right sidebar properties

Shiny Usage

Call this function from program/ui_right_sidebar.R to set right sidebar parameters

See Also

bs4Dash:bs4DashControlbar()

periscope2:add_ui_footer()

periscope2:add_ui_left_sidebar()

periscope2:add_ui_header()

periscope2:add_ui_body()

periscope2:set_app_parameters()

periscope2:ui_tooltip()

periscope2:get_url_parameters()

Examples

```
library(shiny)
library(bs4Dash)

# Inside ui_right_sidebar.R
sidebar_elements <- list(div(checkboxInput("checkMe", "Example Check")))
sidebar_menu     <- controlbarMenu(id = "controlbarmenu",
                                   controlbarItem("Item 2", "Simple text"))
# -- Register Right Sidebar Elements in the ORDER SHOWN in the UI
add_ui_right_sidebar(sidebar_elements = sidebar_elements,
                     sidebar_menu     = sidebar_menu)
```

announcementConfigurationsAddin *Build Announcement Module Configuration YAML File*

Description

Call this as an addin to build valid yaml file that is needed for running announcements module. The generated file can be used in periscope2 app using set_app_parameters.

Usage

```
announcementConfigurationsAddin()
```

Details

The method can be called directly via 'R' console or via RStudio addins menu

Value

launch gadget window

See Also

periscope2:set_app_parameters()

Examples

```
if (interactive()) {
   periscope2:::announcementConfigurationsAddin()
}
```

appReset *appReset module server function*

Description

Server-side function for the appResetButton This is a custom high-functionality button for session reload. The server function is used to provide module configurations.

Usage

```
appReset(id, reset_wait = 5000, alert_location = "bodyAlert", logger)
```

Arguments

id	- Character represents the ID of the Module's UI element (the same id used in appResetButton
reset_wait	- Integer represents the period to wait before session reload in milliseconds (default = 5000)
alert_location	- Character represents div ID or selector to display module related messages (default = "bodyAlert")
logger	- logger to use

Value

nothing, function will display a warning message in the app then reload the whole application

Shiny Usage

This function is not called directly by consumers - it is accessed in server_local.R (or similar file) using the same id provided in appResetButton:

```
appReset(id = "appResetId", logger = ss_userAction.Log)
```

See Also

appResetButton

downloadFile

downloadFile_ValidateTypes

downloadFile_AvailableTypes

downloadablePlot

downloadFileButton

downloadableTable

logViewerOutput

Examples

```
if (interactive()) {
   library(shiny)
   library(periscope2)
   shinyApp(
     ui = fluidPage(fluidRow(column(12, appResetButton(id = "appResetId")))),
     server = function(input, output) {
       appReset(id = "appResetId", logger = "")
   })
}
```

appResetButton *appResetButton module UI function*

Description

Creates a toggle button to reset application session. Upon pressing on the button, its state is flipped to cancel application reload with application and console warning messages indicating that the application will be reloaded.

Usage

```
appResetButton(id)
```

Arguments

id	character id for the object

Details

User can either resume reloading application session or cancel reloading process which will also generate application and console messages to indicate reloading status and result.

Value

an html div with prettyToggle button

Button Features

- Initial state label is "Application Reset" with warning status
- Reloading state label is "Cancel Application Reset" with danger status

Shiny Usage

Call this function at any place in UI section.

It is paired with a call to `appReset(id, ...)` in server

See Also

appReset

downloadFile

downloadFile_ValidateTypes

downloadFile_AvailableTypes

downloadablePlot

downloadFileButton

downloadableTable

logViewerOutput

Examples

```
if (interactive()) {
   library(shiny)
   library(periscope2)
   shinyApp(
     ui = fluidPage(fluidRow(column(12, appResetButton(id = "appResetId")))),
     server = function(input, output) {
       appReset(id = "appResetId", logger = "")
   })
}
```

`createAlert` *Display alert panel at specified location*

Description

Create an alert panel in server code to be displayed in the specified UI selector location

Usage

```
createAlert(
  session = shiny::getDefaultReactiveDomain(),
  id = NULL,
  selector = NULL,
  options
)
```

Arguments

`session`	- Shiny session object
`id`	- Anchor id (either id or selector only should be set)
`selector`	- Character vector represents jQuery selector to add the alert to is (i.e ".alert-Class", div.badge-danger.navbar-badge). If 'id' is specified, this parameter will be neglected
`options`	- List of options to pass to the alert

Value

html div and inserts it in the app DOM

Shiny Usage

Call this function from `program/server_local.R` or any other server file to setup the needed alert

See Also

bs4Dash:closeAlert()

periscope2:set_app_parameters()

periscope2:ui_tooltip()

periscope2:get_url_parameters()

Examples

```
library(shiny)
library(bs4Dash)

# Inside server_local.R
periscope2::createAlert(id       = "sidebarRightAlert",
                        options  = list(title    = "Right Side",
                                        status   = "success",
                                        closable = TRUE,
                                        content  = "Example Basic Sidebar Alert"))
# Test selector
## a div with class "badge-danger.navbar-badge" must be exist in UI to display alert
selector <- "div.badge-danger.navbar-badge"
periscope2::createAlert(selector = selector,
                        options  = list(title    = "Selector Title",
                                        status   = "danger",
                                        closable = TRUE,
                                        content  = "Selector Alert"))
```

---

create_application *Create a new templated framework application*

Description

Creates ready-to-use templated application files using the periscope2 framework. The application can be created either empty (default) or with a sample/documented example application.

Usage

```
create_application(
  name,
  location,
  sample_app = FALSE,
  left_sidebar = TRUE,
  right_sidebar = FALSE
)
```

Arguments

`name`	- name for the new application and directory
`location`	- base path for creation of `name`
`sample_app`	- whether to create a sample shiny application
`left_sidebar`	- whether the left sidebar should be enabled. It can be TRUE/FALSE
`right_sidebar`	- parameter to set the right sidebar. It can be TRUE/FALSE

Value

no return value, creates application folder structure and files

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
 -- log (log files)
 -- program (user application)
 -- -- config   (application configuration files)
 -- -- data     (user application data)
 -- -- fxn      (user application function)
 -- -- modules  (application modules files)
 -- www (supporting shiny files)
 -- -- css  (application css files)
 -- -- img  (application image files)
 -- -- js   (application js 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/config directory :
Use this location for configuration files.

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/fxn directory :
Use this location for supporting and helper R files.

name/program/modules directory :
Use this location for application new modules files.

***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.

***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_footer.R** :
Create footer UI elements in this file and register them with the framework using a call to add_ui_footer

***name/program*/ui_header.R** :
Create header UI elements in this file and register them with the framework using a call to add_ui_header

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

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

***name*/www/css/custom.css** :
This is the application custom styling css file. User can update application different parts style using this file.

***name*/www/js/custom.js** :
This is the application custom javascript file.

***name*/www/periscope_style.yaml** :
This is the application custom styling yaml file. User can update application different parts style using this file.

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
```

See Also

bs4Dash:dashboardPage()

waiter:waiter_show()

Examples

```
# sample app named 'mytestapp' created in a temp dir
location <- tempdir()
create_application(name = 'mytestapp', location = location, sample_app = TRUE)
unlink(paste0(location,'/mytestapp'), TRUE)

# sample app named 'mytestapp' with a right sidebar using a custom icon created in a temp dir
location <- tempdir()
create_application(name = 'mytestapp', location = location, sample_app = TRUE, right_sidebar = TRUE)
unlink(paste0(location,'/mytestapp'), TRUE)

# blank app named 'myblankapp' created in a temp dir
location <- tempdir()
create_application(name = 'myblankapp', location = location)
unlink(paste0(location,'/myblankapp'), TRUE)

# blank app named 'myblankapp' without a left sidebar created in a temp dir
location <- tempdir()
create_application(name = 'myblankapp', location = location, left_sidebar = FALSE)
unlink(paste0(location,'/myblankapp'), TRUE)
```

---

create_left_sidebar *Add the left sidebar to an existing application*

Description

User can update an existing application that does not have a left side bar and add a new empty one using this function.

Usage

```
create_left_sidebar(location)
```

Arguments

location	path of the existing periscope2 application

Details

If conversion is successful, the following message will be returned *"Add left sidebar conversion was successful. File(s) updated: ui.R, ui_left_sidebar.R"*

If the function called on an application with an existing left bar, message *"Left sidebar already available, no conversion needed"* will be returned with no conversion

If the passed location is invalid, empty, not exist or not a valid periscope2 application, nothing will be added and a related error message will be printed in console

Value

no return value, creates left sidebar related UI R file and updates related source call in ui.R

See Also

create_right_sidebar

`create_right_sidebar` *Add the right sidebar to an existing application*

Description

User can update an existing application that does not have a right side bar and add a new empty one using this function.

Usage

```
create_right_sidebar(location)
```

Arguments

`location`	path of the existing periscope2 application.

Details

If conversion is successful, the following message will be returned *"Add right sidebar conversion was successful. File(s) updated: ui.R, ui_right_sidebar.R"*

If the function called on an application with an existing right bar, message *"Right sidebar already available, no conversion needed"* will be returned with no conversion

If the passed location is invalid, empty, not exist or not a valid periscope2 application, nothing will be added and a related error message will be printed in console

Value

no return value, creates right sidebar related UI R file and updates related source call in ui.R

See Also

create_left_sidebar

`downloadablePlot` *downloadablePlot module server function*

Description

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

Usage

```
downloadablePlot(
  id,
  logger,
  filenameroot,
  aspectratio = 1,
  downloadfxns = list(),
  visibleplot
)
```

Arguments

`id`	the ID of the Module's UI element
`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.

Value

Reactive expression containing the currently selected plot to be available for display and download

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`:

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

See Also

downloadablePlotUI

appResetButton

appReset

downloadFile

downloadFile_ValidateTypes

downloadFile_AvailableTypes

downloadableTable

logViewerOutput

Examples

```
if (interactive()) {
  library(shiny)
  library(ggplot2)
  library(periscope2)
  shinyApp(ui = fluidPage(fluidRow(column(width = 12,
     downloadablePlotUI("object_id1",
                        downloadtypes      = c("png", "csv"),
                        download_hovertext = "Download plot and data",
                        height             = "500px",
                        btn_halign         = "left")))),
    server = function(input, output) {
      download_plot <- function() {
        ggplot(data = mtcars, aes(x = wt, y = mpg)) +
        geom_point(aes(color = cyl)) +
        theme(legend.justification = c(1, 1),
              legend.position      = c(1, 1),
              legend.title         = element_blank()) +
        ggtitle("GGPlot Example ") +
        xlab("wt") +
        ylab("mpg")
      }
      downloadablePlot(id           = "object_id1",
                       logger       = "",
                       filenameroot = "mydownload1",
                     downloadfxns = list(png = download_plot, csv = reactiveVal(mtcars)),
                       aspectratio  = 1.33,
                       visibleplot  = download_plot)
  })
}
```

`downloadablePlotUI` *downloadablePlot module UI function*

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

Value

list of downloadFileButton UI and plot object

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.

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 `downloadablePlot(id, ...)` in server.R

See Also

downloadablePlot

downloadFileButton

clickOpts

hoverOpts

brushOpts

appResetButton

appReset

downloadFile

downloadFile_ValidateTypes

downloadFile_AvailableTypes

downloadableTable

logViewerOutput

Examples

```
if (interactive()) {
  library(shiny)
  library(ggplot2)
  library(periscope2)
  shinyApp(ui = fluidPage(fluidRow(column(width = 12,
     downloadablePlotUI("object_id1",
                        downloadtypes      = c("png", "csv"),
                        download_hovertext = "Download plot and data",
                        height             = "500px",
                        btn_halign         = "left")))),
    server = function(input, output) {
      download_plot <- function() {
        ggplot(data = mtcars, aes(x = wt, y = mpg)) +
        geom_point(aes(color = cyl)) +
```

```
      theme(legend.justification = c(1, 1),
            legend.position      = c(1, 1),
            legend.title         = element_blank()) +
      ggtitle("GGPlot Example ") +
      xlab("wt") +
      ylab("mpg")
    }
    downloadablePlot(id           = "object_id1",
                     logger       = "",
                     filenameroot = "mydownload1",
                   downloadfxns = list(png = download_plot, csv = reactiveVal(mtcars)),
                     aspectratio  = 1.33,
                     visibleplot  = download_plot)
  })
}
```

`downloadableTable` *downloadableTable module server function*

Description

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

Usage

```
downloadableTable(
  id,
  logger,
  filenameroot,
  downloaddatafxns = list(),
  tabledata,
  selection = NULL,
  table_options = list()
)
```

Arguments

`id`	the ID of the Module's UI element
`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.

`selection`	function or reactive expression providing the row_ids of the rows that should be selected
`table_options`	optional table formatting parameters check `?DT::datatable` for options full list. Also see example below to see how to pass options

Details

Generated table can highly customized using function `?DT::datatable` same arguments except for 'options' and 'selection' parameters.

For 'options' user can pass the same `?DT::datatable` options using the same names and values one by one separated by comma.

For 'selection' parameter it can be either a function or reactive expression providing the row_ids of the rows that should be selected.

Also, user can apply the same provided `?DT::formatCurrency` columns formats on passed dataset using format functions names as keys and their options as a list.

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.
- `selection` parameter has different usage than DT::datatable `selection` option. See parameters usage section.
- DT::datatable options `editable`, `width` and `height` are not supported

Shiny Usage

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

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

Note: calling module server returns the reactive expression containing the currently selected rows in the display table.

See Also

downloadableTableUI

downloadFileButton

logViewerOutput

downloadFile

downloadFile_ValidateTypes

downloadFile_AvailableTypes

downloadablePlot

Examples

```
if (interactive()) {
 library(shiny)
 library(periscope2)
 shinyApp(ui = fluidPage(fluidRow(column(width = 12,
   downloadableTableUI("object_id1",
                       downloadtypes = c("csv", "tsv"),
                       hovertext     = "Download the data here!",
                       contentHeight = "300px",
                       singleSelect  = FALSE)))),
   server = function(input, output) {
     mydataRowIds <- function(){
       rownames(head(mtcars))[c(2, 5)]
     }
     selectedrows <- downloadableTable(
       id               = "object_id1",
       logger           = "",
       filenameroot     = "mydownload1",
       downloaddatafxns = list(csv = reactiveVal(mtcars), tsv = reactiveVal(mtcars)),
       tabledata        = reactiveVal(mtcars),
       selection        = mydataRowIds,
       table_options    = list(rownames = TRUE,
                               caption  = "This is a great table!"))
     observeEvent(selectedrows(), {
       print(selectedrows())
     })})
}
```

`downloadableTableUI` *downloadableTable module UI function*

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).

Value

list of downloadFileButton UI and DT datatable

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.

Shiny Usage

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

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

See Also

downloadableTable

downloadFileButton

logViewerOutput

downloadFile

downloadFile_ValidateTypes

downloadFile_AvailableTypes

downloadablePlot

Examples

```
if (interactive()) {
 library(shiny)
 library(periscope2)
 shinyApp(ui = fluidPage(fluidRow(column(width = 12,
   downloadableTableUI("object_id1",
                       downloadtypes = c("csv", "tsv"),
                       hovertext     = "Download the data here!",
                       contentHeight = "300px",
                       singleSelect  = FALSE)))),
   server = function(input, output) {
     mydataRowIds <- function(){
       rownames(head(mtcars))[c(2, 5)]
     }
     selectedrows <- downloadableTable(
       id               = "object_id1",
       logger           = "",
       filenameroot     = "mydownload1",
       downloaddatafxns = list(csv = reactiveVal(mtcars), tsv = reactiveVal(mtcars)),
       tabledata        = reactiveVal(mtcars),
       selection        = mydataRowIds,
       table_options    = list(rownames = TRUE,
                               caption  = "This is a great table!"))
     observeEvent(selectedrows(), {
       print(selectedrows())
     })})
}
```

`downloadFile` *downloadFile module server function*

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(id, logger, filenameroot, datafxns = list(), aspectratio = 1)
```

Arguments

`id`	ID of the Module's UI element
`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).

Value

no return value, called for downloading selected file type

Shiny Usage

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

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

See Also

downloadFileButton

downloadFile_ValidateTypes

downloadFile_AvailableTypes

logViewerOutput

downloadablePlot

downloadableTableUI

downloadableTable

Examples

```
if (interactive()) {
   library(shiny)
   library(periscope2)
   shinyApp(ui = fluidPage(fluidRow(column(width = 6,
     # single download type
     downloadFileButton("object_id1",
                        downloadtypes = c("csv"),
                        hovertext     = "Button 1 Tooltip")),
      column(width = 6,
      # multiple download types
      downloadFileButton("object_id2",
                         downloadtypes = c("csv", "tsv"),
                         hovertext     = "Button 2 Tooltip")))),
     server = function(input, output) {
       # single download type
       downloadFile(id           = "object_id1",
                    logger       = "",
                    filenameroot = "mydownload1",
                    datafxns     = list(csv = reactiveVal(iris)),
                    aspectratio  = 1)
       # multiple download types
```

```
        downloadFile(id           = "object_id2",
                     logger       = "",
                     filenameroot = "mydownload2",
                     datafxns     = list(csv = reactiveVal(mtcars),
                     tsv = reactiveVal(mtcars)))
   })
}
```

`downloadFileButton` *downloadFileButton module UI function*

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

Value

html span with tooltip and either shiny downloadButton in case of single download or shiny actionButton otherwise

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 `downloadFile(id, ...)` in server.R

See Also

downloadFile

downloadFile_ValidateTypes

downloadFile_AvailableTypes

logViewerOutput

downloadablePlot

downloadableTableUI

downloadableTable

Examples

```
if (interactive()) {
   library(shiny)
   library(periscope2)
   shinyApp(ui = fluidPage(fluidRow(column(width = 6,
     # single download type
     downloadFileButton("object_id1",
                        downloadtypes = c("csv"),
                        hovertext     = "Button 1 Tooltip")),
      column(width = 6,
      # multiple download types
      downloadFileButton("object_id2",
                         downloadtypes = c("csv", "tsv"),
                         hovertext     = "Button 2 Tooltip")))),
     server = function(input, output) {
       # single download type
       downloadFile(id           = "object_id1",
                    logger       = "",
                    filenameroot = "mydownload1",
                    datafxns     = list(csv = reactiveVal(iris)),
                    aspectratio  = 1)
       # multiple download types
       downloadFile(id           = "object_id2",
                    logger       = "",
                    filenameroot = "mydownload2",
                    datafxns     = list(csv = reactiveVal(mtcars),
                    tsv = reactiveVal(mtcars)))
   })
}
```

`downloadFile_AvailableTypes` *downloadFile module list of allowed file types*

Description

Returns a list of all supported types

Usage

```
downloadFile_AvailableTypes()
```

Value

a vector of all supported types

See Also

downloadFileButton

downloadFile

`downloadFile_ValidateTypes` *Check passed file types against downloadFile module allowed file types list*

Description

It is a downloadFile module helper to return periscope2 defined file types list and warns user if an invalid type is included

Usage

```
downloadFile_ValidateTypes(types)
```

Arguments

`types`	list of types to test

Value

the list input given in types

See Also

downloadFileButton

downloadFile

logViewerOutput

downloadablePlot

downloadableTableUI

downloadableTable

Examples

```
#inside console
## Check valid types
result <- periscope2::downloadFile_AvailableTypes()
identical(result, c("csv", "xlsx", "tsv", "txt", "png", "jpeg", "tiff", "bmp"))

## check invalid type
testthat::expect_warning(downloadFile_ValidateTypes(types = "csv_invalid"),
                         "file download list contains an invalid type <csv_invalid>")
```

---

get_url_parameters *Parse application passed 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

Shiny Usage

Call this function from `program/server_local.R` or any other server file

See Also

periscope2:set_app_parameters()

periscope2:add_ui_footer()

periscope2:add_ui_left_sidebar()

periscope2:add_ui_header()

periscope2:add_ui_body()

periscope2:add_ui_right_sidebar()

periscope2:ui_tooltip()

Examples

```
library(shiny)
library(periscope2)

# Display application info
observeEvent(input$app_info, {
             url_params <- get_url_parameters(session)
             show_alert(html                = TRUE,
                        showCloseButton     = FALSE,
                        animation           = "slide-from-top",
                        closeOnClickOutside = TRUE,
                        text                = url_params[["passed_paramter"]],
                        title               = "alert title")
})
```

---

logging-entrypoints *Entry points for logging actions*

Description

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

Usage

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

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

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

logerror(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

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.

Value

no return value, prints log contents into R console and app log file

---

`logViewerOutput` *Display app logs*

Description

Creates a shiny table with table containing logged user actions. Table contents are auto updated whenever a user action is logged. The id must match the same id configured in **server.R** file upon calling `fw_server_setup` method

Usage

```
logViewerOutput(id = "logViewer")
```

Arguments

`id`	- character id for the object(default = "logViewer")

Value

shiny tableOutput instance

Table columns

- action - the action that id logged in any place in app
- time - action time

Example

```
logViewerOutput('logViewer')
```

Shiny Usage

Add the log viewer box to your box list

It is paired with a call to `fw_server_setup` method in **server.R** file

See Also

downloadFile

downloadFile_ValidateTypes

downloadFile_AvailableTypes

downloadablePlot

downloadFileButton

downloadableTableUI

downloadableTable

Examples

```
# Inside ui_body add the log viewer box to your box list

logViewerOutput('logViewerId')
```

`periscope2` *Periscope2 Shiny Application Framework*

Description

Periscope2 is the next-generation package following the paradigm of the 'periscope' package to support a UI-standardized and rail-guarded enterprise quality application environment. This package also includes a variety of convenience functions for 'shiny' applications in a more modernized way. Base reusable functionality as well as UI paradigms are included to ensure a consistent user experience regardless of application or developer.

Details

'periscope2' differs from the 'periscope' package as follows:

- Upgraded dependency on bootstrap v4 instead of bootstrap v3
- New user modules (i.e. announcements)
- More functionality and finer control over existing modules such as alert and reset
- More control over customizing different application parts (header, footer, left sidebar, right sidebar and body)
- Enhanced file structure to organize application UI, shiny modules, app configuration, .. etc

A gallery of 'periscope' and 'periscope2' example apps is hosted at http://periscopeapps.org

Function Overview

Create a new framework application instance:
create_application

Set application parameters in program/global.R:
set_app_parameters

Get any url parameters passed to the application:
get_url_parameters

Update an existing application with a needed sidebar:
create_left_sidebar
create_right_sidebar

Register user-created UI objects to the requisite application locations:
add_ui_body
add_ui_footer
add_ui_header
add_ui_left_sidebar
add_ui_right_sidebar

Included shiny modules with a customized UI:
downloadFileButton
downloadableTableUI
downloadablePlotUI
appResetButton
logViewerOutput

High-functionality standardized tooltips:
ui_tooltip

More Information

```
browseVignettes(package = 'periscope2')
```

`set_app_parameters` *Set Application Parameters*

Description

This function sets global parameters customizing the shiny application.

Usage

```
set_app_parameters(
  title,
  app_info = NULL,
  log_level = "DEBUG",
  app_version = "1.0.0",
  loading_indicator = NULL,
  announcements_file = NULL
)
```

Arguments

`title`	- Application title text
`app_info`	- Application detailed information. It can be 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.

`log_level` - Designating the log level to use for the user log as 'DEBUG','INFO', 'WARN' or 'ERROR' (default = 'DEBUG')

`app_version` - Character string designating the application version (default = '1.0.0')

`loading_indicator`
- It uses waiter (see https://waiter.john-coene.com/#/).
Pass a list like list(html = spin_1(), color = "#333e48") to
configure waiterShowOnLoad (refer to the package help for all styles).

`announcements_file`
- The path to announcements configuration file. Use announcementConfigurationsAddin to generate that file.

Value

no return value, called for setting new application global properties

Shiny Usage

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

See Also

periscope2:announcementConfigurationsAddin()

waiter:waiter_show()

periscope2:add_ui_footer()

periscope2:add_ui_left_sidebar()

periscope2:add_ui_header()

periscope2:add_ui_body()

periscope2:add_ui_right_sidebar()

periscope2:ui_tooltip()

periscope2:get_url_parameters()

Examples

```
library(shiny)
library(waiter)
library(periscope2)
```

```
# Inside program/global.R
set_app_parameters(title             = "periscope Example Application",
                   app_info          = HTML("Example info"),
                   log_level         = "DEBUG",
                   app_version       = "1.0.0",
                   loading_indicator  = list(html = tagList(spin_1(), "Loading ...")),
                   announcements_file = "./program/config/announce.yaml")
```

`ui_tooltip` *Add tooltip icon and text to UI elements labels*

Description

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

Usage

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

Arguments

`id`	- The 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
`placement`	- Where to display tooltip label. Available places are "top", "bottom", "left", "right" (default is "top")

Value

html span with the label, tooltip image and tooltip text

Shiny Usage

Call this function from `program/ui_body.R` to set tooltip parameters

See Also

periscope2:add_ui_footer()

periscope2:add_ui_left_sidebar()

periscope2:add_ui_header()

periscope2:add_ui_body()

periscope2:add_ui_right_sidebar()

periscope2:set_app_parameters()

periscope2:ui_tooltip()

periscope2:get_url_parameters()

Examples

```
library(shiny)
library(periscope2)

# Inside ui_body.R or similar UI file
 ui_tooltip(id   = "top_tip",
            label = "Top Tooltips",
            text  = "Top tooltip")
```

Index