Package 'systemPipeShiny'

March 30, 2021

Title systemPipeShiny: An Interactive Framework for Workflow Management and Visualization

Version 1.0.20

Date 2021-03-15

Description systemPipeShiny (SPS) extends the widely used systemPipeR (SPR) workflow environment with a versatile graphical user interface provided by a Shiny App. This allows non-R users, such as experimentalists, to run many systemPipeR's workflow designs, control, and visualization functionalities interactively without requiring knowledge of R. Most importantly, SPS has been designed as a general purpose framework for interacting with other R packages in an intuitive manner. Like most Shiny Apps, SPS can be used on both local computers as well as centralized server-based deployments that can be accessed remotely as a public web service for using SPR's functionalities with community and/or private data. The framework can integrate many core packages from the R/Bioconductor ecosystem. Examples of SPS' current functionalities include: (a) interactive creation of experimental designs and metadata using an easy to use tabular editor or file uploader; (b) visualization of workflow topologies combined with auto-generation of R Markdown preview for interactively designed workflows; (d) access to a wide range of data processing routines; (e) and an extendable set of visualization functionalities. Complex visual results can be managed on a 'Canvas Workbench' allowing users to organize and to compare plots in an efficient manner combined with a session snapshot feature to continue work at a later time. The present suite of preconfigured visualization examples. The modular design of SPR makes it easy to design custom functions without any knowledge of Shiny, as well as extending the environment in the future with contributions from the community.

Depends R (>= 4.0.0), shiny (>= 1.5.0), spsUtil, spsComps (>= 0.1.1), drawer

Imports DT, assertthat, bsplus, crayon, dplyr, ggplot2, glue, magrittr, methods, plotly, rlang, rstudioapi, shinyAce, shinyFiles, shinyWidgets, shinydashboard, shinydashboardPlus (>= 2.0.0), shinyjqui, shinyjs, shinytoastr, stringr, stats, styler, tibble, utils, vroom (>= 1.3.1), yaml

Suggests testthat, BiocStyle, knitr, rmarkdown, systemPipeR, systemPipeRdata, networkD3, rhandsontable, zip, callr, pushbar, fs, openssl, readr, R.utils, DOT, shinyTree, DESeq2, SummarizedExperiment, glmpca, pheatmap, grid, ape, ggtree, Rtsne, UpSetR, tidyr, esquisse, cicerone

VignetteBuilder knitr

biocViews Infrastructure, DataImport, Sequencing, QualityControl, ReportWriting, ExperimentalDesign, Clustering

License GPL (>= 3)

Encoding UTF-8

LazyData true

BugReports https://github.com/systemPipeR/systemPipeShiny/issues

URL https://github.com/systemPipeR/systemPipeShiny

RoxygenNote 7.1.1

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

git_url https://git.bioconductor.org/packages/systemPipeShiny

git_branch RELEASE_3_12

git_last_commit 51f0dac

git_last_commit_date 2021-03-15

Date/Publication 2021-03-29

Author Le Zhang [aut, cre],
Daniela Cassol [aut],
Ponmathi Ramasamy [aut],
Jianhai Zhang [aut],
Gordon Mosher [aut],
Thomas Girke [aut]

Maintainer Le Zhang <le.zhang001@email.ucr.edu>

R topics documented:

`canvasBtn` *Screenshot a plot or UI to SPS Canvas or download as an image*

Description

A upper level function of drawer::toCanvasBtn. You should only use it under SPS projects. For you own apps, still use the drawer::toCanvasBtn.

Usage

```
canvasBtn(dom, id = "", isID = TRUE, class = "text-center", placement = "top")
```

Arguments

dom	a HTML DOM selector, mostly common is to select the element by ID: e.g. a plot with ID "plot1", to select, use `dom = "plot1"` to select the plot if `isID = TRUE`. If `isID = FALSE`, use `dom = "#plot1"` Other complex selector is supported. First turn `isID = FALSE`, then try things like `dom = ".btn i"` selects an icon inside an element with "btn" class. If more than one element is matched, only the first one will be screenshoted.
id	ID of this button, optional.
isID	bool, if the dom argument is selected by ID or other selector
class	string, length 1, other html class add to the button wrapper
placement	where should the tiptool place, top, bottom, left, right.

Value

a button group with several options

Examples

```
canvasBtn("#mydiv")
```

---

genGallery *Generate gallery by only providing tab names*

Description

A fast way in SPS to generate a gallery to display plot tab screenshots

Usage

```
genGallery(
  tab_ids = NULL,
  Id = NULL,
  title = "Gallery",
  type = NULL,
  title_color = "#0275d8",
  image_frame_size = 3,
  app_path = NULL
)
```

Arguments

tab_ids	a vector of tab IDs
Id	element ID
title	gallery title
type	If this value is not NULL, filter by tab type, and tab_ids will be ignored. One of c("core", "wf", "data", "vs"). use spsTabInfo() to see tab information
title_color	title color, common colors or hex code
image_frame_size	integer, 1-12
app_path	app path, default current working directory

Details

require a SPS project and the config/tabs.csv file. If you want to use gallery outside a SPS project, use spsComps::gallery

Value

gallery div

Examples

```
if(interactive()){
    spsInit()
    ui <- fluidPage(
        genGallery(c("plot_example1")),
        genGallery(type = "plot")
    )
    server <- function(input, output, session) {

    }
    shinyApp(ui, server)
}
```

---

genHrefTable *Generate a table that lists tabs by rows*

Description

A fast way in SPS to generate a table that lists some SPS tabs

Usage

```
genHrefTable(
  rows,
  Id = NULL,
  title = "A Table to list tabs",
  text_color = "#0275d8",
  app_path = NULL,
  ...
)
```

Arguments

rows	a named list of character vector, the item names in the list will be the row names and each item should be a vector of tab IDs. Or you can use one of 'core', 'wf', 'vs', 'data', 'plot' to specify a tab type, so it will find all tabs matching that type. See tab_info.csv under config directory for type info.
Id	element ID
title	table title
text_color	text color for table
app_path	app path, default is current working directory
...	any additional arguments to the html element, like class, style...

Details

For rows, there are some specially reserved characters for type and sub-types, one of c('core', 'wf', 'vs', 'data', 'plot'). If indicated, it will return a list of tabs matching the indicated tabs instead of searching individual tab names. See examples.

This function requires a SPS project and the config/tabs.csv file. If you want to use hrefTable outside a SPS project, or want to create some links pointing to outside web resources, use spsComps::hrefTable

Value

HTML elements

Examples

```
if(interactive()){
    spsInit()
    # will be two rows, one row is searched by tab IDs and the other is
    # searched by type.
    rows <- list(row1 = c("core_canvas", "core_about"),
                 row2 =  "data")
    ui <- fluidPage(
        genHrefTable(rows)
    )
    server <- function(input, output, session) {

    }
    shinyApp(ui, server)
}
```

---

removeSpsTab *Remove a SPS tab*

Description

Remove a tab R file and remove from the tabs.csv config file

Usage

```
removeSpsTab(
  tab_id = "none",
  force = FALSE,
  app_path = getwd(),
  multiple = FALSE,
  verbose = spsOption("verbose"),
  colorful = spsOption("use_crayon")
)
```

Arguments

tab_id	tab ID, string, length 1, supports regular expressions, so be careful. If more than one tabs are matched, stop by default
force	bool, whether to ask for confirmation
app_path	app directory
multiple	bool, if matched more than one tab, turn this to *TRUE* can remove more than one tab at a time. Be careful.
verbose	bool, follows project setting, but can be overwrite. *TRUE* will give you more information
colorful	bool, whether the message will be colorful?

Value

remove the tab file and register info in *tabs.csv*

Examples

```
spsInit(change_wd = FALSE, overwrite = TRUE)
spsNewTab("vs_new", app_path = glue::glue("SPS_{format(Sys.time(), '%Y%m%d')}"))
removeSpsTab("vs_new", force = TRUE,
             app_path = glue::glue("SPS_{format(Sys.time(), '%Y%m%d')}"))
```

---

sps *SystemPipeShiny app main function*

Description

SystemPipeShiny app main function

Usage

```
sps(tabs = "", server_expr = NULL, app_path = getwd())
```

Arguments

tabs	custom visualization tab IDs that you want to display, in a character vector. Use spsTabInfo() to see what tab IDs you can load
server_expr	additional top level sever expression you want to run. This will run after the default server expressions. It means you can have access to internal server expression objects, like the shiny::reactiveValues() object shared. You can also overwrite other values. Read "shared object" in vignette.
app_path	SPS project path

Details

You must set the project root as working directory for this function to find required files.

Value

a list contains the UI and server

Examples

```
if(interactive()){
    spsInit()
    sps_app <- sps(
        tabs = "",
        server_expr = {
            msg("Hello World", "GREETING", "green")
        }
    )
}
```

---

spsCoreTabReplace *Overwrite a default SPS tab*

Description

If you want to load your custom content on any of the default tabs in a SPS project, you can overwrite the tab with your own UI and server function. First, use this function to create a template for the tab you want to replace and then fill your own content.

Usage

```
spsCoreTabReplace(
  replace_tab,
  app_path = getwd(),
  open_file = TRUE,
  overwrite = FALSE
)
```

Arguments

replace_tab	one of "welcome", "module_main", "vs_main", "canvas", "about", for the welcome tab, module home tab, custom tab home tab, Canvas tab, about tab respectively.
app_path	string, where is SPS project root path
open_file	bool, open the newly created template if you are in Rstudio?
overwrite	bool, if the template exists, overwrite it with a new, empty one?

Value

a template file

Examples

```
if(interactive()){
    spsInit(project_name = "default_overwrite_demo",
            change_wd = FALSE, open_files = FALSE)
    ## try to run it for the first look
    # shiny::runApp("default_overwrite_demo")
    spsCoreTabReplace("welcome", app_path = "default_overwrite_demo")
    ## edit the file and save it.
    ## run again and watch the difference on the welcome tab.
    shiny::runApp("default_overwrite_demo")
}
```

---

spsEzUI *Easy and simple UI and server for a SPS custom tab*

Description

SPS custom tab simple UI and server , spsEzUI must use together with the spsEzServer function. The easiest way to use is to use spsNewTab function to create both.

Usage

```
spsEzUI(
  desc = "",
  tab_title = "Tab Title",
  plot_title = "My Plot",
  plot_control = shiny::tagList()
)

spsEzServer(
  plot_code,
  example_data_path = system.file(package = "systemPipeShiny", "app", "data",
    "iris.csv"),
  other_server_code = ""
)
```

Arguments

desc	character string, length 1 in markdown format. Tab description and instructions. You can make type it in multiple lines but in only one string (one pair of quotes). e.g.

```
"
# some desc
## second line,
- bullet 1
- bullet 2
"
```

tab_title	string, tab title
plot_title	string, plot title
plot_control	some Shiny components (UI) to control the plot, like plot title, x,y labels, color, font size, etc. Group all components in a shiny `tagList`.
plot_code	some R code to make the plot.
example_data_path	example dataset path, this dataset will be loaded on app start to display
other_server_code	optional, other server R code to run for this tab

Value

`spsEzUI` returns a shiny module UI function, `spsEzServer` returns the server function

See Also

spsNewTab

Examples

```
# use `spsInit()` to create an SPS project and use `spsNewTab("Your_tabID", template = "easy")`
# to create a new tab file. The specified use of these two functions is in that file.
```

---

spsInit *Create a SystemPipeShiny project*

Description

To run a SPS app, you need to first create a SPS project, a directory contains the required files.

Usage

```
spsInit(
  app_path = getwd(),
  project_name = glue::glue("SPS_{format(Sys.time(), '%Y%m%d')}"),
  database_name = "sps.db",
  overwrite = FALSE,
  change_wd = TRUE,
  verbose = FALSE,
```

```
  open_files = TRUE,
  colorful = TRUE
)
```

Arguments

app_path	path, a directory where do you want to create this project, must exist.
project_name	Your project name, default is SPS_ + time
database_name	deprecated in current version. project database name, recommend to use the default name: "sps.db". It is used to store app meta information.
overwrite	bool, overwrite the app_path if there is a folder that has the same name as project_name?
change_wd	bool, when creation is done, change working directory into the project?
verbose	bool, do you want additional message?
open_files	bool, If change_wd == TRUE and you are also in Rstudio, it will open up *global.R* for you
colorful	bool, should message from this function be colorful?

Details

Make sure you have write permission to app_path.

The database in not used in current version.

Value

creates the project folder

Examples

```
if(interactive()){
    spsInit(change_wd = FALSE)
}
```

---

spsNewTab *Create a new SPS tab*

Description

create custom tabs in newer (> 1.1) version of SPS. The old creation functions will be deprecated by next Bioconductor major release.

Usage

```
spsNewTab(
  tab_id = "vs_mytab",
  tab_displayname = "My custom plotting tab",
  img = "",
  app_path = getwd(),
  out_folder_path = file.path(app_path, "R"),
  author = "",
```

```
  template = c("simple", "full"),
  preview = FALSE,
  reformat = FALSE,
  open_file = TRUE,
  verbose = spsOption("verbose"),
  colorful = spsOption("use_crayon")
)
```

Arguments

tab_id	character string, length 1, must be unique. Use spsTabInfo(app_path = "YOUR_APP_PATH") to see current tab IDs.
tab_displayname	character string, length 1, the name to be displayed on side navigation bar list and tab title
img	realtive path, an image representation of the new plot. It can be a internet link or a local link which uses the *www* folder as the root. e.g. drop your image *plot.png* inside *www/plot_list*, then the link here is "plot_list/plot.png". You will see these images on "Custom Tabs" main page. If no provided, a warning will be given on app start and an empty image will show up on "Custom Tabs".
app_path	string, app directory, default is current directory
out_folder_path	string, which directory to write the new tab file, default is the *R* folder in the SPS project. If you write the file other than *R*, this file will not be automatically loaded by SPS or Shiny. You must source it manually.
author	character string, or a vector of strings. authors of the tab
template	one of "simple" or "full", default "simple". "simple" gives a tab file with minimum Shiny code, you can only focus on you R plotting code. "full" gives the full tab code, so you can modify everything on the tab.
preview	bool, *TRUE* will print the new tab code to console and will not write the file and will not register the tab
reformat	bool, whether to use styler::style_file reformat the code
open_file	bool, if Rstudio is detected, open the new tab file?
verbose	bool, default follows the project verbosity level. *TRUE* will give you more information on progress and debugging
colorful	bool, whether the message will be colorful or not

Details

- template "simple": hides the UI and server code and use spsEzUI and spsEzServer instead.
- template "full": full tab code. You need to know some Shiny development knowledge.

Value

returns a new tab file

Examples

```
spsInit(change_wd = FALSE, overwrite = TRUE)
spsNewTab("vs_newtab_ez", app_path = glue::glue("SPS_{format(Sys.time(), '%Y%m%d')}"))
spsNewTab("vs_newtab_full", template = "full",
app_path = glue::glue("SPS_{format(Sys.time(), '%Y%m%d')}"))
spsNewTab("vs_newtab_pre", preview = TRUE,
app_path = glue::glue("SPS_{format(Sys.time(), '%Y%m%d')}"))
```

`spsOptDefaults` *Print SPS options*

Description

Make sure you have created the app directory and it has *config/config.yaml* file.

spsOptDefaults prints out all default and other avaliable values for each option. spsOptions print all current set option values.

Note: the spsUtil::spsOption is used to get or set a **single** option value. spsOptions is used to print **all** current option values. If you need to set all values at once, use the *global.R* file under SPS project root.

Usage

```
spsOptDefaults(app_path = getwd())

spsOptions(app_path = getwd(), show_legend = TRUE)
```

Arguments

`app_path`	path, where is the app directory
`show_legend`	bool, show the color legend?

Value

cat to console SPS option values

Examples

```
if(interactive()){
    # start a SPS project
    spsInit(open_files = FALSE)
    viewSpsDefaults()
    # change a few options
    options(sps = list(
        mode = "server",
        warning_toast = TRUE,
        loading_screen = FALSE,
        loading_theme = "vhelix",
        use_crayon = TRUE
    ))
    # view current options
    spsOptions()
}
```

spsTabInfo *View SPS project 'config/tabs.csv' information*

Description

View SPS project 'config/tabs.csv' information

Usage

```
spsTabInfo(return_type = "print", n_print = 40, app_path = getwd())
```

Arguments

return_type	one of 'print', 'data', 'colnames', or a specified column name
n_print	how many lines of tab info you want to print out
app_path	SPS project root

Details

- 'print' will print out the entire *tabs.csv*, you can specify `n_print` for how many lines you want to print;
- 'data' will return the tab info tibble
- 'colnames' will return all column names of tab info file
- A column name will extract the specified column out and return as a vector

Value

return depends on `return_type`

Examples

```
spsInit(project_name = "SPS_tabinfo", overwrite = TRUE,
        change_wd = FALSE, open_files = FALSE)
# all lines
spsTabInfo("print", app_path = "SPS_tabinfo")
# 5 lines
spsTabInfo("print", app_path = "SPS_tabinfo", n_print = 5L)
spsTabInfo("data", app_path = "SPS_tabinfo")
spsTabInfo("colnames", app_path = "SPS_tabinfo")
spsTabInfo("tab_id", app_path = "SPS_tabinfo")
```

Index