Package ‘formods’

May 25, 2024

Type Package

Title 'Shiny' Modules for General Tasks

Version 0.1.6

Maintainer John Harrold <john.m.harrold@gmail.com>

Description 'Shiny' apps can often make use of the same key elements, this package provides modules for common tasks (data upload, wrangling data, figure generation and saving the app state), and also a framework for developing. These modules can react and interact as well as generate code to create reproducible analyses.

License BSD_2_clause + file LICENSE

BugReports https://github.com/john-harrold/formods/issues

URL https://formods.ubiquity.tools/

Encoding UTF-8

Depends R (>= 4.2.0)

Imports cli, digest, dplyr, ggforce, ggplot2, onbrand (>= 1.0.3), readr, readxl, rang, rhandsontable, shiny, shinyAce, shinyWidgets, stats, stringr, tools, writexl, yaml, zip

Suggests clipr, covr, devtools, DT, flextable, ggpur, gtools, here, janitor, knitr, plotly, prompter, rmarkdown, shinybusy, shinydashboard, testthat (>= 3.0.0), utils

RoxygenNote 7.3.1

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author John Harrold [aut, cre] (<https://orcid.org/0000-0003-2052-4373>)

Repository CRAN

Date/Publication 2024-05-25 02:40:02 UTC
R topics documented:

ASM_fetch_code ............................ 4
ASM_fetch_dfln ............................ 4
ASM_fetch_state ........................... 5
ASM_init_state ............................ 6
ASM_Server ................................ 7
ASM_test_mksession ....................... 10
ASM_write_state ........................... 11
autocast .................................. 12
dwrs_builder ............................... 13
DW_add_wrangling_element ................. 14
DW_append_report .......................... 15
DW_attach_ds .............................. 16
dw Eval_element ........................... 17
DW_fetch_code .............................. 18
DW_fetch_current_view ..................... 19
DW_fetch_ds ................................ 20
DW_fetch_state ............................ 21
DW_init_state .............................. 23
DW_new_view ............................... 24
DW_Server .................................. 25
DW_set_current_view ....................... 28
DW_test_mksession ......................... 29
DW_update_checksum ....................... 30
fers_builder ............................... 31
fetch_hold .................................. 31
fetch_package_version ..................... 32
FG_append_report .......................... 33
FG_build .................................... 34
FG_extract_page ........................... 35
FG_fetch_app_code .......................... 37
FG_fetch_app_info ........................ 37
FG_fetch_current_fig ...................... 38
FG_fetch_state ............................ 38
FG_init_state .............................. 41
FG_new_fig .................................. 42
FG_Server ................................... 43
FG_set_current_fig ........................ 46
FG_test_mksession ......................... 47
FG_update_checksum ....................... 48
FM_add_ui_tooltip .......................... 49
FM_build_comment ........................ 50
FM_fetch_app_code .......................... 51
FM_fetch_app_info .......................... 51
FM_fetch_app_state ........................ 52
FM_fetch_current_mods ........................ 53
FM_fetch_data_format ........................ 53
FM_fetch_deps .............................. 54
topics documented:

FM_fetch_ds ............................................................ 55
FM_fetch_log_path .................................................. 56
FM_fetch_mdl ......................................................... 57
FM_fetch_mod_state ................................................ 58
FM_fetch_user_files_path ........................................ 59
FM_generate_report ................................................ 59
FM_init_state ....................................................... 61
FM_le ................................................................. 62
FM_message .......................................................... 63
FM_mk_error_fig .................................................... 64
FM_notify ............................................................. 64
FM_pause_screen ..................................................... 66
FM_pretty_sort ....................................................... 66
FM_proc_include ..................................................... 67
FM_resume_screen .................................................. 68
FM_set_app_state ................................................... 68
FM_set_mod_state ................................................... 69
FM_set_notification ................................................ 70
FM_set_ui_msg ........................................................ 71
FM_tc ................................................................. 72
formods ............................................................... 73
formods_check ........................................................ 73
has_changed .......................................................... 74
has_updated .......................................................... 74
icon_link .............................................................. 75
is_installed .......................................................... 76
linspace ................................................................. 76
new_module_template .............................................. 77
remove_hold .......................................................... 78
set_hold ............................................................... 79
UD_attach_ds .......................................................... 80
UD_ds_read ............................................................ 82
UD_fetch_code ........................................................ 83
UD_fetch_ds ........................................................... 84
UD_fetch_state ........................................................ 85
UD_init_state ........................................................ 86
UD_Server .............................................................. 87
UD_test_mksession .................................................. 90
unfactor ............................................................... 91
use_formods .......................................................... 92

Index 94
### ASM_fetch_code

**Fetch Module Code**

**Description**
Fetched the code to generate results seen in the app

**Usage**
ASM_fetch_code(state)

**Arguments**
- state: ASM state from ASM_fetch_state()

**Value**
The ASM module does not generate code

**Examples**
```r
# Creating a state object for testing
sess_res = ASM_test_mksession(session=list(), full_session=FALSE)
state = sess_res$state
code = ASM_fetch_code(state)
```

### ASM_fetch_dlfn

**Fetch Download File Name**

**Description**
Gets either the file name specified by the user or the default value if that is null

**Usage**
ASM_fetch_dlfn(state, extension = ".zip")

**Arguments**
- state: ASM state from ASM_fetch_state()
- extension: File extension for the download (default: ".zip")

**Value**
character object with the download file name
**ASM_fetch_state**

**Examples**

```r
# Creating a state object for testing
sess_res = ASM_test_mksession(session=list(), full_session=FALSE)
state = sess_res$state
dlfn = ASM_fetch_dlfn(state)
dlfn
```

---

**ASM_fetch_state**  
*Fetch State Manager State*

---

**Description**

Merges default app options with the changes made in the UI

**Usage**

```r
ASM_fetch_state(id, input, session, FM_yaml_file, MOD_yaml_file)
```

**Arguments**

- **id**  
  Shiny module ID
- **input**  
  Shiny input variable
- **session**  
  Shiny session variable
- **FM_yaml_file**  
  App configuration file with FM as main section.
- **MOD_yaml_file**  
  Module configuration file with MC as main section.

**Value**

A list containing the current state of the app, including default values from the yaml file as well as any changes made by the user. The list has the following structure:

- **yaml**: Full contents of the supplied yaml file.
- **MC**: Module components of the yaml file.
- **ASM**:
  - **isgood**: Boolean object indicating if the file was successfully loaded.
  - **checksum**: This is an MD5 sum of the loaded state file
- **MOD_TYPE**: Character data containing the type of module "ASM"
- **id**: Character data containing the module id module in the session variable.
- **FM_yaml_file**: App configuration file with FM as main section.
- **MOD_yaml_file**: Module configuration file with MC as main section.
ASM_init_state

Initialize ASM Module State

Description

Creates a list of the initialized module state

Usage

ASM_init_state(FM_yaml_file, MOD_yaml_file, id, session)

Arguments

FM_yaml_file: App configuration file with FM as main section.
MOD_yaml_file: Module configuration file with MC as main section.
id: ID string for the module.
session: Shiny session variable

Value

list containing an empty ASM state
ASM_Server

Examples

# Within shiny the session variable will exist,
# this creates an example here for testing purposes:
sez_res = ASM_test_mksession(session=list(), full_session=FALSE)
session = sess_res$session
state = ASM_init_state(
  FM_yaml_file = system.file(package = "formods",
                              "templates",
                              "formods.yaml"),
  MOD_yaml_file = system.file(package = "formods",
                              "templates",
                              "ASM.yaml"),
  id = "ASM",
  session = session)
state

---

ASM_Server                  Save State Server

Description

Server function for the Save State Shiny Module

Usage

ASM_Server(
  id,
  FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml"),
  MOD_yaml_file = system.file(package = "formods", "templates", "ASM.yaml"),
  deployed = FALSE,
  react_state = NULL,
  mod_ids
)

Arguments

id          An ID string that corresponds with the ID used to call the modules UI elements
FM_yaml_file App configuration file with FM as main section.
MOD_yaml_file Module configuration file with MC as main section.
deployed    Boolean variable indicating whether the app is deployed or not.
react_state Variable passed to server to allow reaction outside of module (NULL)
mod_ids     Vector of module IDs and order they are needed (used for code generation).

Value

UD Server object
Examples

if(interactive()){
  # These are suggested packages
  library(shinydashboard)
  library(ggpubr)
  library(plotly)
  library(shinybusy)
  library(prompter)
  library(utils)
  library(clipr)
  library(formods)

  CSS <- "
  .wrapfig {
    float: right;
    shape-margin: 20px;
    margin-right: 20px;
    margin-bottom: 20px;
  }
  "

  # Default to not deployed
  if(!exists("deployed")){
    deployed = FALSE
  }

  #https://fontawesome.com/icons?from=io

  ui <- dashboardPage(
    skin="black",
    dashboardHeader(title="formods"),
    dashboardSidebar(
      sidebarMenu(
        menuItem("Source Data", tabName="upload", icon=icon("table")),
        menuItem("Wrangle", tabName="wrangle", icon=icon("hat-cowboy")),
        menuItem("Plot", tabName="plot", icon=icon("chart-line")),
        menuItem("App State", tabName="app_state", icon=icon("archive")),
        menuItem("App Info", tabName="sysinfo", icon=icon("book-medical"))
      ),
    ),
    dashboardBody(
      tags$head(
        tags$style(HTML(CSS))
      ),
      tabItems(
        tabItem(tabName="app_state",
          box(title="Manage App State",
            htmlOutput(NS("ASM", "ui_asm_compact"))
          )
        ),
        tabItem(tabName="upload",
          box(title="Load Data", width=12,
Formods is a set of modules and an framework for developing modules which interact and create code to replicate analyses performed within an app. To experiment download this', tags$p('and upload it into the App using the form on the left.'))
)}
),

tabItem(tabName="wrangle",
box(title="Transform and Create Views of Your Data", width=12,
htmlOutput(NS("DW", "DW_ui_compact")))),

tabItem(tabName="plot",
box(title="Visualize Data", width=12,
htmlOutput(NS("FG", "FG_ui_compact")))),

tabItem(tabName="sysinfo",
box(title="System Details", width=12,
shinydashboard::tabBox(
  width = 12,
  title = NULL,
  shiny::tabPanel(id="sys_modules",
    title=tagList(shiny::icon("ghost"),
                  "Modules"),
    htmlOutput(NS("ASM", "ui_asm_sys_modules"))
  ),
  shiny::tabPanel(id="sys_packages",
    title=tagList(shiny::icon("ghost"),
                  "Packages"),
    htmlOutput(NS("ASM", "ui_asm_sys_packages"))
  ),
  shiny::tabPanel(id="sys_log",
    title=tagList(shiny::icon("clipboard-list"),
                  "App Log"),
    verbatimTextOutput(NS("ASM", "ui_asm_sys_log"))
  ),
  shiny::tabPanel(id="sys_options",
    title=tagList(shiny::icon("sliders"),
                  "R Options"),
    htmlOutput(NS("ASM", "ui_asm_sys_options"))
  )
))
)
# Main app server
server <- function(input, output, session) {
  # Empty reactive object to track and react to
  # changes in the module state outside of the module
  react_FM = reactiveValues()

  # This is the list of module ids used for reproducible script generation. The
  # order here is important.
  mod_ids = c("UD", "DW", "FG")

  #Populating with test data
  # FG_test_mksession(session)
  # Module servers
  formods::ASM_Server(id="ASM",
    deployed = deployed,
    react_state = react_FM, mod_ids = mod_ids)
  formods::UD_Server( id="UD", id_ASM = "ASM",
    deployed = deployed,
    react_state = react_FM)
  formods::DW_Server( id="DW", id_ASM = "ASM",id_UD = "UD",
    deployed = deployed,
    react_state = react_FM)
  formods::FG_Server( id="FG", id_ASM = "ASM",id_UD = "UD", id_DW = "DW",
    deployed = deployed,
    react_state = react_FM)
}

shinyApp(ui, server)
ASM_write_state

Arguments

- `session`: Shiny session variable (in app) or a list (outside of app).
- `id`: An ID string that corresponds with the ID used to call the modules UI elements.
- `id_UD`: An ID string that corresponds with the ID used to call the UD modules UI elements.
- `id_DW`: An ID string that corresponds with the ID used to call the DW modules UI elements.
- `full_session`: Boolean to indicate if the full test session should be created (default `TRUE`).

Value

A list with the following elements:

- `isgood`: Boolean indicating the exit status of the function.
- `session`: The value Shiny session variable (in app) or a list (outside of app) after initialization.
- `input`: The value of the shiny input at the end of the session initialization.
- `state`: App state.
- `rsc`: The `react_state` components.

Examples

```r
sess_res = ASM_test_mksession(session=list(), full_session=FALSE)
```

---

Write State to File for Saving

Description

Called from download handler and used to write a saved state value if that is null.

Usage

```r
ASM_write_state(state, session, file, mod_ids)
```

Arguments

- `state`: ASM state from `ASM_fetch_state()`.
- `session`: Shiny session variable.
- `file`: File name to write zipped state.
- `mod_ids`: Vector of module IDs and order they are needed (used for code generation).

Value

This function only writes the state and has no return value.
Examples

# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
sess_res = ASM_test_mksession(session=list(), full_session=FALSE)
session = sess_res$session
input = sess_res$input

# Configuration files
FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml")
MOD_yaml_file = system.file(package = "formods", "templates", "ASM.yaml")

# We need to specify the ID of the ASM module
id = "ASM"

state = ASM_fetch_state(id = id,
    input = input,
    session = session,
    FM_yaml_file = FM_yaml_file,
    MOD_yaml_file = MOD_yaml_file)

ASM_write_state(state, session,
    file = tempfile(fileext=".zip"),
    mod_ids = c("UD"))

caption: autocast

autocast automatically cast UI input variable

description

Takes UI input and tries to figure out if it's numeric or text

usage

autocast(ui_input, quote_char = TRUE)

arguments

ui_input UI input from a shiny form
quote_char TRUE will include double quotes in the character string

value

Best guess of type casting applied to the ui_input

examples

number = autocast('10')
text = autocast('ten')
dwrs_builder

Builds a Data Wrangling R Statement From UI Elements:

Description

Takes the current UI elements and constructs the appropriate data wrangling command from the user input.

Usage

dwrs_builder(state)

Arguments

state

DW state from DW_fetch_state()

Value

list containing the following elements

• isgood: Return status of the function
• cmd: Data wrangling R command
• action: The action being performed
• desc: Verbose description of the action
• msgs: Messages to be passed back to the user

Examples

library(formods)
# The example requires a formods DW state object
state = DW_test_mksession(session=list())$state
state["DW"]["ui"]["select_dw_element"] = "filter"
state["DW"]["ui"]["select_fds_filter_column"] = "EVID"
state["DW"]["ui"]["select_fds_filter_operator"] = "=="
state["DW"]["ui"]["fds_filter_rhs"] = 0

# This builds the data wrangling statement based on
# elements scraped from the UI
dwb_res = dwrs_builder(state)

# Here we evaluate the resulting command:
dwee_res = dw_eval_element(state, dwb_res["cmd"])

# Next we add this wrangling element to the state
state = DW_add_wrangling_element(state, dwb_res, dwee_res)

# This creates a new data view and makes it active
state = DW_new_view(state)
Here we can pluck out that data view from the state:

```r
current_view = DW_fetch_current_view(state)
```

This will update the key in this view:

```r
current_view[['key']] = "My new view"
```

And this will place it back into the state:

```r
state = DW_set_current_view(state, current_view)
```

## Description

Adds the wrangling element to the current data view.

## Usage

```r
DW_add_wrangling_element(state, dwb_res, dwee_res)
```

## Arguments

- **state**: DW state from `DW_fetch_state()`
- **dwb_res**: Output from `dwrs_builder()`
- **dwee_res**: Output from `dw_eval_element()` returned by `UD_fetch_state()`.

## Value

state with data set attached

## Examples

```r
library(formods)

# The example requires a formods DW state object
state = DW_test_mksession(session=list())$state
state["DW"]["ui"]["select_dw_element"] = "filter"
state["DW"]["ui"]["select_fds_filter_column"] = "EVID"
state["DW"]["ui"]["select_fds_filter_operator"] = "=="
state["DW"]["ui"]["fds_filter_rhs"] = 0

# This builds the data wrangling statement based on
# elements scraped from the UI

dwb_res = dwrs_builder(state)

dwee_res = dw_eval_element(state, dwb_res[['cmd']])

# Here we evaluate the resulting command:

dwee_res = dw_eval_element(state, dwb_res[['cmd']])
```
# Next we add this wrangling element to the state
state = DW_add_wrangling_element(state, dwb_res, dwee_res)

# This creates a new data view and makes it active
state = DW_new_view(state)

# Here we can pluck out that data view from the state
current_view = DW_fetch_current_view(state)

# This will update the key in this view
current_view["key"] = "My new view"

# And this will place it back into the state
state = DW_set_current_view(state, current_view)

---

**DW_append_report**

*Append Report Elements*

**Description**

Takes the current state of the app and appends data views to an xlsx report object.

**Usage**

`DW_append_report(state, rpt, rpttype, gen_code_only = FALSE)`

**Arguments**

- **state**: DW state from `DW_fetch_state()`
- **rpt**: Report with the current content of the report which will be appended to in this function. For details on the structure see the documentation for `FM_generate_report`.
- **rpttype**: Type of report to generate (supported "xlsx").
- **gen_code_only**: Boolean value indicating that only code should be generated (FALSE).

**Value**

list containing the following elements

- **isgood**: Return status of the function.
- **hasrptele**: Boolean indicator if the module has any reportable elements.
- **code**: Code to generate reporting elements.
- **msgs**: Messages to be passed back to the user.
- **rpt**: Report with any additions passed back to the user.

**See Also**

`FM_generate_report`
**Examples**

# We need a state object to use below
```r
sess_res = DW_test_mksession(session=list())
state = sess_res$state
```

```r
rpt = list(summary = list(), sheets=list())
rpt_res = DW_append_report(state, rpt = rpt, rpttype = "xlsx")
```

# Shows if report elements are present
```r
rpt_res$hasrptele
```

# Code chunk to generate report element
```r
cat(paste(rpt_res$code, collapse="\n"))
```

# Tabular summary of data views
```r
rpt_res$rpt$summary
```

---

**DW_attach_ds**  
*Attach Data Set to DW State*

**Description**

Attaches a dataset to the DW state supplied.

**Usage**

```r
DW_attach_ds(state, id_UD, session)
```

**Arguments**

- **state**: DW state from `DW_fetch_state()`
- **id_UD**: ID string for the upload data module used to handle uploads
- **session**: Shiny session variable

**Value**

state with data set attached

**Examples**

# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
```r
sess_res = DW_test_mksession(session=list())
session = sess_res$session
input = sess_res$input
```
# We also need a state variable
state = sess_res$state

# We need to identify the UD module with the data
id_UD = "UD"
state = DW_attach_ds(state, id_UD, session)

dw_eval_element

_Evaluates Data Wrangling Generated Code_

**Description**

Takes the current state and a string containing a data wrangling command and evaluates it.

**Usage**

dw_eval_element(state, cmd)

**Arguments**

- state: DW state from DW_fetch_state()
- cmd: string containing the data wrangling command

**Value**

list with the following elements

- isgood: Return status of the function.
- msgs: Messages to be passed back to the user.
- DS: Wrangled dataset.

**Examples**

```r
library(formods)
# The example requires a formods DW state object
state = DW_test_mksession(session=list())$state
state["DW"]["ui"]["select_dw_element"] = "filter"
state["DW"]["ui"]["select_fds_filter_column"] = "EVID"
state["DW"]["ui"]["select_fds_filter_operator"] = "=="
state["DW"]["ui"]["fds_filter_rhs"] = 0

# This builds the data wrangling statement based on
# elements scraped from the UI
dwb_res = dwrs_builder(state)

# Here we evaluate the resulting command:
dwee_res = dw_eval_element(state, dwb_res["cmd"])
```
# Next we add this wrangling element to the state
state = DW_add_wrangling_element(state, dwb_res, dwee_res)

# This creates a new data view and makes it active
state = DW_new_view(state)

# Here we can pluck out that data view from the state
current_view = DW_fetch_current_view(state)

# This will update the key in this view
current_view["key"] = "My new view"

# And this will place it back into the state
state = DW_set_current_view(state, current_view)

## DW_fetch_code

<table>
<thead>
<tr>
<th>Fetch Module Code</th>
</tr>
</thead>
</table>

### Description

Fetches the code to generate results seen in the app

### Usage

`DW_fetch_code(state)`

### Arguments

- **state**: DW state from `DW_fetch_state()`

### Value

Character object vector with the lines of code and isgood)

### Examples

# This will create a formods DW state object for the example
sess_res = DW_test_mksession(session=list())
state = sess_res$state
code = DW_fetch_code(state)
cat(code)
**DW_fetch_current_view**  
*Fetches Current Data View*

**Description**
Takes a DW state and returns the current active view

**Usage**

```r
DW_fetch_current_view(state)
```

**Arguments**

- `state`  
  DW state from `DW_fetch_state()`

**Value**
List containing the details of the active data view. The structure of this list is the same as the structure of `state$DW$views` in the output of `DW_fetch_state()`.

**Examples**

```r
code
library(formods)
# The example requires a formods DW state object
state = DW_test_mksession(session=list())$state
state[['DW']['ui']['select_dw_element']] = "filter"
state[['DW']['ui']['select_fds_filter_column']] = "EVID"
state[['DW']['ui']['select_fds_filter_operator']] = "=="
state[['DW']['ui']['fds_filter_rhs']] = 0

# This builds the data wrangling statement based on elements scraped from the UI
dwb_res = dwrs_builder(state)

# Here we evaluate the resulting command:
dwee_res = dw_eval_element(state, dwb_res[['cmd']])

# Next we add this wrangling element to the state
state = DW_add_wrangling_element(state, dwb_res, dwee_res)

# This creates a new data view and makes it active
state = DW_new_view(state)

# Here we can pluck out that data view from the state
current_view = DW_fetch_current_view(state)

# This will update the key in this view
current_view[['key']] = "My new view"

# And this will place it back into the state
```
state = DW_set_current_view(state, current_view)

---

**DW_fetch_ds** | *Fetch Module Datasets*

**Description**
 Fetches the datasets contained in the module.

**Usage**

```r
DW_fetch_ds(state)
```

**Arguments**

- `state` | UD state from `UD_fetch_state()`

**Value**
 Character object vector with the lines of code list containing the following elements

- `isgood`: Return status of the function.
- `hasds`: Boolean indicator if the module has any datasets
- `msgs`: Messages to be passed back to the user.
- `ds`: List with datasets. Each list element has the name of the R-object for that dataset. Each element has the following structure:
  - `label`: Text label for the dataset
  - `MOD_TYPE`: Short name for the type of module.
  - `id`: module ID
  - `DS`: Dataframe containing the actual dataset.
  - `DSMETA`: Metadata describing DS, see `FM_fetch_ds()` for details on the format.
  - `code`: Complete code to build dataset.
  - `checksum`: Module checksum.
  - `DSchecksum`: Dataset checksum.

**Examples**

```r
# We need a state variable
sess_res = DW_test_mksession(session=list())
state = sess_res$state

ds = DW_fetch_ds(state)
```
**DW_fetch_state**

**Fetch Data Wrangling State**

**Description**

Merges default app options with the changes made in the UI

**Usage**

```r
DW_fetch_state(
  id,
  input,
  session,
  FM_yaml_file,
  MOD_yaml_file,
  id_UD,
  react_state
)
```

**Arguments**

- **id**: Shiny module ID
- **input**: Shiny input variable
- **session**: Shiny session variable
- **FM_yaml_file**: App configuration file with FM as main section.
- **MOD_yaml_file**: Module configuration file with MC as main section.
- **id_UD**: ID string for the upload data module used to handle uploads or the name of the list element in react_state where the data set is stored.
- **react_state**: Variable passed to server to allow reaction outside of module (NULL)

**Value**

List containing the current state of the DM module including default values from the yaml file as well as any changes made by the user. The structure of the list is defined below.

- **yaml**: Contents of the yaml file.
- **MC**: Module components of the yaml file.
- **DW**: Data wrangling state
  - **isgood**: Boolean status of the state. FALSE if the dataset identified by id_UD is bad.
  - **checksum**: MD5 sum indicating if there was a change in the datasets within the view. Use this to trigger updates in response to changes in this module.
  - **button_counters**: List of counters to detect button clicks.
  - **code_previous**: Loading code from the UD field.
  - **current_view**: View id of the current active data wrangling view.
- UD: Copy of the "UD" field of the id_UD from the react_state input.
- ui: Current value of form elements in the UI
- ui_hold: List of hold elements to disable updates before a full ui refresh is complete.
- view_cntr: Counter for tracking view ids, value contains the id of the last view created.
- views: List of data wrangling views. Each view has the following structure:
  * checksum: MD5 sum of WDS
  * code: Code to generate WDS from start to finish
  * code_dw_only: Code for just the wrangling portion.
  * code_previous: Code to load data and assign to view object.
  * elements_table: Table of data wrangling elements.
  * id: Character id (view_idx)
  * idx: Numeric id (1)
  * isgood: Boolean status of the data view. False if evaluation fails
  * key: User key (short description)
  * view_ds_object_name: Object name for this data view
  * WDS: Current value of the data view with all of the successful commands in elements_table evaluated.

- MOD_TYPE: Character data containing the type of module “DW”
- id: Character data containing the module id
- FM_yaml_file: App configuration file with FM as main section.
- MOD_yaml_file: Module configuration file with MC as main section. module in the session variable.

Examples

# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
sess_res = DW_test_mksession(session=list())
session = sess_res$session
input = sess_res$input

# Configuration files
FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml")
MOD_yaml_file = system.file(package = "formods", "templates", "DW.yaml")

# We need to specify both the DW module id as well as the
# id of the UD module that feeds into it.
id = "DW"
id_UD = "UD"

# Creating an empty state object
state = DW_fetch_state(id = id,
  input = input,
  session = session,
  FM_yaml_file = FM_yaml_file,
  MOD_yaml_file = MOD_yaml_file,
  id_UD = "UD",
)
**DW_init_state**

初始化 DW 模块状态

**Description**

创建初始化模块状态的列表

**Usage**

```r
DW_init_state(FM_yaml_file, MOD_yaml_file, id, id_UD, session)
```

**Arguments**

- `FM_yaml_file`：带有 FM 主部分的应用配置文件。
- `MOD_yaml_file`：带有 MC 主部分的模块配置文件。
- `id`：Shiny 模块 ID。
- `id_UD`：用于处理上传或存储数据集的上传数据模块的 ID 字符串。
- `session`：Shiny 会话变量模块 (NULL)

**Value**

- 包含空 DW 状态的列表

**Examples**

```r
# 在 shiny 中，session 和 input 变量将存在，
# 这里创建示例用于测试目的：
sess_res = DW_test_mksession(session=list())
session = sess_res$session
input = sess_res$input

state = DW_init_state(
  FM_yaml_file = system.file(package = "formods",
  "templates",
  "formods.yaml"),
  MOD_yaml_file = system.file(package = "formods",
  "templates",
  "DW.yaml"),
  id = "DW",
  id_UD = "UD",
  session = session)

state
```
**New Data Wrangling View**

**Description**

Appends a new empty data wrangling view to the DW state object and makes this new view the active view.

**Usage**

`DW_new_view(state)`

**Arguments**

- `state`: DW state from `DW_fetch_state()`

**Value**

DW state object containing a new data view and that view set as the current active view. See the help for `DW_fetch_state()` for view format.

**Examples**

```r
library(formods)
# The example requires a formods DW state object
state = DW_test_mksession(session=list())$state
state["DW"][["ui"]]["select_dw_element"] = "filter"
state["DW"][["ui"]]["select_fds_filter_column"] = "EVID"
state["DW"][["ui"]]["select_fds_filter_operator"] = "=="
state["DW"][["ui"]]["fds_filter_rhs"] = 0

# This builds the data wrangling statement based on elements scraped from the UI
dwb_res = dwrs_builder(state)

# Here we evaluate the resulting command:
dwee_res = dw_eval_element(state, dwb_res["cmd"])

# Next we add this wrangling element to the state
state = DW_add_wrangling_element(state, dwb_res, dwee_res)

# This creates a new data view and makes it active
state = DW_new_view(state)

# Here we can pluck out that data view from the state
current_view = DW_fetch_current_view(state)

# This will update the key in this view
current_view["key"] = "My new view"
```
# And this will place it back into the state
state = DW_set_current_view(state, current_view)

---

**DW_Server**

### Description

Server function for the data wrangling module

### Usage

```
DW_Server(
  id,
  id_ASM = "ASM",
  id_UD = "UD",
  FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml"),
  MOD_yaml_file = system.file(package = "formods", "templates", "DW.yaml"),
  deployed = FALSE,
  react_state = NULL
)
```

### Arguments

- **id**: An ID string that corresponds with the ID used to call the modules UI elements
- **id_ASM**: ID string for the app state managment module used to save and load app states
- **id_UD**: ID string for the upload data module used to handle uploads or the name of the list element in react_state where the data set is stored.
- **FM_yaml_file**: App configuration file with FM as main section.
- **MOD_yaml_file**: Module configuration file with DW as main section.
- **deployed**: Boolean variable indicating whether the app is deployed or not.
- **react_state**: Variable passed to server to allow reaction outside of module (NULL)

### Value

DW Server object

### Examples

```
if(interactive()){
  # These are suggested packages
  library(shinydashboard)
  library(ggpubr)
  library(plotly)
  library(shinybusy)
  library(prompter)
  ```
library(utils)
library(clipr)
library(formods)

CSS <- "
  .wrapfig {
    float: right;
    shape-margin: 20px;
    margin-right: 20px;
    margin-bottom: 20px;
  }
"

# Default to not deployed
if(!exists("deployed")){
  deployed = FALSE
}

#https://fontawesome.com/icons?from=io

ui <- dashboardPage(
  skin="black",
  dashboardHeader(title="formods"),
  dashboardSidebar(
    sidebarMenu(
      menuItem("Source Data", tabIndex="upload", icon=icon("table")),
      menuItem("Wrangle", tabIndex="wrangle", icon=icon("hat-cowboy")),
      menuItem("Plot", tabIndex="plot", icon=icon("chart-line")),
      menuItem("App State", tabIndex="app_state", icon=icon("archive")),
      menuItem("App Info", tabIndex="sysinfo", icon=icon("book-medical"))
    ),
  ),
  dashboardBody(
    tags$head(
      tags$style(HTML(CSS))
    ),
    tabItems(
      tabItem(tabName="app_state",
        box(title="Manage App State",
          htmlOutput(NS("ASM", "ui_asm_compact")))))
    ,
    tabItem(tabName="upload",
      box(title="Load Data", width=12,
        fluidRow(
          prompter::use_prompt(),
          column(width=6,
            htmlOutput(NS("UD", "UD_ui_compact")))))
      ,
    tags$p(
      tags$img(
        class = "wrapfig",
        src = "https://github.com/john-harrold/formods/raw/master/man/figures/logo.png")
    )
  )
)
Formods is a set of modules and a framework for developing modules which interact and create code to replicate analyses performed within an app. To experiment download this, and upload it into the App using the form on the left.

```r
# Main app server
server <- function(input, output, session) {
  # Empty reactive object to track and react to changes in the module state outside of the module
  react_FM = reactiveValues()
```

```r
tabItem(tabName="wrangle",
  box(title="Transform and Create Views of Your Data", width=12, htmlOutput(NS("DW", "DW_ui_compact")))),
tabItem(tabName="plot",
  box(title="Visualize Data", width=12, htmlOutput(NS("FG", "FG-ui_compact")))),
tabItem(tabName="sysinfo",
  box(title="System Details", width=12,
  shinydashboard::tabBox(
    width = 12,
    title = NULL,
    shiny::tabPanel(id="sys_modules",
      title=tagList(shiny::icon("ghost"), "Modules"),
      htmlOutput(NS("ASM", "ui_asm_sys_modules")))
    ,
    shiny::tabPanel(id="sys_packages",
      title=tagList(shiny::icon("ghost"), "Packages"),
      htmlOutput(NS("ASM", "ui_asm_sys_packages")))
    ,
    shiny::tabPanel(id="sys_log",
      title=tagList(shiny::icon("clipboard-list"), "App Log"),
      verbatimTextOutput(NS("ASM", "ui_asm_sys_log")))
    ,
    shiny::tabPanel(id="sys_options",
      title=tagList(shiny::icon("sliders"), "R Options"),
      htmlOutput(NS("ASM", "ui_asm_sys_options")))
    )
  )))
}
```
# This is the list of module ids used for reproducible script generation. The
# order here is important.
mod_ids = c("UD", "DW", "FG")

# Populating with test data
# FG_test_mksession(session)
# Module servers
formods::ASM_Server(id="ASM",
    deployed = deployed,
    react_state = react_FM, mod_ids = mod_ids)
formods::UD_Server( id="UD", id_ASM = "ASM",
    deployed = deployed,
    react_state = react_FM)
formods::DW_Server( id="DW", id_ASM = "ASM", id_UD = "UD",
    deployed = deployed,
    react_state = react_FM)
formods::FG_Server( id="FG", id_ASM = "ASM", id_UD = "UD", id_DW = "DW",
    deployed = deployed,
    react_state = react_FM)
}
shinyApp(ui, server)
}

---

**DW_set_current_view**  
_Sets Current Data View_

**Description**

Takes a DW state and an updated view and sets that view to the current view_id

**Usage**

```R
DW_set_current_view(state, dw_view)
```

**Arguments**

- **state**  
  DW state from `DW_fetch_state()`

- **dw_view**  
  Data view list of the format returned from `DW_fetch_current_view()` (see the structure of `state$DW$views` in the output of `DW_fetch_state()`).

**Value**

DW state object with the value of `dw_view` set to the current view id.
Examples

```
library(formods)

# The example requires a formods DW state object
state = DW_test_mksession(session=list())$state
state[["DW"][["ui"][["select_dw_element"]]] = "filter"
state[["DW"][["ui"][["select_fds_filter_column"]]] = "EVID"
state[["DW"][["ui"][["select_fds_filter_operator"]]] = "=="
state[["DW"][["ui"][["fds_filter_rhs"]]] = 0

# This builds the data wrangling statement based on elements scraped from the UI
dwb_res = dwrs_builder(state)

# Here we evaluate the resulting command:
dwee_res = dw_eval_element(state, dwb_res[["cmd"]])

# Next we add this wrangling element to the state
state = DW_add_wrangling_element(state, dwb_res, dwee_res)

# This creates a new data view and makes it active
state = DW_new_view(state)

# Here we can pluck out that data view from the state
current_view = DW_fetch_current_view(state)

# This will update the key in this view
current_view[["key"]]] = "My new view"

# And this will place it back into the state
state = DW_set_current_view(state, current_view)
```

---

**DW_test_mksession**  
*Populate Session Data for Module Testing*

**Description**

Populates the supplied session variable for testing.

**Usage**

```
DW_test_mksession(session, id = "DW", id_UD = "UD")
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>Shiny session variable (in app) or a list (outside of app)</td>
</tr>
<tr>
<td>id</td>
<td>An ID string that corresponds with the ID used to call the modules UI elements</td>
</tr>
<tr>
<td>id_UD</td>
<td>An ID string that corresponds with the ID used to call the UD modules UI elements</td>
</tr>
</tbody>
</table>
Value

list with the following elements

• isgood: Boolean indicating the exit status of the function.
• session: The value Shiny session variable (in app) or a list (outside of app) after initialization.
• input: The value of the shiny input at the end of the session initialization.
• state: App state.
• rsc: The react_state components.

Examples

```
sess_res = DW_test_mksession(session=list())
```

---

**DW_update_checksum**

Updates DW Module Checksum

Description

Takes a DW state and updates the checksum used to trigger downstream updates

Usage

```
DW_update_checksum(state)
```

Arguments

state

DW state from `DW_fetch_state()`

Value

DW state object with the checksum updated

Examples

```
# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
sess_res = DW_test_mksession(session=list())
session = sess_res$session
input = sess_res$input

# We also need a state variable
state = sess_res$state

state = DW_update_checksum(state)
```
**fers_builder**

*Builds a Figure Element R Statement From UI Elements:*

**Description**

Takes the current UI elements and constructs the appropriate ggplot commands from the user input. The plot commands assume the existence of a ggplot object `p`.

**Usage**

```r
fers_builder(state)
```

**Arguments**

- `state` FG state from `FG_fetch_state()`

**Value**

list containing the following elements:

- **isgood**: Return status of the function.
- **cmd**: ggplot R command as a character string
- **element**: The type of element being added
- **desc**: Verbose description of the element
- **msgs**: Messages to be passed back to the user

**Examples**

```r
sess_res = FG_test_mksession(session=list(), full_session=FALSE)
state = sess_res$state
fb_res = fers_builder(state)
```

---

**fetch_hold**

*Fetches the Hold Status UI Element Supplied*

**Description**

When some buttons are clicked they will change the state of the system, but other UI components will not detect that change correctly. So those triggers are put on hold. This will fetch hold status for a specified inputId.

**Usage**

```r
fetch_hold(state, inputId = NULL)
```
Arguments

state module state with all of the current ui elements populated
inputId The input ID of the UI element that was put on hold

Value

Boolean value with the hold status

Examples

# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
sess_res = DW_test_mksession(session=list())
session = sess_res$session
input = sess_res$input

# For this example we also need a state variable
state = sess_res$state

# This sets a hold on the specified inputID. This is normally done in
# your XX_fetch_state() function.
state = set_hold(state, inputId = "select_dw_views")

# This will fetch the hold status of the specified inputID.
fetch_hold(state, inputId = "select_dw_views")

# This will remove the hold and is normally done in one of the UI outputs
# with a priority set to ensure it happens after the rest of the UI has
# refreshed.
state = remove_hold(state, session, inputId = "select_dw_views")

fetch_package_version

Fetches the Current Version of Package

Description

The specified package version is extracted and returned. This can simply be the version installed from CRAN or if a development version from GitHub is used details from that will be returned.

Usage

fetch_package_version(pkgname)

Arguments

pkgname Name of package
FG_append_report

Value

String with the version information

Examples

# This package should exist
fetch_package_version('digest')

# This package should not exist
fetch_package_version('bad package name')

FG_append_report  Append Report Elements

Description

Description

Usage

FG_append_report(state, rpt, rpttype, gen_code_only = FALSE)

Arguments

state  FG state from FG_fetch_state()

rpt  Report with the current content of the report which will be appended to in this function. For details on the structure see the documentation for FM_generate_report.

rpttype  Type of report to generate (supported "xlsx", "pptx", "docx").

gen_code_only  Boolean value indicating that only code should be generated (FALSE).

Value

list containing the following elements

- isgood: Return status of the function.
- hasrptele: Boolean indicator if the module has any reportable elements.
- code: Data wrangling R command.
- msgs: Messages to be passed back to the user.
- rpt: Report with any additions passed back to the user.

See Also

FM_generate_report
Examples

```
sess_res = FG_test_mksession(session=list(), full_session=FALSE)
state = sess_res$state
# This will read in the default PowerPoint report template
rpt =
onbrand::read_template(
  template = system.file(package="onbrand","templates","report.pptx"),
  mapping = system.file(package="onbrand","templates","report.yaml"))

rpt_res =
  FG_append_report(state = state, rpt = rpt,
  rpttype = "pptx", gen_code_only=TRUE)

# Shows if report elements are present
rpt_res$hasrpteles

# Code chunk to generate report element
cat(paste(rpt_res$code, collapse="\n"))
```

---

**FG_build**  
*Evaluates Figure Generation Code*

**Description**

Takes the current state and rebuilds the active figure. If the elements table has a row flagged for deletion, it will be deleted. If the cmd input is not NULL it will attempt to append that element to the figure.

**Usage**

```
FG_build(
  state,
  del_row = NULL,
  cmd = NULL,
  element = "unknown",
  desc = "unknown"
)
```

**Arguments**

- `state`  
  FG state from `FG_fetch_state()`
- `del_row`  
  Row number to be deleted (NULL if no rows need to be deleted)
- `cmd`  
  String containing the plotting command. Set to NULL to initialize a new figure or force a rebuild after a dataset update.
- `element`  
  Short name for the figure element being performed, eg. point
- `desc`  
  Verbose description for the action being performed
**FG_extract_page**

**Value**

list with the following elements

- isgood: Return status of the function.
- msgs: Messages to be passed back to the user.
- pages: List with each element containing a ggplot object (p) and the code to generate that object (code)

**Examples**

```r
library(formods)
# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
sess_res = FG_test_mksession(session=list(), full_session=FALSE)
session = sess_res$session
input = sess_res$input

# This will create a populated FG state object:
state = sess_res$state

# This sets the current active figure to Fig_1
state["FG"][["current_fig"]]? = "Fig_1"

# This is a paginated figure, and we can access a specific
# figure using the following:
pg_1 = FG_extract_page(state, 1)
pg_2 = FG_extract_page(state, 2)

# This will give you access to the current figure directly:
current_fig = FG_fetch_current_fig(state)

# For example this will set the key for that figure:
current_fig$key = "Individual profiles by cohort (multiple pages)"

# Once you're done you can put it back into the state:
state = FG_set_current_fig(state, current_fig)

# If you made any changes to the actual figure, this will
# force a rebuild of the current figure:
state = FG_build(state=state, del_row=NULL, cmd=NULL)

# To create a new empty figure you can do this:
state = FG_new_fig(state)
```

**FG_extract_page**

Extracts Specific Page from Paginated Figure
Description

Used to extract the specified page from the current figure.

Usage

\[
\text{FG\_extract\_page}(\text{state, page})
\]

Arguments

- \text{state} FG state from \text{FG\_fetch\_state()}
- \text{page} Page number to extract

Value

\begin{itemize}
\item ggplot object with the specified page.
\end{itemize}

Examples

\begin{verbatim}
library(formods)
# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
sess_res = FG\_test\_mksession(session=list(), full_session=FALSE)
session = sess_res$session
input = sess_res$input

# This will create a populated FG state object:
state = sess_res$state

# This sets the current active figure to Fig_1
state["FG"]["current\_fig"] = "Fig_1"

# This is a paginated figure, and we can access a specific
# figure using the following:
pg_1 = FG\_extract\_page(state, 1)
pg_2 = FG\_extract\_page(state, 2)

# This will give you access to the current figure directly:
current\_fig = FG\_fetch\_current\_fig(state)

# For example this will set the key for that figure:
current\_fig$key = "Individual profiles by cohort (multiple pages)"

# Once you're done you can put it back into the state:
state = FG\_set\_current\_fig(state, current\_fig)

# If you made any changes to the actual figure, this will
# force a rebuild of the current figure:
state = FG\_build( state=state, del\_row = NULL, cmd = NULL)

# To create a new empty figure you can do this:
\end{verbatim}
FG_fetch_code

state = FG_new_fig(state)

FG_fetch_code  Fetch Module Code

Description
Fetches the code to generate results seen in the app

Usage
FG_fetch_code(state)

Arguments
state    UD state from FG_fetch_state()

Value
Character object vector with the lines of code

Examples

# This will create a populated FG state object:
sess_res = FG_test_mksession(session=list(), full_session=FALSE)
state   = sess_res$state
code    = FG_fetch_code(state)
cat(paste(code, collapse="\n"))

FG_fetch_current_fig  Fetches Current Figure

Description
Takes a FG state and returns the current active figure

Usage
FG_fetch_current_fig(state)

Arguments
state    FG state from FG_fetch_state()
FG_fetch_state

Value

List containing the details of the active figure. The structure of this list is the same as the structure of `state$FG$figs` in the output of `FG_fetch_state()`.

Examples

```r
library(formods)
# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
.sess_res = FG_test_mksession(session=list(), full_session=FALSE)
session = sess_res$session
input = sess_res$input

# This will create a populated FG state object:
state = sess_res$state

# This sets the current active figure to Fig_1
state["FG" ]["current_fig"] = "Fig_1"

# This is a paginated figure, and we can access a specific
# figure using the following:
pg_1 = FG_extract_page(state, 1)
pg_2 = FG_extract_page(state, 2)

# This will give you access to the current figure directly:
current_fig = FG_fetch_current_fig(state)

# For example this will set the key for that figure:
current_fig$key = "Individual profiles by cohort (multiple pages)"

# Once you're done you can put it back into the state:
state = FG_set_current_fig(state, current_fig)

# If you made any changes to the actual figure, this will
# force a rebuild of the current figure:
state = FG_build( state=state, del_row = NULL, cmd = NULL)

# To create a new empty figure you can do this:
state = FG_new_fig(state)
```

FG_fetch_state    Fetch Figure Generation State

Description

Merges default app options with the changes made in the UI
Usage

FG_fetch_state(
    id,
    input,
    session,
    FM_yaml_file,
    MOD_yaml_file,
    id_ASM = NULL,
    id_UD = NULL,
    id_DW = NULL,
    react_state
)

Arguments

id          Shiny module ID
input       Shiny input variable
session     Shiny session variable
FM_yaml_file App configuration file with FM as main section.
MOD_yaml_file Module configuration file with MC as main section.
id_ASM      ID string for the app state management module used to save and load app states
id_UD       ID string for the upload data module used to handle uploads or the name of the
             list element in react_state where the data set is stored.
id_DW       ID string for the data wrangling module to process any uploaded data
react_state Variable passed to server to allow reaction outside of module (NULL)

Value

list containing the current state of the app including default values from the yaml file as well as any
changes made by the user. The structure of the list is defined below:

- yaml: Contents of the yaml file.
- MC: Module components of the yaml file.
- FG: Data wrangling state
  - isgood: Boolean status of the state. Currently just TRUE
  - button_counters: List of counters to detect button clicks.
  - ui_msg: Message returned when users perform actions.
  - ui: Current value of form elements in the UI.
  - ui_ids: Vector of UI elements for the module.
  - ui_hold: List of hold elements to disable updates before a full UI refresh is complete.
  - checksum: checksum of the FG module used to detect changes in the module.
  - aes_elements: Plot elements defined by aesthetics (i.e. the X in geom_X)
  - current_fig: fig_id of the currently figure.
  - fig_cntr: Counter for figures, incremented each time a new figure is created.
- DSV: Available data sets from the UD and DW modules.
- figs: List of figures. Each view has the following structure:
  - add_isgood: JMH
  - checksum: Checksum of the figure used to detect changes in the figure.
  - code: Code to generate figure from start to finish.
  - code_fg_only: Code to just generate the figure.
  - code_previous: Code to load and/or wrangle the dataset.
  - elements_table: Table of figure generation elements.
  - fg_object_name: JMH
  - fig_dsview: Name of the dataset view for the current figure (also the R object name of the dataset view).
  - fobj: JMH
  - id: Character id (fig_idx)
  - idx: Numeric id (1)
  - isgood: Boolean status of the figure. FALSE if evaluation/build fails.
  - key: Figure key acts as a title/caption (user editable)
  - msgs: JMH
  - notes: Figure notes (user editable)
  - num_pages: JMH
  - page: JMH

- MOD_TYPE: Character data containing the type of module "DW"
- id: Character data containing the module id module in the session variable.
- FM_yaml_file: App configuration file with FM as main section.
- MOD_yaml_file: Module configuration file with MC as main section.

Examples

```r
# Configuration files
FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml")
MOD_yaml_file = system.file(package = "formods", "templates", "FG.yaml")

# We need to specify both the FG module id as well as the id of the UD module that feeds into it.
id = "FG"
id_UD = "UD"
id_DW = "DW"

# These would be the Shiny input and session variables
input = list()
session = list()

# Creating an empty state object
state = FG_fetch_state(id = id,
                       input = input,
                       session = session,
                       FM_yaml_file = FM_yaml_file,
                       MOD_yaml_file = MOD_yaml_file)
```

FG_init_state

$$\begin{align*}
\text{MOD\_yaml\_file} & = \text{MOD\_yaml\_file}, \\
\text{id\_UD} & = \text{id\_UD}, \\
\text{id\_DW} & = \text{id\_DW}, \\
\text{react\_state} & = \text{NULL} \\
\end{align*}$$

state

FG_init_state | Initialize FG Module State

**Description**

Creates a list of the initialized module state

**Usage**

$$\text{FG_init_state(FM\_yaml\_file, MOD\_yaml\_file, id, id\_UD, id\_DW, session)}$$

**Arguments**

- **FM\_yaml\_file** App configuration file with FM as main section.
- **MOD\_yaml\_file** Module configuration file with MC as main section.
- **id** Shiny module ID
- **id\_UD** ID string for the upload data module used to handle uploads or the name of the list element in react\_state where the data set is stored.
- **id\_DW** ID string for the data wrangling module to process any uploaded data
- **session** Shiny session variable

**Value**

list containing an empty app state object

**Examples**

```r
# These would be the Shiny input and session variables
input = list()
session = list()

state = FG_init_state(
  FM\_yaml\_file = system.file(package = "formods",
                            "templates",
                            "formods.yaml"),
  MOD\_yaml\_file = system.file(package = "formods",
                            "templates",
                            "FG.yaml"),
  id = "FG",
  id\_UD = "UD",
  id\_DW = "DW",
)```
FG_new_fig  

Initialize New Figure

**Description**

Creates a new figure in a FG module

**Usage**

```r
FG_new_fig(state)
```

**Arguments**

- `state`  
  FG state from `FG_fetch_state()`

**Value**

FG state object containing a new empty figure and that figure set as the current active figure

**Examples**

```r
library(formods)
# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
sess_res = FG_test_mksession(session=list(), full_session=FALSE)
session = sess_res$session
input = sess_res$input

# This will create a populated FG state object:
state = sess_res$state

# This sets the current active figure to Fig_1
state[["FG"]][["current_fig"]]] = "Fig_1"

# This is a paginated figure, and we can access a specific
# figure using the following:
pg_1 = FG_extract_page(state, 1)
pg_2 = FG_extract_page(state, 2)

# This will give you access to the current figure directly:
current_fig = FG_fetch_current_fig(state)

# For example this will set the key for that figure:
current_fig$key = "Individual profiles by cohort (multiple pages)"
```
FG_Server

# Once you're done you can put it back into the state:
state = FG_set_current_fig(state, current_fig)

# If you made any changes to the actual figure, this will
# force a rebuild of the current figure:
state = FG_build(state=state, del_row = NULL, cmd = NULL)

# To create a new empty figure you can do this:
state = FG_new_fig(state)

---

### FG_Server

**Figure Generation Server**

**Description**

Server function for the figure generation module

**Usage**

```r
FG_Server(
  id,
  FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml"),
  MOD_yaml_file = system.file(package = "formods", "templates", "FG.yaml"),
  id_ASM = "ASM",
  id_UD = "UD",
  id_DW = "DW",
  deployed = FALSE,
  react_state = NULL
)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>An ID string that corresponds with the ID used to call the module's UI function</td>
</tr>
<tr>
<td>FM_yaml_file</td>
<td>App configuration file with FM as main section.</td>
</tr>
<tr>
<td>MOD_yaml_file</td>
<td>Module configuration file with MC as main section.</td>
</tr>
<tr>
<td>id_ASM</td>
<td>ID string for the app state management module used to save and load app states</td>
</tr>
<tr>
<td>id_UD</td>
<td>ID string for the upload data module used to handle uploads or the name of the list element in react_state where the data set is stored.</td>
</tr>
<tr>
<td>id_DW</td>
<td>ID string for the data wrangling module to process any uploaded data</td>
</tr>
<tr>
<td>deployed</td>
<td>Boolean variable indicating whether the app is deployed or not.</td>
</tr>
<tr>
<td>react_state</td>
<td>Variable passed to server to allow reaction outside of module (NULL)</td>
</tr>
</tbody>
</table>

**Value**

FG Server object
Examples

```r
if(interactive()){
  # These are suggested packages
  library(shinydashboard)
  library(ggpubr)
  library(plotly)
  library(shinybusy)
  library(prompter)
  library(utils)
  library(clipr)
  library(formods)

  CSS <- "
  .wrapfig {
    float: right;
    shape-margin: 20px;
    margin-right: 20px;
    margin-bottom: 20px;
  }
"

  # Default to not deployed
  if(!exists("deployed")){
    deployed = FALSE
  }

  #https://fontawesome.com/icons?from=io

  ui <- dashboardPage(
    skin="black",
    dashboardHeader(title="formods"),
    dashboardSidebar(
      sidebarMenu(
        menuItem("Source Data",  tabName="upload",  icon=icon("table")),
        menuItem("Wrangle",    tabName="wrangle",    icon=icon("hat-cowboy")),
        menuItem("Plot",       tabName="plot",       icon=icon("chart-line")),
        menuItem("App State",  tabName="app_state",  icon=icon("archive")),
        menuItem("App Info",   tabName="sysinfo",    icon=icon("book-medical"))
      )
    ),
    dashboardBody(
      tags$head(
        tags$style(HTML(CSS))
      ),
      tabItems(
        tabItem(tabName="app_state",
            box(title="Manage App State",
                htmlOutput(NS("ASM", "ui_asm_compact"))))),
        tabItem(tabName="upload",
            box(title="Load Data", width=12,
            htmlOutput(NS("ui_asm", "ui_asm_compact"))))
    ))
}
Formods is a set of modules and an framework for developing modules which interact and create code to replicate analyses performed within an app. To experiment download this and upload it into the App using the form on the left.
# Main app server
server <- function(input, output, session) {
  # Empty reactive object to track and react to
  # changes in the module state outside of the module
  react_FM = reactiveValues()

  # This is the list of module ids used for reproducible script generation. The
  # order here is important.
  mod_ids = c("UD", "DW", "FG")

  # Populating with test data
  # FG_test_mksession(session)
  # Module servers
  formods::ASM_Server(id="ASM",
    deployed = deployed,
    react_state = react_FM, mod_ids = mod_ids)
  formods::UD_Server( id="UD", id_ASM = "ASM",
    deployed = deployed,
    react_state = react_FM)
  formods::DW_Server( id="DW", id_ASM = "ASM", id_UD = "UD",
    deployed = deployed,
    react_state = react_FM)
  formods::FG_Server( id="FG", id_ASM = "ASM", id_UD = "UD", id_DW = "DW",
    deployed = deployed,
    react_state = react_FM)
}

shinyApp(ui, server)

---

**FG_set_current_fig**  
Sets Current Figure

**Description**

Takes a FG state and a figure list and sets that figure list as the value for the active figure

**Usage**

FG_set_current_fig(state, fig)

**Arguments**

- state  
  FG state from FG_fetch_state()
- fig  
  Figure list from FG_fetch_current_fig
Value

State with the current figure updated

Examples

library(formods)
# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
sess_res = FG_test_mksession(session=list(), full_session=FALSE)
session = sess_res$session
input   = sess_res$input

# This will create a populated FG state object:
state  = sess_res$state

# This sets the current active figure to Fig_1
state["FG"][["current_fig"]]["Fig_1"]

# This is a paginated figure, and we can access a specific
# figure using the following:
pg_1 = FG_extract_page(state, 1)
pg_2 = FG_extract_page(state, 2)

# This will give you access to the current figure directly:
current_fig = FG_fetch_current_fig(state)

# For example this will set the key for that figure:
current_fig$key = "Individual profiles by cohort (multiple pages)"

# Once you're done you can put it back into the state:
state = FG_set_current_fig(state, current_fig)

# If you made any changes to the actual figure, this will
# force a rebuild of the current figure:
state = FG_build( state=state, del_row = NULL, cmd = NULL)

# To create a new empty figure you can do this:
state = FG_new_fig(state)
FG_update_checksum

Usage

FG_update_checksum(state)

Description

Called after any changes to figures, this function will update the checksum of the module. This allows other modules to determine if there were any changes to the figures within it.

Usage

FG_update_checksum(state)
FM_add_ui_tooltip

Arguments

state FG state from FG_fetch_state()

Value

state with checksum updated.

Examples

# This will create a populated FG state object:

```r
sess_res = FG_test_mksession(session=list(), full_session=FALSE)
state = sess_res$state
state = FG_update_checksum(state)
```

Description

Adds a tool tip to a user element.

Usage

```r
FM_add_ui_tooltip(
    state,
    uiele,
    tooltip = "mytooltip",
    position = "right",
    size = "medium"
)
```

Arguments

state Current module state after yaml file has been read.
uiele UI element to add the tooltip to.
tooltip Text containing the tool tip.
position Position of the tooltip.
size size of the tooltip

Value

If tooltips are enabled and the suggested packages are installed then a uiele with the tooltip added will be returned. Otherwise it will just return the original uiele unchanged.
Examples

if(interactive()){
  # We need a module state object to use this function:
  id="UD"
  sess_res = UD_test_mksession(session=list(), id=id)
  state = sess_res$state
  uiele = shiny::textInput(inputId = "my input", label="example input")
  uiele = FM_add_ui_tooltip(state, uiele)
}

Description

Takes a character string and builds a comment so it will be formatted as a section at the specified level in RStudio

Usage

FM_build_comment(level = 1, comment_str)

Arguments

level Integer (1 (default),2, or 3) indicating the section level of the comment.
comment_str Character object.

Value

Formatted comment.

Examples

FM_build_comment(1, "This is a level 1 header")

FM_build_comment(2, paste0(rep("Long string repeated.", 5), collapse=" "))
**FM_fetch_app_code**  
*Fetches the Code to Reproduce Analysis*

**Description**

Takes the current state of the app and builds a script to reproduce the analysis within the app.

**Usage**

```r
FM_fetch_app_code(session, state, mod_ids)
```

**Arguments**

- `session`: Shiny session variable
- `state`: module state after yaml read
- `mod_ids`: Vector of module IDs and order they are needed (used for code generation).

**Value**

list with the following elements:

- `isgood`: Boolean indicating the whether code generation was successful (TRUE)
- `msgs`: Any messages generated
- `code`: Code to regenerate the app

**Examples**

```r
# We need a Shiny session object to use this function:
sess_res = DW_test_mksession(session=list())
session = sess_res$session
state = sess_res$state
app_code = FM_fetch_app_code(session = session,
                               state = state,
                               mod_ids = c("UD", "DW"))
cat(app_code$code)
```

---

**FM_fetch_app_info**  
*Fetches Information About the App*

**Description**

Returns diagnostic information about the app

**Usage**

```r
FM_fetch_app_info(session)
```
**FM_fetch_app_state**

Fetches the App State

---

**Description**

Returns the entire state of the App

**Usage**

`FM_fetch_app_state(session)`

**Arguments**

- `session` Shiny session variable.

**Value**

App state or NULL if it’s not defined.
**Examples**

```r
# We need a Shiny session object to use this function:
id="UD"
sess_res = UD_test_mksession(session=list(), id=id)
session = sess_res$session
app_state = FM_fetch_app_state(session)
app_state
```

---

**FM_fetch_current_mods**  
Fetches Details About Current Modules

**Description**

Use this to get information about the currently supported modules. This includes short names, UI elements.

**Usage**

```r
FM_fetch_current_mods()
```

**Value**

list with details about the currently supported modules.

**Examples**

```r
FM_fetch_current_mods()
```

---

**FM_fetch_data_format**  
Creates Formatting Information for Datasets

**Description**

Takes a data frame and information in the site configuration to produce formatting information to make it easier for the user to see data type information.

**Usage**

```r
FM_fetch_data_format(df, state)
```

**Arguments**

- `df`  
  Raw dataframe to be built into an rhandsontable.

- `state`  
  Current module state after yaml file has been read.
Value

list with the following elements:

- **col_heads**: List (element for each column) of formatting information for column headers to be use with rhandsontable.
- **col_subtext**: List (element for each column) of subtext to be displayed in selections using 'pickerInput' from the 'shinyWidgets' package.

Examples

```r
# We need a module state object to use this function:
sess_res = UD_test_mksession(session=list())
state = sess_res$state

data_file_local = system.file(package="formods", "test_data", "TEST_DATA.xlsx")
sheet = "DATA"

df = readxl::read_excel(path=data_file_local, sheet=sheet)

hfmt = FM_fetch_data_format(df, state)

# Column header formatting
head(as.vector(unlist(hfmt["col_heads"])))

# Column select subtext
head(as.vector(unlist(hfmt["col_subtext"])))
```

---

**FM_fetch_deps** *Fetches Dependency Information*

**Description**

For a given state and session this function will determine the module ids that are dependent as well as any packages the module elements might depend on.

**Usage**

`FM_fetch_deps(state, session)`

**Arguments**

- **state**: Current module state after yaml file has been read
- **session**: Shiny session variable
Value

list with the following elements:

- mod_ids Dependent module ids.
- packages List of package dependencies.
- package_code Library commands to load packages.

Examples

```r
# We need a Shiny session object to use this function:
id = "UD"
sess_res = UD_test_mkession(session=list(), id=id)
session = sess_res$session
state = sess_res$state
mod_deps = FM_fetch_deps(state, session)
```

FM_fetch_ds  Fetches Datasets from Modules in the App

Description

Loops through each specified module ID or all modules if no ID was specified. For each ID, an attempt will be made to extract any datasets available.

Usage

```r
FM_fetch_ds(state, session, ids = NULL)
```

Arguments

- state: Current module state after yaml file has been read
- session: Shiny session variable
- ids: Vector of ID strings for the modules containing the datasets or NULL for all datasets available.

Value

list containing the current dataset with the following format:

- isgood: Boolean indicating the whether a dataset was found (FALSE)
- ds: List of datasets with element names corresponding to the R object name for that dataset. This has the following format
  - label: Text label for the dataset (used to display to the user)
  - DS: Data frame with the dataset
  - DSMETA: Data frame with metadata about the columns of the dataset in DS. The data frame should have the following columns:
* col1: column 1
  - code: Code to generate the dataset.
  - checksum: Module checksum when the dataset was pulled
  - DSchecksum: Checksum of the dataframe in DS

* catalog: Dataframe containing the a tabular catalog of the datasets found.
  - label: Text label
  - object: Name of the R Object containing the data frame
  - MOD_TYPE: Short name of the type of module
  - id: Module ID
  - checksum: Module checksum
  - DSchecksum: Checksum of the dataset
  - code: Code to generate the dataset

* modules: List with an entry for each module. The element name is the short name. Each of these is a list with an entry that is the shiny module ID. For each of these there is a checksum. For example to access the checksum of a DW module with a module ID of 'my_id', you would use the following: res$modules$DW$my_id.

Examples

```r
# We need a module state and a Shiny session variable
# to use this function:
id="UD"
sess_res = UD_test_mksession(session=list(), id=id)
session = sess_res$session
state = sess_res$state
ds = FM_fetch_ds(state, session)
ds$catalog
```

**FM_fetch_log_path**  
*Fetches the Path to the Log File*

Description

Use this to get the path to the formods log file

Usage

`FM_fetch_log_path(state)`

Arguments

- `state`  
  module state after yaml read

Value

Character string with the path to the log file.
Examples

```r
# Within shiny a session variable will exist,
# this creates one here for testing purposes:
sess_res = UD_test_mksession(session=list())
session = sess_res$session
# This function assumes that some module state exists:
state = UD_init_state(
  FM_yaml_file = system.file(package = "formods",
                             "templates",
                             "formods.yaml"),
  MOD_yaml_file = system.file(package = "formods",
                             "templates",
                             "UD.yaml"),
  id = "UD",
  session = session)
FM_fetch_log_path(state)
```

---

**FM_fetch_mdl**  
**Fetches Models from Modules in the App**

Description

Loops through each specified module ID or all modules if no ID was specified. For each ID, an attempt will be made to extract any models available.

Usage

```r
FM_fetch_mdl(state, session, ids = NULL)
```

Arguments

- **state**  
  Current module state after yaml file has been read
- **session**  
  Shiny session variable
- **ids**  
  Vector of ID strings for the modules containing models or NULL for all modules with models available.

Value

list containing the current dataset with the following format:

- **isgood**: General logical indicator of successfully.
- **hasmdl**: Logical indicating if at least one model was found.
- **modules**: List of module checksums.
- **mdl**: Result of `MM_fetch_mdl`, see `vignette("making_modules", package = "formods")`
- **catalog**: Dataframe containing the a tabular catalog of the models found.
  - **label**: Text label for the model.
– object: Name of the object that contains the compiled rxode2 model.
– MOD_TYPE: Type of 'formods' module the model came from.
– id: Source ‘formods’ Module ID.
– checksum: Checksum of the module where the model came from.
– MDLchecksum: Checksum of the model.
– code: Code to generate the model.

Examples

```
# We need a module state and a Shiny session variable
# to use this function:
id="UD"
sess_res = UD_test_mksession(session=list(), id=id)
session = sess_res$session
state = sess_res$state
mdl = FM_fetch_mdl(state, session)
mdl$catalog
```

---

**FM_fetch_mod_state**

*Fetch the Module State*

**Description**

Fetches the module state from the userdata under the specified id

**Usage**

```
FM_fetch_mod_state(session, id)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>Shiny session variable.</td>
</tr>
<tr>
<td>id</td>
<td>ID string for the module.</td>
</tr>
</tbody>
</table>

**Value**

module state or NULL if it’s not defined.

**Examples**

```
# We need a Shiny session variable to use this function:
id="UD"
sess_res = UD_test_mksession(session=list(), id=id)
session = sess_res$session
state = sess_res$state
mdl = FM_fetch_mdl(state, session)
```

**FM_fetch_user_files_path**

*Fetches the Path to the User Files*

**Description**

Use this to get the path to the temporary directory where formods stores user files.

**Usage**

```r
FM_fetch_user_files_path(state)
```

**Arguments**

- `state`: module state after yaml read

**Value**

Character string with the path to the log file.

**Examples**

```r
# We need a state object to use this function:
id="UD"
.sess_res = UD_test_mksession(session=list(), id=id)
state = sess_res$state
user_dir = FM_fetch_user_files_path(state)
user_dir
```

---

**FM_generate_report**

*Generate Report*

**Description**

Generates a report from the states of the different modules. The type of report is based on the file extension of `file_name`.

**Usage**

```r
FM_generate_report(
    state, 
    session, 
    file_dir, 
    file_name, 
    ph = list(), 
    gen_code_only = FALSE, 
    rpterrors = TRUE
)
```
Arguments

state Module state requesting the report generation
session Shiny session variable
file_dir path to the location where the file should be written.
file_name base_filename (acceptable extensions are xlsx, docx, or pptx).
ph List containing placeholders used when generating Word documents (e.g., ph =
   list(HEADERRIGHT = "My text").
gen_code_only Boolean value indicating that only code should be generated (FALSE).
rpterrors Boolean variable to generate reports with errors.

Details

This function will look through the loaded modules and find those with reporting enabled. If
reporting is enabled it will look for reporting functions for that module. Reporting functions should
be of the following format (name and arguments):

XX_append_report(state, rpt, rpttype)

Where XX is the module short name. The state is the current state of the module. The rpt contains
the current content of the report. This will vary based on the report type:

- xlsx: List with two elements. The first is summary a data frame with two columns. The first
column is called Sheet_Name and the second column is called Description. This is a catalog
of sheets added to the report by the user and can be appended to using rbind. The second
element in xlsx rpt is another list with element names corresponding to the report sheet names
and the values corresponding to dataframes to be exported in the report.
- pptx or docx: Corresponding onbrand reporting object.

Value

List with the following elements

Examples

# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
if(interactive()){
  sess_res = FG_test_mksession(session=list(), full_session=FALSE)
session = sess_res$session
input = sess_res$input

  # This will create a populated FG state object:
  state = sess_res$state

  # This is the directory to write the report:
  file_dir = tempdir()

  # This is the file name that determines the type of report to write:
```r
file_name = "my_report.pptx"
#file_name = "my_report.docx"

rpt_res =
FM_generate_report(state = state,
        session = session,
        file_dir = file_dir,
        file_name = file_name,
        gen_code_only = TRUE,
        rpterrors = TRUE)

# This contains the exit status of the report generation
rpt_res$isgood

# This is the underlying code that was used to generate the report
cat(paste0(rpt_res$code, collapse="\n"))
}

---

**FM_init_state**

**Initialize a formods State Object**

**Description**

Initializes a formods state object with common elements.

**Usage**

```r
FM_init_state(
  FM_yaml_file,
  MOD_yaml_file,
  id,
  dep_mod_ids = c(),
  MT,
  button_counters,
  ui_ids,
  ui_hold,
  session
)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FM_yaml_file</td>
<td>App configuration file with FM as main section.</td>
</tr>
<tr>
<td>MOD_yaml_file</td>
<td>Module configuration file with MC as main section.</td>
</tr>
<tr>
<td>id</td>
<td>Shiny module ID.</td>
</tr>
<tr>
<td>dep_mod_ids</td>
<td>Vector of module ids this module depends on.</td>
</tr>
<tr>
<td>MT</td>
<td>Type of module using the short name (e.g. &quot;UD&quot;, &quot;FG&quot;, etc.).</td>
</tr>
</tbody>
</table>
button_counters

Vector of button UI elements that need to be tracked.

ui_ids

List of UI ids in the module.

ui_hold

Vector of UI elements that require holding.

session

Shiny session variable

Value

List with state initialized.

Examples

# Within shiny a session variable will exist,
# this creates examples here for testing purposes:
sess_res = UD_test_mksession(session=list())
session = sess_res$session
state = FM_init_state(
  FM_yaml_file = system.file(package = "formods",
  "templates",
  "formods.yaml"),
  MOD_yaml_file = system.file(package = "formods",
  "templates",
  "UD.yaml"),
  id = "UD",
  MT = "UD",
  button_counters = NULL,
  ui_ids = NULL,
  ui_hold = NULL,
  session = session)
state

FM_le

**Add Message to Log File and Displays it to the Console**

Description

Add the supplied txt and the module type to the log file and display it to the console.

Usage

FM_le(state, entry, escape_braces = TRUE, entry_type = "alert")

Arguments

- state: Module state after yaml read
- entry: Text to add
- escape_braces: Set to TRUE (default) to escape curly braces in the entry, set to FALSE to have the values interpreted.
- entry_type: Set to either "alert" (default), "danger", "info", "success", or "warning"
Value

Boolean value indicating success (TRUE) or failure (FALSE).

Examples

```r
# We need a module state to use this function:
id="UD"
sess_res = UD_test_mksession(session=list(), id=id)
state = sess_res$state
FM_le(state, "This is a normal message")
FM_le(state, "This is a danger message", entry_type="danger")
FM_le(state, "This is a info message", entry_type="info")
FM_le(state, "This is a success message", entry_type="success")
FM_le(state, "This is a warning message", entry_type="warning")
```

---

**FM_message**

*Show Message to User*

Description

Writes a message to the console depending on whether cli is installed or not.

Usage

```
FM_message(line, escape_braces = TRUE, entry_type = "alert")
```

Arguments

- **line**: Text to display
- **escape_braces**: Set to TRUE (default) to escape curly braces in the entry, set to FALSE to have the values interpreted.
- **entry_type**: Set to either "alert" (default), "danger", "info", "success", "warning", "h1", "h2", or "h3"

Value

Returns NULL

Examples

```
mr = FM_message("This is a normal message")
mr = FM_message("This is a danger message", entry_type="danger")
mr = FM_message("This is a info message", entry_type="info")
mr = FM_message("This is a success message", entry_type="success")
mr = FM_message("This is a warning message", entry_type="warning")
mr = FM_message("This is an H1 header", entry_type="h1")
mr = FM_message("This is an H2 header", entry_type="h2")
mr = FM_message("This is an H3 header", entry_type="h3")
```
**FM_mk_error_fig**  
*Generates 'ggplot' Object with Error Message*

**Description**

Takes a vector of messages and returns a ggplot object with the text in the figure. This can be used in automated figure generation to cascade an error message to the end user.

**Usage**

```r
FM_mk_error_fig(msgs)
```

**Arguments**

- `msgs` Vector of error messages

**Value**

A ggplot object

**Examples**

```r
FM_mk_error_fig("Oh nos! You've made a mistake!")
```

---

**FM_notify**  
*Shiny Notification*

**Description**

Generates a notification that should only show once.

**Usage**

```r
FM_notify(state, session)
```

**Arguments**

- `state` Module state generating the notification
- `session` Shiny session variable

**Value**

A boolean variable indicating if the notification was triggered
Examples

```r
if(interactive()){
library(formods)
library(shiny)
library(shinydashboard)
#https://fontawesome.com/icons?from=io

ui <- dashboardPage(
  skin="red",
  dashboardHeader(title="Test Notifications"),
  dashboardSidebar(
    sidebarMenu(
      menuItem("Notifications", tabName="example", icon=icon("table"))
    ),
  ),
  dashboardBody(
    tabItems(
      tabItem(tabName="example",
        fluidRow(
          shiny::actionButton("set_notification", "Set Notification"),
          shiny::textInput("user_text", label="Notify Text Here", value="Notify me"),
          shiny::actionButton("show_notification", "Show Notification")
        )
      )
    )
  )
)

# Main app server
server <- function(input, output, session) {

  # Need formods state object
  sess_res = UD_test_mksession(session, id="UD")

  # Captures input and sets the notification
  observeEvent(input$set_notification, {
    state = FM_fetch_mod_state(session, id="UD")
    state = FM_set_notification(state,
      notify_text = isolate(input$user_text),
      notify_id = "example")
    FM_set_mod_state(session, id="UD", state)
  })

  # Displays the notification
  observeEvent(input$show_notification, {
    state = FM_fetch_mod_state(session, id="UD")
    FM_notify(state, session)
  })
}
```
FM_pause_screen  Starts Modal Screen Pause

Description

Start a modal screen pause.

Usage

FM_pause_screen(state, session, message)

Arguments

state  Current module state after yaml file has been read.
session  Shiny session variable.
message  Optional message for the pause.

Value

Pauses the screen and has no return value.

Examples

# We need a module state object and Shiny session objects to use this function:
sess_res = UD_test_mksession(session=list())
session = sess_res$session
state = sess_res$state
FM_pause_screen(state, session)
FM_resume_screen(state, session)

FM_pretty_sort  Centralized Sorting Function

Description

When displaying information in a pull down this function can be used to sort those options.

Usage

FM_pretty_sort(unsrt_data)

Arguments

unsrt_data  Unsorted data.
Value

sorted data

Examples

# This is the full path to a test data file:
data_file_local = system.file(package="formods", "test_data", "TEST_DATA.xlsx")
# Excel files need a sheet specification:
sheet = "DATA"
# We will also attach the sheets along with it
df = readxl::read_excel(path=data_file_local, sheet=sheet)
# Regular sorting:
sort(unique(df$Cohort))
FM_pretty_sort(unique(df$Cohort))

FM_proc_include  Sets Message in State from UI Processing

Description

Any errors that need to be passed back to the user can be set with this function.

Usage

FM_proc_include(state, session)

Arguments

state formods State object.
session Shiny session variable.

Value

No return value, sets message in supplied session variable.

Examples

# We need a module state object to use this function:
id="UD"
sess_res = UD_test_mksession(session=list(), id=id)
state = sess_res$state
session = sess_res$session
FM_proc_include(state, session)
FM_set_app_state

**Description**

Sets the App State

**Usage**

```
FM_set_app_state(session, app_state, set_holds = TRUE)
```

**Arguments**

- `session` Shiny session variable.
- `app_state` Loaded app state.
- `set_holds` If TRUE (default) the holds will be set for all of the modules present in the app state.

---

FM_resume_screen

**Description**

Stops Modal Screen Pause

**Usage**

```
FM_resume_screen(state, session)
```

**Arguments**

- `state` Current module state after yaml file has been read.
- `session` Shiny session variable.

**Value**

No return value, called to disable screen pause.

**Examples**

```r
# We need a module state object and Shiny session objects to use this function:
sess_res = UD_test_mksession(session=list())
session = sess_res$session
state = sess_res$state
FM_pause_screen(state, session)
FM_resume_screen(state, session)
```
Value

No return value, just updates the app state in the session variable.

Examples

# We need a Shiny session object to use this function:
id="UD"
sess_res = UD_test_mksession(session=list(), id=id)
session = sess_res$session
app_state = FM_fetch_app_state(session)
FM_set_app_state(session, app_state)

**FM_set_mod_state**  
*Set the Module State*

Description

Sets the module state from the userdata under the specified id

Usage

FM_set_mod_state(session, id, state)

Arguments

- **session**  
  Shiny session variable

- **id**  
  ID string for the module.

- **state**  
  Module state to set.

Value

Session variable with the module state set.

Examples

# We need a Shiny session variable and a module state
# object to use this function:
id="UD"
sess_res = UD_test_mksession(session=list(), id=id)
session = sess_res$session
state = sess_res$state
FM_set_mod_state(session, id, state)
**FM_set_notification**  

*Shiny Notification*

**Description**
Generates a notification that should only show once.

**Usage**

```r
FM_set_notification(state, notify_text, notify_id, type = "info")
```

**Arguments**

- **state**: Module state generating the notification
- **notify_text**: Text to go in the notification
- **notify_id**: Unique string for this notification
- **type**: - Can be either "success", "failure", "info" (default), or "warning"

**Value**

Module state with notification text set

**Examples**

```r
if(interactive()){
  library(formods)
  library(shiny)
  library(shinydashboard)
  #https://fontawesome.com/icons?from=io

  ui <- dashboardPage(
    skin="red",
    dashboardHeader(title="Test Notifications"),
    dashboardSidebar(
      sidebarMenu(
        menuItem("Notifications", tabName="example", icon=icon("table"))
      ),
    ),
    dashboardBody(
      tabItems(
        tabItem(tabName="example",
          fluidRow(
            shiny::actionButton("set_notification", "Set Notification"),
            shiny::textInput("user_text", label="Notify Text Here", value="Notify me"),
            shiny::actionButton("show_notification", "Show Notification")
          )
        )
      )
    )
  }
```

# Main app server
server <- function(input, output, session) {

  # Need formods state object
  sess_res = UD_test_mksession(session, id="UD")

  # Captures input and sets the notification
  observeEvent(input$set_notification, {
    state = FM_fetch_mod_state(session, id="UD")
    state = FM_set_notification(state,
      notify_text = isolate(input$user_text),
      notify_id = "example")
    FM_set_mod_state(session, id="UD", state)
  })

  # Displays the notification
  observeEvent(input$show_notification, {
    state = FM_fetch_mod_state(session, id="UD")
    FM_notify(state, session)
  })

  shinyApp(ui, server)
}

---

**FM_set_ui_msg**  
*Sets Message in State from UI Processing*

**Description**

Any errors that need to be passed back to the user can be set with this function.

**Usage**

```r
FM_set_ui_msg(state, msgs, append = FALSE)
```

**Arguments**

- **state**: formods State object.
- **msgs**: Character vector of messages.
- **append**: When **TRUE**, msgs will be appended to any current messages. When **FALSE** (default) msgs will replace any existing messages.
Value

state with ui message set.

Examples

# We need a module state object to use this function:
id="UD"

sess_res = UD_test_mksession(session=list(), id=id)
state = sess_res$state
state = FM_set_ui_msg(state, "Something happened.")

FM_tc Run Try/Catch and Process Results

Description

Attempts to execute the text in cmd. This is done in a try/catch environment to capture any errors.

Usage

FM_tc(cmd, tc_env, capture)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd</td>
<td>Character object containing the R command to evaluate in the try/catch block</td>
</tr>
<tr>
<td>tc_env</td>
<td>list of with names corresponding to object names and corresponding Values to</td>
</tr>
<tr>
<td></td>
<td>define in the try/catch environment</td>
</tr>
<tr>
<td>capture</td>
<td>Character vector of values to capture after the command is successfully captured</td>
</tr>
</tbody>
</table>

Value

list with the following fields:

- isgood: Boolean indicating the whether the evaluation was successful.
- error: If the evaluation failed this contains the error object.
- msgs: Character vector of messages and/or errors.
- capture: List with names of objects to be captured and values corresponding to those captured objects.

Examples

# Successful command
res_good = FM_tc("good_cmd=ls()", list(), c("good_cmd"))
res_good

# Failed command
res_bad = FM_tc("bad_cmd =not_a_command()", list(), c("bad_cmd"))
res_bad
**formods**

**formods: Shiny modules for common tasks.**

**Description**

Shiny apps can often make use of the same key elements, this package provides modules for common tasks (data upload, wrangling data, figure generation and saving the app state). These modules can react and interact as well as generate code to create reproducible analyses.

**Author(s)**

Maintainer: John Harrold <john.m.harrold@gmail.com> (ORCID)

**See Also**

https://formods.ubiquity.tools/

**formods_check**

**Checks 'formods' Dependencies**

**Description**

Looks at the suggested dependencies and checks to make sure

**Usage**

formods_check(VERBOSE = TRUE)

**Arguments**

verbose Logical indicating if messages should be displayed

**Value**

List with the following elements:

- all_found: Boolean indicating if all packages were found
- found_pkgs: Character vector of found packages
- missing_pkgs: Character vector of missing packages

**Examples**

fcres = formods_check()
has\_changed \hspace{1cm} \textit{Detect if a UI element has changed}

\textbf{Description}
Takes a UI element value and an older value and determines if it has been modified

\textbf{Usage}
\begin{verbatim}
has\_changed(ui\_val = NULL, old\_val = NULL, init\_value = c(""))
\end{verbatim}

\textbf{Arguments}
\begin{itemize}
\item \texttt{ui\_val} \hspace{1cm} Current value from the UI.
\item \texttt{old\_val} \hspace{1cm} Last value of of the element.
\item \texttt{init\_value} \hspace{1cm} Default value for reading in UI data when it has not been defined.
\end{itemize}

\textbf{Value}
Boolean result of the comparison

\textbf{Examples}
\begin{verbatim}
changed\_true = has\_changed(ui\_val = "a", old\_val = ")
changed\_false = has\_changed(ui\_val = "a", old\_val = "a")
\end{verbatim}

has\_updated \hspace{1cm} \textit{Detect if a UI element has updated}

\textbf{Description}
Takes a UI element value and an older value and determines if it has been modified

\textbf{Usage}
\begin{verbatim}
has\_updated(ui\_val = NULL, old\_val = NULL)
\end{verbatim}

\textbf{Arguments}
\begin{itemize}
\item \texttt{ui\_val} \hspace{1cm} Current value from the UI.
\item \texttt{old\_val} \hspace{1cm} Last value of of the element. defined.
Value

Boolean result of the comparison

Examples

```r
classified_true  = has_updated(ui_val = "a", old_val = "")
classified_true

classified_false = has_updated(ui_val = "a", old_val = "a")
classified_false
```

Description

Creates a link to a Shiny icon

Usage

```r
icon_link(href, target = "_blank", icon_name = "circle-info")
```

Arguments

- **href**: URL to link to.
- **target**: New tab name.
- **icon_name**: Name of icon to use (argument to shiny::icon, default: "circle-info")

Value

A list with a shiny.tag class that can be converted into an HTML string via as.character() and saved to a file with save_html(). Note if href is NULL then NULL is returned.

Examples

```r
icon_link(href="https://formods.ubiquity.tools")
```
is_installed  

Determines if a Package is Installed

Description

Determines if the specified package is installed.

Usage

is_installed(pkgname)

Arguments

pkgname  
Name of package

Value

Logical indicating if the packages is installed or not

Examples

# This package should exist
is_installed('digest')

# This package should not exist
is_installed('bad package name')

linspace  

Implementation of the linspace Function from Matlab

Description

Creates a vector of n elements equally spaced apart.

Usage

linspace(a, b, n = 100)

Arguments

a  
initial number

b  
final number

n  
number of elements (integer >= 2)

Value

vector of numbers from a to b with n linearly spaced apart
**new_module_template**

**Description**

If you want to create a new formods module this function will create the template files for you.

**Usage**

```r
new_module_template(
  SN = "NM",
  Module_Name = "New Module",
  package = "pkgname",
  element = "analysis",
  file_dir = tempdir()
)
```

**Arguments**

- **SN**: Module short name
- **Module_Name**: Module long name
- **package**: Name of package that will contain the module
- **element**: What you would call the thing the module provides for example the FG module provides "figures", the DW module provides "data views".
- **file_dir**: Directory to save file

**Value**

list with the following elements:

- `mc`: Module components.
- `yaml`: Yaml configuration file.

Each of these is a list with paths to the respective files:

- `source`: Template source.
- `dest`: Destination file name.
- `dest_full`: Full path to the destination file name.

**Examples**

```r
new_module_template()
```
remove_hold

Removes Hold on UI Element

Description

When some buttons are clicked they will change the state of the system, but other UI components will not detect that change correctly. So those triggers are put on hold. This will remove the hold after those UI components have updated.

Usage

remove_hold(state, session, inputId)

Arguments

state module state with all of the current UI elements populated
session Shiny session variable
inputId The input ID of the UI element that was put on hold

Value

No return value, called to remove holds.

Examples

# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
sess_res = DW_test_mksession(session=list())
session = sess_res$session
input = sess_res$input

# For this example we also need a state variable
state = sess_res$state

# This sets a hold on the specified inputID. This is normally done in
# your XX_fetch_state() function.
state = set_hold(state, inputId = "select_dw_views")

# This will fetch the hold status of the specified inputID.
fetch_hold(state, inputId = "select_dw_views")

# This will remove the hold and is normally done in one of the UI outputs
# with a priority set to ensure it happens after the rest of the UI has
# refreshed.
state = remove_hold(state, session, inputId = "select_dw_views")
set_hold

Sets Hold on One or All UI Elements

Description

When some buttons are clicked they will change the state of the system, but other UI components will not detect that change correctly. So those triggers are put on hold. This will set the hold for a specified inputId or all ids if that value is set to NULL.

Usage

```
set_hold(state, inputId = NULL)
```

Arguments

- `state`: module state with all of the current ui elements populated
- `inputId`: The input ID of the UI element that was put on hold or NULL to hold all IDs in the module

Value

state with hold or holds set

Examples

```
# Within shiny both session and input variables will exist, 
# this creates examples here for testing purposes:
sess_res = DW_test_mksession(session=list())
session = sess_res$session
input = sess_res$input

# For this example we also need a state variable
state = sess_res$state

# This sets a hold on the specified inputID. This is normally done in 
# your XX_fetch_state() function.
state = set_hold(state, inputId = "select_dw_views")

# This will fetch the hold status of the specified inputID.
fetch Hold(state, inputId = "select_dw_views")

# This will remove the hold and is normally done in one of the UI outputs 
# with a priority set to ensure it happens after the rest of the UI has 
# refreshed.
state = remove Hold(state, session, inputId = "select_dw_views")
```
### Description

Attaches a dataset to the UD state supplied.

### Usage

```r
UD_attach_ds(
  state,
  clean = NULL,
  isgood = TRUE,
  load_msg = NULL,
  data_file_local = NULL,
  data_file_ext = NULL,
  data_file = NULL,
  sheet = NULL,
  sheets = NULL,
  code = "",
  object_name = NULL,
  contents = NULL
)
```

### Arguments

- **state**: UD state module.
- **clean**: Boolean switch to determine if the headers in the loaded dataset was cleaned.
- **isgood**: Boolean object indicating if the file was successfully loaded.
- **load_msg**: Text message indicated the success or any problems encountered when uploading the file.
- **data_file_local**: Full path to the data file on the server.
- **data_file_ext**: File extension of the uploaded file.
- **data_file**: Dataset file name without the path.
- **sheet**: If the uploaded file is an excel file, this is the currently selected sheet.
- **sheets**: If the uploaded file is an excel file, this is a character vector of the sheets present in that file.
- **code**: Code to load dataset.
- **object_name**: Name of the dataset object created when code is evaluated.
- **contents**: Data frame containing the contents of the data file.

### Value

state with data set attached
Examples

# We need a module state object to use this function:
id="UD"
sess_res = UD_test_mksession(session=list())
state = sess_res$state

# This is the full path to a test data file:
data_file_local = system.file(package="formods", "test_data", "TEST_DATA.xlsx")

# Excel file extension
data_file_ext = "xlsx"

# Base file name
data_file = "TEST_DATA.xlsx"

# Excel files need a sheet specification:
sheet = "DATA"

# We will also attach the sheets along with it
sheets = readxl::excel_sheets(data_file_local)

ds_read_res = UD_ds_read(state,
data_file_ext = data_file_ext,
data_file_local = data_file_local,
data_file = data_file,
sheets = sheets,
sheet = sheet)

# This would contain the loading code that will cascade down
# to the other modules when generating snippets and
# reproducible scripts
code = ds_read_res$code

# This is the R Object name that is used internally
# and in generated scripts. Should be the same as in
# the code above
object_name = ds_read_res$object_name

# This is the actual dataset:
contents = ds_read_res$contents

state = UD_attach_ds(
    state,
data_file_local = data_file_local,
data_file_ext = ".xlsx",
data_file = data_file,
sheet = sheet,
sheets = sheets,
code = code,
object_name = object_name,
contents = contents)
Generate Code and Load DS

Description
Generates the code for loading a dataset and returns both the code and the contents

Usage

```
UD_ds_read(
  state,
  data_file_ext = NULL,
  data_file_local = NULL,
  data_file = NULL,
  sheets = NULL,
  sheet = NULL
)
```

Arguments

- **state**: UD state from `UD_fetch_state()`
- **data_file_ext**: File extension of the uploaded file (e.g. "xlsx", "csv", etc).
- **data_file_local**: Full path to the data file on the server.
- **data_file**: Dataset file name without the path.
- **sheets**: If the uploaded file is an excel file, this is a character vector of the sheets present in that file.
- **sheet**: If the uploaded file is an excel file, this is the currently selected sheet.

Value
list with the elements of the dataset (contents, object_name, code, and isgood)

Examples

```
# We need a module state object to use this function:
id="UD"
sess_res = UD_test_mksession(session=list())
state = sess_res$state

# This is the full path to a test data file:
data_file_local = system.file(package="formods", "test_data", "TEST_DATA.xlsx")

# Excel file extension
data_file_ext = "xlsx"
```
```r
# Base file name
data_file = "TEST_DATA.xlsx"

# Excel files need a sheet specification:
sheet = "DATA"

# We will also attach the sheets along with it
sheets = readxl::excel_sheets(data_file_local)

ds_read_res = UD_ds_read(state,
data_file_ext = data_file_ext,
data_file_local = data_file_local,
data_file = data_file,
sheets = sheets,
sheet = sheet)

ds_read_res
```

### UD_fetch_code

**Fetch Module Code**

#### Description
Fetches the code to generate results seen in the app

#### Usage
```
UD_fetch_code(state)
```

#### Arguments
- `state` UD state from `UD_fetch_state()`

#### Value
Character object vector with the lines of code

#### Examples
```
# This creates a session variable that will be available in Shiny
state = UD_test_mksession(session=list())$state
UD_fetch_code(state)
```
**Fetch Module Datasets**

**Description**

Fetches the datasets contained in the module.

**Usage**

```r
UD_fetch_ds(state)
```

**Arguments**

- **state**
  
  UD state from `UD_fetch_state()`

**Value**

Character object vector with the lines of code list containing the following elements

- **isgood**: Return status of the function.
- **hasds**: Boolean indicator if the module has any datasets
- **msgs**: Messages to be passed back to the user.
- **ds**: List with datasets. Each list element has the name of the R-object for that dataset. Each element has the following structure:
  
  - **label**: Text label for the dataset
  - **MOD_TYPE**: Short name for the type of module.
  - **id**: module ID
  - **DS**: Dataframe containing the actual dataset.
  - **DSMETA**: Metadata describing DS, see `FM_fetch_ds()` for details on the format.
  - **code**: Complete code to build dataset.
  - **checksum**: Module checksum.
  - **DSchecksum**: Dataset checksum.

**Examples**

```r
# YAML configuration files from the package:
FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml")
MOD_yaml_file = system.file(package = "formods", "templates", "UD.yaml")

# This is the module id:
id = "UD"

# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
sess_res = UD_test_mksession(session=list())
session = sess_res$session
input = sess_res$input
```
state = UD_fetch_state(
    id = id,
    input = input,
    session = session,
    FM_yaml_file = FM_yaml_file,
    MOD_yaml_file = MOD_yaml_file
)

ds_res = UD_fetch_ds(state)

---

**Description**

Merges default app options with the changes made in the UI

**Usage**

`UD_fetch_state(id, id_ASM, input, session, FM_yaml_file, MOD_yaml_file)`

**Arguments**

- **id**  
  Shiny module ID
- **id_ASM**  
  ID string for the app state management module used to save and load app states
- **input**  
  Shiny input variable
- **session**  
  Shiny session variable
- **FM_yaml_file**  
  App configuration file with FM as main section.
- **MOD_yaml_file**  
  Module configuration file with MC as main section.

**Value**

List containing the current state of the app including default values from the yaml file as well as any changes made by the user. The list has the following structure:

- **yaml**: Full contents of the supplied yaml file.
- **MC**: Module components of the yaml file.
- **DS**: Loaded dataset with the following elements
  - **isgood**: Boolean object indicating if the file was successfully loaded.
  - **load_msg**: Text message indicated the success or any problems encountered when uploading the file.
  - **data_file_local**: Full path to the data file on the server.
  - **data_file**: Dataset file name without the path.
  - **data_file_ext**: File extension of the uploaded file.
  - **sheet**: If the uploaded file is an excel file, this is the currently selected sheet.
UD_init_state

Description

Creates a list of the initialized module state

Usage

UD_init_state(FM_yaml_file, MOD_yaml_file, id, session)

Arguments

FM_yaml_file  App configuration file with FM as main section.
MOD_yaml_file  Module configuration file with MC as main section.
id  ID string for the module.
session  Shiny session variable
UD_Server

Value

list containing an empty UD state

Examples

# Within shiny a session variable will exist,
# this creates one here for testing purposes:
 sess_res = UD_test_mksession(session=list())
 session = sess_res$session
 state = UD_init_state(
   FM_yaml_file = system.file(package = "formods",
                              "templates",
                              "formods.yaml"),
   MOD_yaml_file = system.file(package = "formods",
                               "templates",
                               "UD.yaml"),
   id = "UD",
   session = session)
 state

UD_Server Data Upload Server

Description

Server function for the Data Upload Shiny Module

Usage

UD_Server(  
id,  
id_ASM = "ASM",  
FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml"),  
MOD_yaml_file = system.file(package = "formods", "templates", "UD.yaml"),  
deployed = FALSE,  
react_state = NULL  )

Arguments

id An ID string that corresponds with the ID used to call the modules UI elements
id_ASM ID string for the app state management module used to save and load app states
FM_yaml_file App configuration file with FM as main section.
MOD_yaml_file Module configuration file with MC as main section.
deployed Boolean variable indicating whether the app is deployed or not.
react_state Variable passed to server to allow reaction outside of module (NULL)
Value

UD Server object

Examples

if(interactive()){
    # These are suggested packages
    library(shinydashboard)
    library(ggpubr)
    library(plotly)
    library(shinybusy)
    library(prompter)
    library(utilities)
    library(plotly)
    library(formods)

    CSS <- "
    .wrapfig {
        float: right;
        shape-margin: 20px;
        margin-right: 20px;
        margin-bottom: 20px;
    }
   "

    # Default to not deployed
    if(!exists("deployed")){
        deployed = FALSE
    }

    #https://fontawesome.com/icons?from=io

    ui <- dashboardPage(
        skin="black",
        dashboardHeader(title="formods"),
        dashboardSidebar(
            sidebarMenu(
                menuItem("Source Data", tabName="upload", icon=icon("table")),
                menuItem("Wrangle", tabName="wrangle", icon=icon("hat-cowboy")),
                menuItem("Plot", tabName="plot", icon=icon("chart-line")),
                menuItem("App State", tabName="app_state", icon=icon("archive")),
                menuItem("App Info", tabName="sysinfo", icon=icon("book-medical"))
            ),
            dashboardBody(
                tags$head(
                    tags$style(HTML(CSS))
                ),
                tabItems(
                    tabItem(tabName="app_state",
                )
            )
        )
    )
}
`box(title="Manage App State",
   htmlOutput(NS("ASM", "ui_asm_compact"))))

`tabItem(tabName="upload",
   box(title="Load Data", width=12,
   fluidRow(
      prompter::use_prompt(),
      column(width=6,
         htmlOutput(NS("UD", "UD_ui_compact"))),
      column(width=6,
         tags$p(
            tags$img(
               class = "wrapfig",
               width = 100,
               alt = "formods logo" ),
            'Formods is a set of modules and an framework for developing modules
   which interact and create code to replicate analyses performed within an app.
   To experiment download this',
            tags$a("test dataset", href=data_url),
            'and upload it into the App using the form on the left.'))
   )
   ),

`tabItem(tabName="wrangle",
   box(title="Transform and Create Views of Your Data", width=12,
   htmlOutput(NS("DW", "DW_ui_compact"))))

`tabItem(tabName="plot",
   box(title="Visualize Data", width=12,
   htmlOutput(NS("FG", "FG_ui_compact"))))

`tabItem(tabName="sysinfo",
   box(title="System Details", width=12,
   shinydashboard::tabBox(
      width = 12,
      title = NULL,
      shiny::tabPanel(id="sys_modules",
         title=tagList(shiny::icon("ghost"),
                        "Modules"),
         htmlOutput(NS("ASM", "ui_asm_sys_modules"))
      ),
      shiny::tabPanel(id="sys_packages",
         title=tagList(shiny::icon("ghost"),
                        "Packages"),
         htmlOutput(NS("ASM", "ui_asm_sys_packages"))
      ),
      shiny::tabPanel(id="sys_log",
         title=tagList(shiny::icon("clipboard-list"),
                        "App Log"),
         verbatimTextOutput(NS("ASM", "ui_asm_sys_log"))
      ),
      shiny::tabPanel(id="sys_options",
         title=tagList(shiny::icon("sliders"),
                        "R Options"),
         htmlOutput(NS("ASM", "ui_asm_sys_options"))
      ),
      shiny::tabPanel(id="sys_users",
         title=tagList(shiny::icon("user")
                        "Users"),
         htmlOutput(NS("ASM", "ui_asm_sys_users"))
      ))
   )
))

`
# Main app server
server <- function(input, output, session) {
  # Empty reactive object to track and react to
  # changes in the module state outside of the module
  react_FM = reactiveValues()

  # This is the list of module ids used for reproducible script generation. The
  # order here is important.
  mod_ids = c("UD", "DW", "FG")

  #Populating with test data
  # FG_test_mksession(session)
  # Module servers
  formods::ASM_Server(id="ASM",
    deployed = deployed,
    react_state = react_FM, mod_ids = mod_ids)
  formods::UD_Server(id="UD", id_ASM = "ASM",
    deployed = deployed,
    react_state = react_FM)
  formods::DW_Server(id="DW", id_ASM = "ASM", id_UD = "UD",
    deployed = deployed,
    react_state = react_FM)
  formods::FG_Server(id="FG", id_ASM = "ASM", id_UD = "UD", id_DW = "DW",
    deployed = deployed,
    react_state = react_FM)
}

shinyApp(ui, server)

---

**UD_test_mksession**

**Populate Session Data for Module Testing**

**Description**

Populates the supplied session variable for testing.

**Usage**

`UD_test_mksession(session, id = "UD")`
unfactor

Arguments

  session Shiny session variable (in app) or a list (outside of app)
  id An ID string that corresponds with the ID used to call the modules UI elements

Value

  list with the following elements

    • isgood: Boolean indicating the exit status of the function.
    • session: The value Shiny session variable (in app) or a list (outside of app) after initialization.
    • input: The value of the shiny input at the end of the session initialization.
    • state: App state.
    • rsc: The react_state components.

Examples

  res = UD_test_mksession(session=list())

unfactor Remove Factor From Object

Description

  Takes an object that is a factor and returns an unfactored vector with the same type by the value removed

Usage

  unfactor(fctobj)

Arguments

  fctobj Factorized object

Value

  Object with factors removed
Examples

```r
df = data.frame(
    text = c("a", "b", "c"),
    float = c(1, 2, 3))

df$float = as.factor(df$float)
# This is a factor
df$float
# This is not a factor
unfactor(df$float)
```

use_formods  

Create Module Templates in a Package Repository

Description

If you are developing a package within a repository (i.e. git) and want to create a new formods module this function will create the template files for you and install them in the correct location.

Usage

```r
use_formods(
    SN = "NM",
    Module_Name = "New Module",
    package = "pkgname",
    element = "analysis",
    overwrite = FALSE,
    repo_root = NULL
)
```

Arguments

- **SN**  
  Module short name
- **Module_Name**  
  Module long name
- **package**  
  Name of package that will contain the module
- **element**  
  What you would call the thing the module provides for example the FG module provides "figures", the DW module provides "data views"
- **overwrite**  
  Boolean to indicate if you should overwrite files
- **repo_root**  
  Root of the repository.

Value

Same as the return value for new_module_template()
use_formods

Examples

if(FALSE){
    use_formods(repo_root=tempdir())
}

Index

ASM_fetch_code, 4
ASM_fetch_dllfn, 4
ASM_fetch_state, 5
ASM_init_state, 6
ASM_Server, 7
ASM_test_mksession, 10
ASM_write_state, 11
autocast, 12

DW_add_wrangling_element, 14
DW_append_report, 15
DW_attach_ds, 16
dw_eval_element, 17
DW_fetch_code, 18
DW_fetch_current_view, 19
DW_fetch_ds, 20
DW_fetch_state, 21
DW_init_state, 23
DW_new_view, 24
DW_Server, 25
DW_set_current_view, 28
DW_test_mksession, 29
DW_update_checksum, 30
dwrs_builder, 13
fers_builder, 31
fetch_hold, 31
fetch_package_version, 32
FG_append_report, 33
FG_build, 34
FG_extract_page, 35
FG_fetch_code, 37
FG_fetch_current_fig, 37
FG_fetch_state, 38
FG_init_state, 41
FG_new_fig, 42
FG_Server, 43
FG_set_current_fig, 46
FG_test_mksession, 47
FG_update_checksum, 48

FM_add_ui_tooltip, 49
FM_build_comment, 50
FM_fetch_app_code, 51
FM_fetch_app_info, 51
FM_fetch_app_state, 52
FM_fetch_current_mods, 53
FM_fetch_data_format, 53
FM_fetch_deps, 54
FM_fetch_ds, 55
FM_fetch_log_path, 56
FM_fetch_mdl, 57
FM_fetch_mod_state, 58
FM_fetch_user_files_path, 59
FM_generate_report, 15, 33, 59
FM_init_state, 61
FM_le, 62
FM_message, 63
FM_mk_error_fig, 64
FM_notify, 64
FM_pause_screen, 66
FM_pretty_sort, 66
FM_proc_include, 67
FM_resume_screen, 68
FM_set_app_state, 68
FM_set_mod_state, 69
FM_set_notification, 70
FM_set_ui_msg, 71
FM_tc, 72
formods, 73
formods-package (formods), 73
formods_check, 73
has_changed, 74
has_updated, 74
icon_link, 75
is_installed, 76
linspace, 76
new_module_template, 77
INDEX

remove_hold, 78
set_hold, 79
UD_attach_ds, 80
UD_ds_read, 82
UD_fetch_code, 83
UD_fetch_ds, 84
UD_fetch_state, 85
UD_init_state, 86
UD_Server, 87
UD_test_mksession, 90
unfactor, 91
use_formods, 92