

Package: formods (via r-universe)

October 18, 2026

Type Package

Title 'Shiny' Modules for General Tasks

Version 0.3.0

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)

LazyData false

Imports cli, digest, dplyr, ggforce, ggplot2, onbrand (>= 1.0.8), readxl, rhandsontable, rlang, rio, shiny, shinyAce, shinyWidgets (>= 0.9.0), stats, stringr, tools, yaml, zip

Suggests clipr, covr, devtools, DT, flextable, ggpubr, gtools, here, janitor, knitr, plotly (>= 4.11.0), prompter, rmarkdown, shinybusy, shinydashboard, shinytest2, testthat (>= 3.0.0), utils, zoo

RoxygenNote 7.3.3

VignetteBuilder knitr

Config/testthat/edition 3

Config/pak/sysreqs libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev make libharfbuzz-dev libicu-dev libjpeg-dev libpng-dev libtiff-dev libuv1-dev libwebp-dev libxml2-dev libssl-dev libx11-dev zlib1g-dev

Repository <https://john-harrold.r-universe.dev>

Date/Publication 2026-02-11 01:17:58 UTC

RemoteUrl <https://github.com/john-harrold/formods>

RemoteRef HEAD

RemoteSha 668085f14bee4a778cd4add65f1e5445537f3428

Contents

ASM_check_workflow	5
ASM_fetch_code	6
ASM_fetch_dfn	7
ASM_fetch_state	7
ASM_init_state	8
ASM_load_state	9
ASM_mk_preload	10
ASM_onload	11
ASM_preload	11
ASM_read_app_state	12
ASM_run_workflow	13
ASM_save_state	14
ASM_Server	15
ASM_test_mksession	19
ASM_test_preload	20
autocast	21
DM_add_file	22
DM_add_url	23
DM_append_report	25
DM_del_current_element	26
DM_delete_source	27
DM_fetch_code	29
DM_fetch_current_element	30
DM_fetch_ds	31
DM_fetch_source	32
DM_fetch_state	34
DM_hads	35
DM_init_state	36
DM_mk_preload	37
DM_new_element	37
DM_onload	38
DM_preload	39
DM_run_code	40
DM_Server	42
DM_set_current_element	46
DM_test_mksession	47
DM_update_checksum	48
DM_update_element_code	49
DW_add_wrangling_element	50
DW_append_report	52

DW_attach_ds	53
dw_eval_element	54
DW_fetch_available_sources	55
DW_fetch_code	57
DW_fetch_current_view	58
DW_fetch_ds	59
DW_fetch_obj_ds_meta	60
DW_fetch_state	62
DW_fetch_view_deps	64
DW_hasds	65
DW_init_state	65
DW_mk_preload	66
DW_new_view	67
DW_preload	68
DW_proc_pll	69
DW_rebuild_current_view	70
DW_rectify	70
DW_Server	71
DW_set_current_view	76
DW_test_mksession	77
DW_update_checksum	78
dwrs_builder	78
fers_builder	80
fetch_hold	81
fetch_package_version	82
fetch_resource	82
FG_append_report	83
FG_build	84
FG_extract_page	86
FG_fetch_code	87
FG_fetch_current_fig	88
FG_fetch_state	89
FG_init_state	91
FG_mk_preload	92
FG_new_fig	92
FG_preload	93
FG_Server	94
FG_set_current_fig	98
FG_test_mksession	100
FG_update_checksum	100
FM_add_ui_tooltip	101
FM_app_preload	102
FM_build_comment	103
FM_fetch_app_code	103
FM_fetch_app_info	104
FM_fetch_app_state	105
FM_fetch_current_mods	105
FM_fetch_data_format	106

FM_fetch_deps	107
FM_fetch_ds	108
FM_fetch_log_path	109
FM_fetch_md1	110
FM_fetch_mod_state	111
FM_fetch_user_files_path	112
FM_generate_report	112
FM_has_ds_changed	114
FM_init_state	115
FM_le	116
FM_message	117
FM_mk_app_preload	117
FM_mk_error_fig	118
FM_notify	119
FM_pause_screen	120
FM_pretty_sort	121
FM_proc_include	122
FM_read_yaml	122
FM_reset_app	123
FM_resume_screen	123
FM_set_app_state	124
FM_set_mod_state	125
FM_set_notification	125
FM_set_ui_msg	127
FM_tc	128
formods	129
formods_check	129
has_changed	130
has_updated	130
icon_link	131
is_excel	132
is_importable	132
is_installed	133
is_shiny	133
linspace	134
mk_preload_dir	134
new_module_template	135
remove_hold	137
render_str	138
run_formods	138
set_hold	139
UD_attach_ds	140
UD_ds_read	142
UD_fetch_code	144
UD_fetch_ds	144
UD_fetch_state	145
UD_init_state	147
UD_mk_preload	148

<i>ASM_check_workflow</i>	5
UD_preload	148
UD_Server	149
UD_test_mksession	153
unfactor	154
use_formods	155
Index	157

ASM_check_workflow	<i>Checks Workflow Preload List Against Current App State</i>
--------------------	---

Description

Compares the resource dependencies in a workflow preload list to those currently available in the state to determine if the workflow can be run.

Usage

ASM_check_workflow(state, session, pll)

Arguments

state	ASM state from ASM_fetch_state()
session	Shiny session variable
pll	Workflow preload list.

Value

A list with the following attributes following elements:

- isgood: Boolean indicating the exit status of the function.
- msgs: Messages to be passed back to the user.
- chk_msgs: Results of check, this can contain "tags()"
- deps_found: Boolean indicating if all resources were found.
- dep_table: Table of dependency information with the following columns:
 - mod_ID: The formods module ID of the module that needs the resource.
 - dep_type : Type of resource dependency currently only "ds" for data source.
 - id : The formods module ID supplying the resource.
 - res_label: The resource label.
 - res_found: Logical indicating if the resource was found or not.

Examples

```
# The ASM session won't have any components so the check below should fail
sess_res = ASM_test_mksession()
session = sess_res[["session"]]
state = FM_fetch_mod_state(id="ASM", session=session)

# The DW test merge should require resource labels that are not currently present
pll = formods::FM_read_yaml(system.file(package="formods", "preload", "workflow_DW_merge.yaml"))

cwf_res = ASM_check_workflow(state=state, session=session, pll=pll)

cwf_res
```

ASM_fetch_code

Fetch Module Code

Description

Fetches 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

```
# Creating a state object for testing
sess_res = ASM_test_mksession()
state = sess_res$state
code = ASM_fetch_code(state)
```

ASM_fetch_dlfm	<i>Fetch Download File Name</i>
----------------	---------------------------------

Description

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

Usage

```
ASM_fetch_dlfm(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

Examples

```
# Creating a state object for testing
sess_res = ASM_test_mksession()
state = sess_res$state
dlfn = ASM_fetch_dlfm(state)
dlfn
```

ASM_fetch_state	<i>Fetch State Manager State</i>
-----------------	----------------------------------

Description

Merges default app options with the changes made in the UI

Usage

```
ASM_fetch_state(id, input, session, FM_yaml_file, MOD_yaml_file, 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.
react_state	Variable passed to server to allow reaction outside of

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.
- 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.

Examples

```
# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
sess_res = ASM_test_mksession()
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,
                        react_state = react_state,
                        FM_yaml_file = FM_yaml_file,
                        MOD_yaml_file = MOD_yaml_file)

state
```

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

Examples

```
# Within shiny the session variable will exist,
# this creates an example here for testing purposes:
sess_res = ASM_test_mksession()
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_load_state	<i>Load App State</i>
----------------	-----------------------

Description

Used to load the saved app state from a zip file

Usage

```
ASM_load_state(state, session, file_path)
```

Arguments

state	ASM state from ASM_fetch_state()
session	Shiny session variable
file_path	Zip file with the saved sate

Value

This function overwrites the current app state and returns a list with the following elements:

- isgood: Boolean indicating the exit status of the function.
- msgs: Messages to be passed back to the user.
- state: AMS state
- session: Session object
- react_state: Reaction state initialized after loading

Examples

```
# Within shiny the session variable will exist
# this creates examples here for testing purposes:
sess_res = ASM_test_mksession()
session = sess_res$session
state    = sess_res$state

ssf = tempfile(fileext=".zip")

ss_res =
  ASM_save_state(state, session,
                file_path = ssf)

ls_res =
  ASM_load_state(state, session,
                file_path = ssf)
```

ASM_mk_preload

Make List of Current ASM State

Description

Converts the current ASM state into a preload list.

Usage

```
ASM_mk_preload(state)
```

Arguments

state ASM state object

Value

list with the following elements

- isgood: Boolean indicating the exit status of the function.
- msgs: Messages to be passed back to the user.
- yaml_list: Lists with preload components.

Examples

```

sess_res = ASM_test_mksession()
state = sess_res$state
res = ASM_mk_preload(state)

```

ASM_onload

Updates ASM After State Load

Description

Creates a list of the initialized module state

Usage

```
ASM_onload(state, session)
```

Arguments

state	ASM state object
session	Shiny session variable

Value

List containing the ASM state objects and shiny session object

ASM_preload

Preload Data for ASM Module

Description

Populates the supplied session variable with information from list of sources.

Usage

```

ASM_preload(
  session,
  src_list,
  yaml_res = NULL,
  mod_ID = NULL,
  react_state = list(),
  quickload = FALSE
)

```

Arguments

session	Shiny session variable (in app) or a list (outside of app)
src_list	List of preload data (all read together with module IDs at the top level)
yaml_res	Result of reading in the formods (fm_cfg) and module (mod_cfg) yaml files
mod_ID	Module ID of the module being loaded.
react_state	Reactive shiny object (in app) or a list (outside of app) used to trigger reactions
quickload	Logical TRUE to load reduced analysis FALSE to load the full analysis

Value

list with the following elements

- isgood: Boolean indicating the exit status of the function.
- msgs: Messages to be passed back to the user.
- session: Session object
- input: The value of the shiny input at the end of the session initialization.
- state: App state.
- react_state: The react_state components.

ASM_read_app_state	<i>Read App State From Yaml Files</i>
--------------------	---------------------------------------

Description

Reads in the app state from yaml files.

Usage

```
ASM_read_app_state(sources = NULL)
```

Arguments

sources	Vector of at 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.
- msgs: Messages to be passed back to the user.
- src_list: List containing the result of reading all of the sources.
- yaml_res: Lists with elements for each module ID found in src_list with elements holding the modules configuration file "mod_cfg" and the modules formods configuration file "fm_cfg"

Examples

```
res = ASM_read_app_state(sources=system.file(package="formods", "preload", "UD_preload.yaml"))
```

ASM_run_workflow	<i>Run Specified Workflow</i>
------------------	-------------------------------

Description

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

Usage

```
ASM_run_workflow(state, session, wfl)
```

Arguments

state	ASM state from ASM_fetch_state()
session	Shiny session variable
wfl	List contining details about the workflow

Value

The ASM state object with the results stored as a list in the field `rwf_res` following elements:

- `isgood`: Boolean indicating the exit status of the function.
- `msgs`: Messages to be passed back to the user.
- `react_state`: Reaction state initilized after loading

Examples

```
# Populating the session with data
session = list()
ds_plf = c(system.file(package="formods", "preload", "ASM_preload_empty.yaml"),
           system.file(package="formods", "preload", "UD_preload.yaml"),
           system.file(package="formods", "preload", "DM_preload_empty.yaml"))
session = list()
res_ds = FM_app_preload(session=session, sources=ds_plf)

session = res_ds[["session"]]
state = res_ds[["all_sess_res"]][["ASM"]][["state"]]

# Creating the workflow preload list
wf_pll = c(
  FM_read_yaml( system.file(package="formods", "preload", "ASM_preload.yaml")),
  FM_read_yaml( system.file(package="formods", "preload", "UD_preload.yaml")),
  FM_read_yaml( system.file(package="formods", "preload", "DM_preload_empty.yaml")),
  FM_read_yaml( system.file(package="formods", "preload", "DW_preload_empty.yaml")))

tmp_preload = tempfile(fileext=".yaml")
yaml::write_yaml(file=tmp_preload, x=wf_pll)
```

```

wfl = list(
  require_ds = TRUE,
  preload    = tmp_preload
)

# Running the workflow
state = ASM_run_workflow(state=state, session=session, wfl=wfl)

state$ASM$rwf_res

```

ASM_save_state

Write State to File for Saving

Description

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

Usage

```
ASM_save_state(state, session, file_path, pll = NULL, update_modal = TRUE)
```

Arguments

state	ASM state from ASM_fetch_state()
session	Shiny session variable
file_path	File path to write zipped state
pll	Preload list of the format generated by FM_mk_app_preload(). IF set to NULL it will be generated from the contents of the session variable.
update_modal	Logical controlling updates to modal messages (TRUE)

Value

This function only writes the state and returns a list with the following elements:

- isgood: Boolean indicating the exit status of the function.
- msgs: Messages to be passed back to the user.

Examples

```

# Within shiny the session variable will exist
# this creates examples here for testing purposes:
sess_res = ASM_test_mksession()
session = sess_res$session
state    = sess_res$state

ssf = tempfile(fileext=".zip")

ss_res =
  ASM_save_state(state, session,
                 file_path = ssf)

```

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

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

Value

UD Server object

Examples

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

    CSS <- "
    .wrapfig {
```

```

float: right;
shape-margin: 20px;
margin-right: 20px;
margin-bottom: 20px;
}
"

formods.yaml = system.file(package="formods", "templates", "formods.yaml")
ASM.yaml     = system.file(package="formods", "templates", "ASM.yaml")
UD.yaml      = system.file(package="formods", "templates", "UD.yaml")
DW.yaml      = system.file(package="formods", "templates", "DW.yaml")
DM.yaml      = system.file(package="formods", "templates", "DM.yaml")
FG.yaml      = system.file(package="formods", "templates", "FG.yaml")

# Name of file to indicate we need to load testing data
ftmptest = file.path(tempdir(), "formods.test")

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

#https://fontawesome.com/icons?from=io
data_url =
"https://github.com/john-harrold/formods/raw/master/inst/test_data/TEST_DATA.xlsx"

ui <- dashboardPage(
  skin="black",
  dashboardHeader(title="formods"),
  dashboardSidebar(
    sidebarMenu(
      menuItem("Source Data",      tabName="upload",      icon=icon("table")),
      menuItem("Manage Data",      tabName="manage_data", icon=icon("folder-tree")),
      menuItem("Workflows",        tabName="workflows",  icon=icon("robot")),
      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",
src = "https://github.com/john-harrold/formods/raw/master/man/figures/logo.png",
    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="workflows",
  box(title="Run Workflows",
    htmlOutput(NS("ASM", "ASM_ui_workflows"))),
tabItem(tabName="wrangle",
  box(title="Transform and Create Views of Your Data", width=12,
    htmlOutput(NS("DW", "DW_ui_compact"))),
tabItem(tabName="manage_data",
  box(title="Manage and Create Data Sources", width=12,
    htmlOutput(NS("DM", "DM_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()

  #react_FM[["UD"]] = list()
  #react_FM[["FG"]] = list()
  #react_FM[["DW"]] = list()
  #react_FM[["DM"]] = list()
  #react_FM[["ASM"]] = list()

  #Uncomment to populate with test data
  # If the ftmpstest file is present we load test data
  if(file.exists(ftmpstest)){
    sources = c(system.file(package="formods", "preload", "ASM_preload.yaml"),
                 system.file(package="formods", "preload", "UD_preload.yaml"),
                 system.file(package="formods", "preload", "DM_preload.yaml"),
                 system.file(package="formods", "preload", "DW_preload.yaml"),
                 system.file(package="formods", "preload", "FG_preload.yaml"))
    res = FM_app_preload(session=session, sources=sources, react_state = react_FM)
  } else if(file.exists("preload.yaml")){

    shinybusy::show_modal_spinner(text="Preloading analysis, be patient", session=session)
    res = FM_app_preload(session=session, sources="preload.yaml", react_state=react_FM)
    shinybusy::remove_modal_spinner(session = session)
  }

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

  # Module servers
  formods::ASM_Server(id="ASM",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = ASM.yaml,
    deployed      = deployed,
    react_state   = react_FM,
    mod_ids       = mod_ids)
  formods::UD_Server( id="UD",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = UD.yaml,
    deployed      = deployed,

```

```

        react_state = react_FM)
formods::DW_Server( id="DW",
  FM_yaml_file = formods.yaml,
  MOD_yaml_file = DW.yaml,
  deployed     = deployed,
  react_state  = react_FM)
formods::DM_Server( id="DM",
  FM_yaml_file = formods.yaml,
  MOD_yaml_file = DM.yaml,
  deployed     = deployed,
  react_state  = react_FM)
formods::FG_Server( id="FG",
  FM_yaml_file = formods.yaml,
  MOD_yaml_file = FG.yaml,
  deployed     = deployed,
  react_state  = react_FM)
}

shinyApp(ui, server)
}
}

```

ASM_test_mksession	<i>Populate Session Data for Module Testing</i>
--------------------	---

Description

Populates the supplied session variable for testing.

Usage

```
ASM_test_mksession(session = list())
```

Arguments

session	Shiny session variable (in app) or a list (outside of app)
---------	--

Value

The ASM portion of the ‘all_sess_res’ returned from [FM_app_preload](#)

See Also

[FM_app_preload](#)

Examples

```

session = shiny::MockShinySession$new()
sess_res = ASM_test_mksession(session=session)

```

ASM_test_preload	<i>Test Specified Preload Functionality</i>
------------------	---

Description

Loads specified preload lists to verify execution, saves the state to the specified zip file and attempts to load the saved state if test_save_state is TRUE. To test a workflow you can use sources to specify the loading of data and then workflow to specify the yaml workflow.

Usage

```
ASM_test_preload(
  sources = NULL,
  preload_files = NULL,
  preload_dir = tempfile(pattern = "preload_"),
  save_state_file = tempfile(fileext = ".zip"),
  test_save_state = TRUE,
  workflow = NULL
)
```

Arguments

sources	Vector of at corresponds with the ID used to call the modules UI elements
preload_files	Dataframe of files needed to run the workflow with a column called src for the local source of the file and dest for the destination. Not strictly required but almost always needed.
preload_dir	Directory to run out of (tempdir())
save_state_file	When testing 'ASM_save_state()' this is the file name where the stat will be written (tempfile(fileext=".zip"))
test_save_state	Logical when set to TRUE (default) it will test the zip file (save_state_file) to make sure it can be loaded.
workflow	Yaml preload file to run a workflow (optional, NULL default).

Value

list with the following elements

- isgood: Boolean indicating the exit status of the function.
- msgs: Messages to be passed back to the user.
- apl_res: Results from FM_app_preload() if run, list(isgood=FALSE, msgs="not run") otherwise.
- ss_res: Results from ASM_save_state() if run, list(isgood=FALSE, msgs="not run") otherwise.

- `ls_res`: Results from `ASM_load_state()` if run, `list(isgood=FALSE, msgs="not run")` otherwise.
- `rwf_res`: Results from `ASM_run_workflow()` if run, `list(isgood=FALSE, msgs="not run")` otherwise.
- `save_state_file`: If `ss_res$isgood` is `TRUE` this is the path to the zip file generated when saving the app state after loading.

Examples

```
sources = c(
  system.file(package="formods", "preload", "UD_preload.yaml"),
  system.file(package="formods", "preload", "ASM_preload.yaml"))

res = ASM_test_preload(sources=sources, test_save_state = FALSE)

res$isgood
```

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

<code>ui_input</code>	UI input from a shiny form
<code>quote_char</code>	<code>TRUE</code> 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')
```

DM_add_file

*Make List of Current DM State***Description**

Adds a file to the DM State

Usage

```
DM_add_file(state, file_path, file_name, load_ID = NULL)
```

Arguments

state	DM state object
file_path	Location of file to add
file_name	Name of file
load_ID	When loading from a file the resource ID can be specified, set to NULL (default) have the ID auto generated.

Value

DM state with file added and the 'res\$add_file' field with the following elements:

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

Examples

```
# This creates a testing environment
sess_res = DM_test_mksession()
session = sess_res$session
input   = sess_res$input
state   = sess_res$state

# Next we delete all the elements that are currently defined
ele_ids = names(state[["DM"]][["elements"]])
while(state[["DM"]][["current_element"]] %in% ele_ids){
  state = suppressMessages(DM_del_current_element(state))
}

# This will delete all the pre-defined sources
sources_df = state[["DM"]][["defined_sources"]]
source_ids = sources_df$ID
for(source_id in source_ids){
  state = suppressMessages(DM_delete_source(state = state, id = source_id))
}

# Adding a file as a resource
```

```

DM_add_file(
  state      = state,
  file_name  = "TEST_DATA.xlsx",
  file_path  = system.file(package="formods", "test_data", "TEST_DATA.xlsx"),
  load_ID    = 10)

# Because CRAN won't let you have lines longer than 100 chars we have to
# build the string then pass it to the function below:
tmp_url      = paste0('https://raw.githubusercontent.com/john-harrold', '/',
                      'formods/refs/heads/master/inst/test_data/SDTM_DM.csv')
# Adding a url as a resource
DM_add_url(
  state      = state,
  url        = tmp_url,
  load_ID    = 11)

# Attaching a file to dataset element:
current_ele  = DM_fetch_current_element(state)
current_ele[["ui"]][["source_id"]] = "10"
current_ele[["ui"]][["ds_sheet"]]  = "DATA"

#This will pull out the source details for an element:
fsres = DM_fetch_source(state=state, element=current_ele)

# Rebuilding the code to attach the file:
current_ele = DM_update_element_code(state = state, element=current_ele, session)

# Running the attachment code:
current_ele = DM_run_code(state = state, element=current_ele)

# Saving the element:
DM_set_current_element(
  state      = state,
  element    = current_ele)

```

DM_add_url

Attaches Data Source from URL

Description

Using the user specified url a data source will be downloaded and attached (if possible).

Usage

```
DM_add_url(state, url = "", load_ID = NULL)
```

Arguments

state	DM state from DM_fetch_state()
url	URL to load
load_ID	When loading from a file the resource ID can be specified set to NULL (default) have the ID auto generated.

Value

DM state with url attachment results stored in the 'res\$add_url' field with the following elements:

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

If successful it will be added to the resources table.

Examples

```
# This creates a testing environment
sess_res = DM_test_mksession()
session = sess_res$session
input    = sess_res$input
state    = sess_res$state

# Next we delete all the elements that are currently defined
ele_ids = names(state[["DM"]][["elements"]])
while(state[["DM"]][["current_element"]] %in% ele_ids){
  state = suppressMessages(DM_del_current_element(state))
}

# This will delete all the pre-defined sources
sources_df = state[["DM"]][["defined_sources"]]
source_ids = sources_df$ID
for(source_id in source_ids){
  state = suppressMessages(DM_delete_source(state = state, id = source_id))
}

# Adding a file as a resource
DM_add_file(
  state      = state,
  file_name  = "TEST_DATA.xlsx",
  file_path  = system.file(package="formods", "test_data", "TEST_DATA.xlsx"),
  load_ID    = 10)

# Because CRAN won't let you have lines longer than 100 chars we have to
# build the string then pass it to the function below:
tmp_url      = paste0('https://raw.githubusercontent.com/john-harrold', '/',
                      'formods/refs/heads/master/inst/test_data/SDTM_DM.csv')

# Adding a url as a resource
DM_add_url(
  state      = state,
```

```

    url      = tmp_url,
    load_ID  = 11)

# Attaching a file to dataset element:
current_ele = DM_fetch_current_element(state)
current_ele[["ui"]][["source_id"]] = "10"
current_ele[["ui"]][["ds_sheet"]] = "DATA"

#This will pull out the source details for an element:
fsres = DM_fetch_source(state=state, element=current_ele)

# Rebuilding the code to attach the file:
current_ele = DM_update_element_code(state = state, element=current_ele, session)

# Running the attachment code:
current_ele = DM_run_code(state = state, element=current_ele)

# Saving the element:
DM_set_current_element(
  state = state,
  element = current_ele)

```

DM_append_report	<i>Append Report Elements</i>
------------------	-------------------------------

Description

Appends report elements to a formods report.

Usage

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

Arguments

state	DM state from <code>DM_fetch_state()</code>
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 template_details
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: 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

`formods::FM_generate_report()`

Examples

```
# We need a state object to use below
sess_res = DM_test_mksession()
state = sess_res$state

rpt = list(summary = list(), sheets=list())

rpt_res = DM_append_report(state,
  rpt      = rpt,
  rpttype = "xlsx")

# Shows if report elements are present
rpt_res$hasrptele

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

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

DM_del_current_element

Deletes Current ds

Description

Takes a DM state and deletes the current ds. If that is the last element, then a new default will be added.

Usage

```
DM_del_current_element(state)
```

Arguments

state DM state from DM_fetch_state()

Value

DM state object with the current ds deleted.

Examples

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

# Creates a new empty element
state = DM_new_element(state)

# Delete the current element
state = DM_del_current_element(state)

# Fetch a list of the current element
element = DM_fetch_current_element(state)

# You can modify the element
element[["name"]] = "A more descriptive name"

# You can now place element back in the state
state = DM_set_current_element(state, element)

# This will pull the portion of the code associated with this module.
code = DM_fetch_code(state)

cat(code)

# This forces and update of the module checksum
state = DM_update_checksum(state)
```

DM_delete_source	<i>Make List of Current DM State</i>
------------------	--------------------------------------

Description

Deletes a file based on the specified ID

Usage

```
DM_delete_source(state, id)
```

Arguments

state	DM state object
id	Location of file to add

Value

DM state with file deleted and the 'res\$del_file' field with the following elements:

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

Examples

```
# This creates a testing environment
sess_res = DM_test_mksession()
session = sess_res$session
input    = sess_res$input
state    = sess_res$state

# Next we delete all the elements that are currently defined
ele_ids = names(state[["DM"]][["elements"]])
while(state[["DM"]][["current_element"]] %in% ele_ids){
  state = suppressMessages(DM_del_current_element(state))
}

# This will delete all the pre-defined sources
sources_df = state[["DM"]][["defined_sources"]]
source_ids = sources_df$ID
for(source_id in source_ids){
  state = suppressMessages(DM_delete_source(state = state, id = source_id))
}

# Adding a file as a resource
DM_add_file(
  state      = state,
  file_name  = "TEST_DATA.xlsx",
  file_path  = system.file(package="formods", "test_data", "TEST_DATA.xlsx"),
  load_ID    = 10)

# Because CRAN won't let you have lines longer than 100 chars we have to
# build the string then pass it to the function below:
tmp_url      = paste0('https://raw.githubusercontent.com/john-harrold', '/',
                      'formods/refs/heads/master/inst/test_data/SDTM_DM.csv')
# Adding a url as a resource
DM_add_url(
  state      = state,
  url        = tmp_url,
  load_ID    = 11)

# Attaching a file to dataset element:
current_ele  = DM_fetch_current_element(state)
current_ele[["ui"]][["source_id"]] = "10"
current_ele[["ui"]][["ds_sheet"]]  = "DATA"

#This will pull out the source details for an element:
fsres = DM_fetch_source(state=state, element=current_ele)
```

```

# Rebuilding the code to attach the file:
current_ele = DM_update_element_code(state = state, element=current_ele, session)

# Running the attachment code:
current_ele = DM_run_code(state = state, element=current_ele)

# Saving the element:
DM_set_current_element(
  state = state,
  element = current_ele)

```

DM_fetch_code

Fetch Module Code

Description

Fetches the code to generate results seen in the app

Usage

```
DM_fetch_code(state)
```

Arguments

state DM state from DM_fetch_state()

Value

Character object vector with the lines of code

Examples

```

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

# Creates a new empty element
state = DM_new_element(state)

# Delete the current element
state = DM_del_current_element(state)

# Fetch a list of the current element
element = DM_fetch_current_element(state)

```

```
# You can modify the element
element[["name"]] = "A more descriptive name"

# You can now place element back in the state
state = DM_set_current_element(state, element)

# This will pull the portion of the code associated with this module.
code = DM_fetch_code(state)

cat(code)

# This forces and update of the module checksum
state = DM_update_checksum(state)
```

DM_fetch_current_element

Fetches Current ds

Description

Takes a DM state and returns the current active ds

Usage

```
DM_fetch_current_element(state)
```

Arguments

state DM state from DM_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\$DM\$elements in the output of DM_fetch_state().

Examples

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

# Creates a new empty element
state = DM_new_element(state)

# Delete the current element
state = DM_del_current_element(state)
```

```

# Fetch a list of the current element
element = DM_fetch_current_element(state)

# You can modify the element
element[["name"]] = "A more descriptive name"

# You can now place element back in the state
state = DM_set_current_element(state, element)

# This will pull the portion of the code associated with this module.
code = DM_fetch_code(state)

cat(code)

# This forces and update of the module checksum
state = DM_update_checksum(state)

```

DM_fetch_ds

Fetch Data Management Module Datasets

Description

Fetches the datasets contained in the module.

Usage

```
DM_fetch_ds(state, meta_only = FALSE)
```

Arguments

state	DM state from DM_fetch_state()
meta_only	Include only metadata and not the dataset (default FALSE)

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.

- idx: unique numerical ID to identify this dataset in the module.
- res_label: optional label that can be defined by a user and used in workflows. Must be unique to the module.
- DS: Dataframe containing the actual dataset.
- DSMETA: Metadata describing DS.
- code: Complete code to build dataset.
- checksum: Module checksum.
- DSchecksum: Dataset checksum.

Examples

```
# We need a module state:
sess_res = DM_test_mksession()
state = sess_res$state

ds = DM_fetch_ds(state)

ds
```

DM_fetch_source

Gets Source File Name for Element

Description

Gets file information about the current element

Usage

```
DM_fetch_source(state, element)
```

Arguments

state	DM state from DM_fetch_state()
element	Element list from DM_fetch_current_element()

Value

list with the following elements

- isgood: Boolean indicating if the file is good (if it's an excel file and has a sheet).
- msgs: Messages to be passed back to the user.
- excel_file: Logical indicating if the source is an excel file.
- excel_good: Logical indicating if the source is an excel file and we have a sheet defined in that file.
- file_name: Base file name (e.g. 'file.csv')
- full_path: Full path to the file (e.g. '/var/tmp/file.csv'), NULL if not found.

- rel_path: Path to the file relative to user directory file (e.g. 'data/DM/file.csv'), NULL if not found.
- sheet_found: Logical indicating if a sheet was found
- sheet: Excel sheet selected by user
- source_type: Type of source selected ("file", "url" or "unknown")

Examples

```
# This creates a testing environment
sess_res = DM_test_mksession()
session = sess_res$session
input    = sess_res$input
state    = sess_res$state

# Next we delete all the elements that are currently defined
ele_ids = names(state[["DM"]][["elements"]])
while(state[["DM"]][["current_element"]] %in% ele_ids){
  state = suppressMessages(DM_del_current_element(state))
}

# This will delete all the pre-defined sources
sources_df = state[["DM"]][["defined_sources"]]
source_ids = sources_df$ID
for(source_id in source_ids){
  state = suppressMessages(DM_delete_source(state = state, id = source_id))
}

# Adding a file as a resource
DM_add_file(
  state      = state,
  file_name  = "TEST_DATA.xlsx",
  file_path  = system.file(package="formods", "test_data", "TEST_DATA.xlsx"),
  load_ID    = 10)

# Because CRAN won't let you have lines longer than 100 chars we have to
# build the string then pass it to the function below:
tmp_url      = paste0('https://raw.githubusercontent.com/john-harrold', '/',
                      'formods/refs/heads/master/inst/test_data/SDTM_DM.csv')

# Adding a url as a resource
DM_add_url(
  state      = state,
  url        = tmp_url,
  load_ID    = 11)

# Attaching a file to dataset element:
current_ele  = DM_fetch_current_element(state)
current_ele[["ui"]][["source_id"]] = "10"
current_ele[["ui"]][["ds_sheet"]]  = "DATA"

#This will pull out the source details for an element:
fsres = DM_fetch_source(state=state, element=current_ele)
```

```
# Rebuilding the code to attach the file:
current_ele = DM_update_element_code(state = state, element=current_ele, session)

# Running the attachment code:
current_ele = DM_run_code(state = state, element=current_ele)

# Saving the element:
DM_set_current_element(
  state = state,
  element = current_ele)
```

DM_fetch_state	<i>Fetch Data Management State</i>
----------------	------------------------------------

Description

Merges default app options with the changes made in the UI

Usage

```
DM_fetch_state(id, input, session, FM_yaml_file, MOD_yaml_file, 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.
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 list has the following structure:

- yaml: Full contents of the supplied yaml file.
- MC: Module components of the yaml file.
- DM:
 - isgood: Boolean object indicating if the file was successfully loaded.
 - checksum: This is an MD5 sum of the contents element and can be used to detect changes in the state.
- MOD_TYPE: Character data containing the type of module "DM"

- 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

```
# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
sess_res = DM_test_mksession()
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", "DM.yaml")

# Creating an empty state object
state = DM_fetch_state(id           = "DM",
                       input        = input,
                       session       = session,
                       FM_yaml_file = FM_yaml_file,
                       MOD_yaml_file = MOD_yaml_file,
                       react_state  = NULL)
```

DM_hasds

Check DM State For Datasets

Description

Walks through the DM state object to see if there are any datasets available

Usage

```
DM_hasds(state)
```

Arguments

state DM state from DM_fetch_state()

Value

Logical TRUE if there is a dataset or FALSE otherwise.

Examples

```
sess_res = DM_test_mksession()
state = sess_res[["state"]]
DM_hasds(state)
```

DM_init_state	<i>Initialize DM Module State</i>
---------------	-----------------------------------

Description

Creates a list of the initialized module state

Usage

```
DM_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 DM state

Examples

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

state = DM_init_state(
  FM_yaml_file = system.file(package = "formods",
                              "templates",
                              "formods.yaml"),
  MOD_yaml_file = system.file(package = "formods",
                              "templates",
                              "DM.yaml"),
  id            = "DM",
  session      = session)

state
```

DM_mk_preload	<i>Make List of Current DM State</i>
---------------	--------------------------------------

Description

Reads in the app state from yaml files.

Usage

```
DM_mk_preload(state)
```

Arguments

state	DM state object
-------	-----------------

Value

list with the following elements

- isgood: Boolean indicating the exit status of the function.
- msgs: Messages to be passed back to the user.
- yaml_list: Lists with preload components.

Examples

```
sess_res = DM_test_mksession()  
state = sess_res$state  
res = DM_mk_preload(state)
```

DM_new_element	<i>New Data Management ds</i>
----------------	-------------------------------

Description

Appends a new empty ds to the DM state object and makes this new ds the active ds.

Usage

```
DM_new_element(state)
```

Arguments

state	DM state from DM_fetch_state()
-------	--------------------------------

Value

DM state object containing a new ds and that ds is set as the current active ds. See the help for `DM_fetch_state()` for ds format.

Examples

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

# Creates a new empty element
state = DM_new_element(state)

# Delete the current element
state = DM_del_current_element(state)

# Fetch a list of the current element
element = DM_fetch_current_element(state)

# You can modify the element
element[["name"]] = "A more descriptive name"

# You can now place element back in the state
state = DM_set_current_element(state, element)

# This will pull the portion of the code associated with this module.
code = DM_fetch_code(state)

cat(code)

# This forces and update of the module checksum
state = DM_update_checksum(state)
```

DM_onload

Processes State After Loading

Description

When loading a saved analysis, this will process the state object to account for differences that may apply between servers.

Usage

```
DM_onload(state, session)
```

Arguments

state	DM state from DM_fetch_state()
session	Shiny session variable

Value

List containing the DM state object and shiny session object

Examples

```
sess_res = DM_test_mksession()
session = sess_res$session
state = sess_res$state
state = DM_onload(state, session)
```

DM_preload	<i>Preload Data for DM Module</i>
------------	-----------------------------------

Description

Populates the supplied session variable with information from list of sources.

Usage

```
DM_preload(
  session,
  src_list,
  yaml_res,
  mod_ID = NULL,
  react_state = list(),
  quickload = FALSE
)
```

Arguments

session	Shiny session variable (in app) or a list (outside of app)
src_list	List of preload data (all read together with module IDs at the top level)
yaml_res	List data from module yaml config
mod_ID	Module ID of the module being loaded.
react_state	Reactive shiny object (in app) or a list (outside of app) used to trigger reactions.
quickload	Logical TRUE to load reduced analysis FALSE to load the full analysis

Value

list with the following elements

- isgood: Boolean indicating the exit status of the function.
- msgs: Messages to be passed back to the user.
- session: Session object
- input: The value of the shiny input at the end of the session initialization.
- state: App state.
- react_state: The react_state components.

DM_run_code	<i>Run Element Code</i>
-------------	-------------------------

Description

Runs the element code and updates the dataset

Usage

DM_run_code(state, element, session)

Arguments

state	DM state from DM_fetch_state()
element	Element list from DM_fetch_current_element()
session	Shiny session variable

Value

DM element with code updated

DM element with code updated and the ‘res\$run_code‘ field with the following elements:

- isgood: Return status of the function.
- msgs: Messages to be passed back to the user.
- ds: Dataset if reading was successful and NULL if not.

Examples

```

# This creates a testing environment
sess_res = DM_test_mksession()
session = sess_res$session
input    = sess_res$input
state    = sess_res$state

# Next we delete all the elements that are currently defined
ele_ids = names(state[["DM"]][["elements"]])
while(state[["DM"]][["current_element"]] %in% ele_ids){
  state = suppressMessages(DM_del_current_element(state))
}

# This will delete all the pre-defined sources
sources_df = state[["DM"]][["defined_sources"]]
source_ids = sources_df$ID
for(source_id in source_ids){
  state = suppressMessages(DM_delete_source(state = state, id = source_id))
}

# Adding a file as a resource
DM_add_file(
  state      = state,
  file_name  = "TEST_DATA.xlsx",
  file_path  = system.file(package="formods", "test_data", "TEST_DATA.xlsx"),
  load_ID    = 10)

# Because CRAN won't let you have lines longer than 100 chars we have to
# build the string then pass it to the function below:
tmp_url      = paste0('https://raw.githubusercontent.com/john-harrold', '/',
                      'formods/refs/heads/master/inst/test_data/SDTM_DM.csv')

# Adding a url as a resource
DM_add_url(
  state      = state,
  url        = tmp_url,
  load_ID    = 11)

# Attaching a file to dataset element:
current_ele  = DM_fetch_current_element(state)
current_ele[["ui"]][["source_id"]] = "10"
current_ele[["ui"]][["ds_sheet"]]  = "DATA"

#This will pull out the source details for an element:
fsres = DM_fetch_source(state=state, element=current_ele)

# Rebuilding the code to attach the file:
current_ele = DM_update_element_code(state = state, element=current_ele, session)

# Running the attachment code:
current_ele = DM_run_code(state = state, element=current_ele)

```

```
# Saving the element:
DM_set_current_element(
  state = state,
  element = current_ele)
```

DM_Server

Data Management Server

Description

Server function for the Data Management Shiny Module

Usage

```
DM_Server(
  id,
  FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml"),
  MOD_yaml_file = system.file(package = "formods", "templates", "DM.yaml"),
  deployed = FALSE,
  react_state = NULL
)
```

Arguments

<code>id</code>	An ID string that corresponds with the ID used to call the modules UI elements
<code>FM_yaml_file</code>	App configuration file with FM as main section.
<code>MOD_yaml_file</code>	Module configuration file with MC as main section.
<code>deployed</code>	Boolean variable indicating whether the app is deployed or not.
<code>react_state</code>	Variable passed to server to allow reaction outside of module (NULL)

Value

UD Server object

Examples

```
if(interactive()){
  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;
}
"

formods.yaml = system.file(package="formods", "templates", "formods.yaml")
ASM.yaml = system.file(package="formods", "templates", "ASM.yaml")
UD.yaml = system.file(package="formods", "templates", "UD.yaml")
DW.yaml = system.file(package="formods", "templates", "DW.yaml")
DM.yaml = system.file(package="formods", "templates", "DM.yaml")
FG.yaml = system.file(package="formods", "templates", "FG.yaml")

# Name of file to indicate we need to load testing data
ftmptest = file.path(tempdir(), "formods.test")

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

#https://fontawesome.com/icons?from=io
data_url =
"https://github.com/john-harrold/formods/raw/master/inst/test_data/TEST_DATA.xlsx"

ui <- dashboardPage(
  skin="black",
  dashboardHeader(title="formods"),
  dashboardSidebar(
    sidebarMenu(
      menuItem("Source Data", tabName="upload", icon=icon("table")),
      menuItem("Manage Data", tabName="manage_data", icon=icon("folder-tree")),
      menuItem("Workflows", tabName="workflows", icon=icon("robot")),
      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,
                htmlOutput(NS("ASM", "ui_asm_compact")))),
tags$p(
    tags$img(
        class = "wrapfig",
src = "https://github.com/john-harrold/formods/raw/master/man/figures/logo.png",
        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="workflows",
        box(title="Run Workflows",
            htmlOutput(NS("ASM", "ASM_ui_workflows")))),
tabItem(tabName="wrangle",
        box(title="Transform and Create Views of Your Data", width=12,
            htmlOutput(NS("DW", "DW_ui_compact")))),
tabItem(tabName="manage_data",
        box(title="Manage and Create Data Sources", width=12,
            htmlOutput(NS("DM", "DM_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()

  #react_FM[["UD"]] = list()
  #react_FM[["FG"]] = list()
  #react_FM[["DW"]] = list()
  #react_FM[["DM"]] = list()
  #react_FM[["ASM"]] = list()

  #Uncomment to populate with test data
  # If the ftmpptest file is present we load test data
  if(file.exists(ftmpptest)){
    sources = c(system.file(package="formods", "preload", "ASM_preload.yaml"),
                system.file(package="formods", "preload", "UD_preload.yaml"),
                system.file(package="formods", "preload", "DM_preload.yaml"),
                system.file(package="formods", "preload", "DW_preload.yaml"),
                system.file(package="formods", "preload", "FG_preload.yaml"))
    res = FM_app_preload(session=session, sources=sources, react_state = react_FM)
  } else if(file.exists("preload.yaml")){

    shinybusy::show_modal_spinner(text="Preloading analysis, be patient", session=session)
    res = FM_app_preload(session=session, sources="preload.yaml", react_state=react_FM)
    shinybusy::remove_modal_spinner(session = session)
  }

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

  # Module servers
  formods::ASM_Server(id="ASM",
                     FM_yaml_file = formods.yaml,
                     MOD_yaml_file = ASM.yaml,
                     deployed      = deployed,
                     react_state   = react_FM,
                     mod_ids       = mod_ids)

```

```

formods::UD_Server( id="UD",
  FM_yaml_file = formods.yaml,
  MOD_yaml_file = UD.yaml,
  deployed     = deployed,
  react_state  = react_FM)
formods::DW_Server( id="DW",
  FM_yaml_file = formods.yaml,
  MOD_yaml_file = DW.yaml,
  deployed     = deployed,
  react_state  = react_FM)
formods::DM_Server( id="DM",
  FM_yaml_file = formods.yaml,
  MOD_yaml_file = DM.yaml,
  deployed     = deployed,
  react_state  = react_FM)
formods::FG_Server( id="FG",
  FM_yaml_file = formods.yaml,
  MOD_yaml_file = FG.yaml,
  deployed     = deployed,
  react_state  = react_FM)
}

shinyApp(ui, server)
}
}

```

DM_set_current_element

Sets the Value for the Current ds

Description

Takes a DM state and returns the current active ds

Usage

```
DM_set_current_element(state, element)
```

Arguments

state	DM state from DM_fetch_state()
element	Element list from DM_fetch_current_element()

Value

DM state object with the current ds set using the supplied value.

Examples

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

# Creates a new empty element
state = DM_new_element(state)

# Delete the current element
state = DM_del_current_element(state)

# Fetch a list of the current element
element = DM_fetch_current_element(state)

# You can modify the element
element[["name"]] = "A more descriptive name"

# You can now place element back in the state
state = DM_set_current_element(state, element)

# This will pull the portion of the code associated with this module.
code = DM_fetch_code(state)

cat(code)

# This forces and update of the module checksum
state = DM_update_checksum(state)
```

DM_test_mksession	<i>Populate Session Data for Module Testing</i>
-------------------	---

Description

Populates the supplied session variable for testing.

Usage

```
DM_test_mksession(session = list())
```

Arguments

session	Shiny session variable (in app) or a list (outside of app)
---------	--

Value

The DM portion of the ‘all_sess_res’ returned from [FM_app_preload](#)

Examples

```
session = shiny::MockShinySession$new()
sess_res = DM_test_mksession(session=session)
```

DM_update_checksum	<i>Updates DM Module Checksum</i>
--------------------	-----------------------------------

Description

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

Usage

```
DM_update_checksum(state)
```

Arguments

state DM state from DM_fetch_state()

Value

DM 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 = DM_test_mksession(session=list())
state    = sess_res$state
session  = sess_res$session
input    = sess_res$input

# Creates a new empty element
state = DM_new_element(state)

# Delete the current element
state = DM_del_current_element(state)

# Fetch a list of the current element
element = DM_fetch_current_element(state)

# You can modify the element
element[["name"]] = "A more descriptive name"

# You can now place element back in the state
state = DM_set_current_element(state, element)

# This will pull the portion of the code associated with this module.
code = DM_fetch_code(state)
```

```
cat(code)

# This forces and update of the module checksum
state = DM_update_checksum(state)
```

DM_update_element_code

Updates Element Code

Description

Updates the code for the current element

Usage

```
DM_update_element_code(state, element, session)
```

Arguments

state	DM state from DM_fetch_state()
element	Element list from DM_fetch_current_element()
session	Shiny session variable

Value

DM element with code updated

DM element with code updated and the 'res\$code_update' field with the following elements:

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

Examples

```
# This creates a testing environment
sess_res = DM_test_mksession()
session = sess_res$session
input    = sess_res$input
state    = sess_res$state

# Next we delete all the elements that are currently defined
ele_ids = names(state[["DM"]][["elements"]])
while(state[["DM"]][["current_element"]] %in% ele_ids){
  state = suppressMessages(DM_del_current_element(state))
}

# This will delete all the pre-defined sources
sources_df = state[["DM"]][["defined_sources"]]
```

```

source_ids = sources_df$ID
for(source_id in source_ids){
  state = suppressMessages(DM_delete_source(state = state, id = source_id))
}

# Adding a file as a resource
DM_add_file(
  state      = state,
  file_name  = "TEST_DATA.xlsx",
  file_path  = system.file(package="formods", "test_data", "TEST_DATA.xlsx"),
  load_ID    = 10)

# Because CRAN won't let you have lines longer than 100 chars we have to
# build the string then pass it to the function below:
tmp_url      = paste0('https://raw.githubusercontent.com/john-harrold', '/',
                      'formods/refs/heads/master/inst/test_data/SDTM-DM.csv')

# Adding a url as a resource
DM_add_url(
  state      = state,
  url        = tmp_url,
  load_ID    = 11)

# Attaching a file to dataset element:
current_ele  = DM_fetch_current_element(state)
current_ele[["ui"]][["source_id"]] = "10"
current_ele[["ui"]][["ds_sheet"]]  = "DATA"

#This will pull out the source details for an element:
fsres = DM_fetch_source(state=state, element=current_ele)

# Rebuilding the code to attach the file:
current_ele = DM_update_element_code(state = state, element=current_ele, session)

# Running the attachment code:
current_ele = DM_run_code(state = state, element=current_ele)

# Saving the element:
DM_set_current_element(
  state  = state,
  element = current_ele)

```

DW_add_wrangling_element

Adding Wrangling Element to Current Data View

Description

Adds the wrangling element to the current data view.

Usage

```
DW_add_wrangling_element(state, session, dwb_res, dwee_res, status = "Success")
```

Arguments

state	DW state from DW_fetch_state()
session	Shiny session variable
dwb_res	Output from dwrs_builder()
dwee_res	Output from dw_eval_element()
status	Wrangling element status returned by UD_fetch_state().

Value

state with data set attached

Examples

```
library(formods)
# The example requires a formods DW state object
sess_res = DW_test_mksession()
state    = sess_res$state
session  = sess_res$session

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    = state,
  session  = session,
  dwb_res  = dwb_res,
  dwee_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, session, current_view)

# This will pull out the available sources for the current data view.
# This is used when doing merging:
avail_ds = DW_fetch_available_sources(state, session, dw_view=current_view)

# This will pull out the meta data for a given object
ds_obj = avail_ds[["catalog"]][["object"]][1]
ds_meta = DW_fetch_obj_ds_meta(
    state = state,
    session = session,
    dw_view=current_view,
    ds_obj=ds_obj)
```

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 <code>DW_fetch_state()</code>
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
sess_res = DW_test_mksession()
state = sess_res$state

rpt = list(summary = list(), sheets=list())

rpt_res = DW_append_report(state,
  rpt      = rpt,
  rpttype = "xlsx")

# Shows if report elements are present
rpt_res$hasrptele

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

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

DW_attach_ds

Attach Data Set to DW View

Description

Attaches a dataset to the DW state supplied.

Usage

```
DW_attach_ds(state, dw_view, ds_source_id)
```

Arguments

state	DW state from DW_fetch_state()
dw_view	Data view list of the format returned from DW_fetch_current_view()
ds_source_id	Source id from DSV[["catalog"]][["object"]] for the dataset.

Value

state with data set attached

Examples

```
# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
sess_res = DW_test_mksession()
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
# JMH TODO: update example
dw_view = DW_fetch_current_view(state=state)

ds_source_id = names(state[["DW"]][["DSV"]][["ds"]])[1]

dw_view = DW_attach_ds(state=state, dw_view=dw_view, ds_source_id=ds_source_id)
```

dw_eval_element	<i>Evaluates Data Wrangling Generated Code</i>
-----------------	--

Description

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

Usage

```
dw_eval_element(state, session, cmd, res_obj = NULL)
```

Arguments

state	DW state from DW_fetch_state()
session	Shiny session variable
cmd	string containing the data wrangling command
res_obj	resource objects needed to run the code such as merge datasets

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

```
library(formods)
# The example requires a formods DW state object
sess_res = DW_test_mksession()
state    = sess_res$state
session  = sess_res$session

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
# elemets 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 = state,
    session = session,
    dwb_res = dwb_res,
    dwee_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, session, current_view)

# This will pull out the available sources for the current data view.
# This is used when doing merging:
avail_ds = DW_fetch_available_sources(state, session, dw_view=current_view)

# This will pull out the meta data for a given object
ds_obj = avail_ds[["catalog"]][["object"]][1]
ds_meta = DW_fetch_obj_ds_meta(
    state = state,
    session = session,
    dw_view=current_view,
    ds_obj=ds_obj)

```

DW_fetch_available_sources

Fetch Available Source

Description

Returns a list of available data sources for a given data wrangling element. This includes any modules the DW module can pull from (e.g. UD and DM) as well as any data views defined before the provided dw_view. If dw_view is NULL then only the sources from the modules DW is dependent on will be returned.

Usage

```
DW_fetch_available_sources(state, session, dw_view = NULL)
```

Arguments

state	DW state object
session	Shiny session variable
dw_view	Optional data view to use as a reference point for DW data sources

Value

list with the following elements

- choices: List of values to use with selectInput()/pickerInput()
 - plain: Raw vector of choices
 - labeled: Named vector of choices
 - grouped: Choices grouped by module
- 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
 - idx: Numerical identifier within the module
 - res_label: optional label that can be defined by a user and used in workflows. Must be unique to the module.
 - checksum: Module checksum
 - DSchecksum: Checksum of the dataset
 - code: Code to generate the dataset
- FM_objs: List of object names that come from other formods modules
- DW_objs: List of object names that come from the current DW module

Examples

```
library(formods)
# The example requires a formods DW state object
sess_res = DW_test_mksession()
state     = sess_res$state
session   = sess_res$session

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 = state,
    session = session,
    dwb_res = dwb_res,
    dwee_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, session, current_view)

# This will pull out the available sources for the current data view.
# This is used when doing merging:
avail_ds = DW_fetch_available_sources(state, session, dw_view=current_view)

# This will pull out the meta data for a given object
ds_obj = avail_ds[["catalog"]][["object"]][1]
ds_meta = DW_fetch_obj_ds_meta(
    state = state,
    session = session,
    dw_view=current_view,
    ds_obj=ds_obj)

```

DW_fetch_code

Fetch Module Code

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()
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

```
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

```
library(formods)
# The example requires a formods DW state object
sess_res = DW_test_mksession()
state    = sess_res$state
session  = sess_res$session

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    = state,
    session  = session,
    dwb_res  = dwb_res,
    dwee_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, session, current_view)

# This will pull out the available sources for the current data view.
# This is used when doing merging:
avail_ds = DW_fetch_available_sources(state, session, dw_view=current_view)

# This will pull out the meta data for a given object
ds_obj   = avail_ds[["catalog"]][["object"]][1]
ds_meta  = DW_fetch_obj_ds_meta(
    state    = state,
    session  = session,
    dw_view=current_view,
    ds_obj=ds_obj)

```

DW_fetch_ds

Fetch Module Datasets

Description

Fetches the datasets contained in the module.

Usage

```
DW_fetch_ds(state, meta_only = FALSE)
```

Arguments

state	UD state from UD_fetch_state()
meta_only	Include only metadata and not the dataset (default FALSE)

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
 - idx: unique numerical ID to identify this dataset in the module.
 - res_label: optional label that can be defined by a user and used in workflows. Must be unique to the module.
 - 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

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

ds = DW_fetch_ds(state)
```

DW_fetch_obj_ds_meta *Fetches Metadata for a Data Source Based on R Object Name*

Description

Data sources can be either resources from other modules (e.g. UD and DM) or previously defined dataviews. This function is a simple interface to retrieve a data source based on the source ID or label.

Resources such as datasets can be specified using the module ID and either identifier index or the resource label. This function will attempt to rectify those and give the "best" answer. If a resource label is supplied and exists uniquely for that module ID, that object will be returned. If not then an error will be returned. If the resource label is not specified and the index is specified and that exists uniquely, that object will be returned. Otherwise an error will be returned.

Usage

```
DW_fetch_obj_ds_meta(state, session, dw_view = NULL, ds_obj = NULL)
```

Arguments

state	DW state from DW_fetch_state()
session	Shiny session variable
dw_view	Optonal data view to use as a reference point for DW data sources set to NULL if only formods sources are to be used.
ds_obj	Name of object that contains data source

Value

list containing the current dataset with the following format:

- isgood: Boolean indicating the whether a resource object was found.
- msgs: Any messages generated
- res_row: Row of catalog associated with resource or NULL if not found.
- DS: Data frame of the current resource

Examples

```
library(formods)
# The example requires a formods DW state object
sess_res = DW_test_mksession()
state    = sess_res$state
session  = sess_res$session

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
# elemets 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  = state,
  session = session,
  dwb_res = dwb_res,
  dwee_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, session, current_view)

# This will pull out the available sources for the current data view.
# This is used when doing merging:
avail_ds = DW_fetch_available_sources(state, session, dw_view=current_view)

# This will pull out the meta data for a given object
ds_obj = avail_ds[["catalog"]][["object"]][1]
ds_meta = DW_fetch_obj_ds_meta(
    state = state,
    session = session,
    dw_view=current_view,
    ds_obj=ds_obj)

```

DW_fetch_state	<i>Fetch Data Wrangling State</i>
----------------	-----------------------------------

Description

Merges default app options with the changes made in the UI

Usage

```
DW_fetch_state(id, input, session, FM_yaml_file, MOD_yaml_file, 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. 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.
 - DSV: Result of FM_fetch_ds().
 - ui: Current value of form elements in the UI
 - ui_hold: List of hold elements to disable updates before a full ui referesh 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 = 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"

# 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,
                        react_state = NULL)

```

DW_fetch_view_deps *Recursively Finds View Dependencies*

Description

For the view ids specified in view_ids this will walk through and find any data views that depend on those view IDS and rebuild them if necessary.

Usage

```
DW_fetch_view_deps(state, session, view_ids = NULL)
```

Arguments

state	DW state object
session	Shiny session variable
view_ids	List of view ids to mend or NULL to attempt to mend all

Value

list with the following elements

- isgood: Return status of the function.
- msgs: Messages to be passed back to the user.
- dep_catalog: Dataframe of dependencies (including specified
 - view_id: Data view ID (e.g., view_3)
 - idx: Numeric data view id (e.g. 3)
 - checksum: Checksum of the data view
 - object_name: R object name used (e.g., DW_myDS_2)
- dep_catalog_ex: Same as dep_catalog but excuding view_ids

Examples

```
sess_res = DW_test_mksession()
state    = sess_res$state
session  = sess_res$session
view_ids = names(state[["DW"]][["views"]])[1]
view_deps = DW_fetch_view_deps(
  state    = state,
  session  = session,
  view_ids = view_ids)
```

DW_hasds	Check DW State For Datasets
----------	-----------------------------

Description

Walks through the DW state object to see if there are any datasets available

Usage

```
DW_hasds(state)
```

Arguments

state DW state from DW_fetch_state()

Value

Logical TRUE if there is a dataset or FALSE otherwise.

Examples

```
sess_res = DW_test_mksession()
state = sess_res[["state"]]
DW_hasds(state)
```

DW_init_state	Initialize DW Module State
---------------	----------------------------

Description

Creates a list of the initialized module state

Usage

```
DW_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	Shiny module ID the name of the list element in react_state where the data set is stored.
session	Shiny session variable module (NULL)

Value

list containing an empty DW state

Examples

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

state
```

DW_mk_preload	<i>Make List of Current DW State</i>
---------------	--------------------------------------

Description

Converts the current DW state into a preload list.

Usage

```
DW_mk_preload(state)
```

Arguments

state	DW state object
-------	-----------------

Value

list with the following elements

- `isgood`: Boolean indicating the exit status of the function.
- `msgs`: Messages to be passed back to the user.
- `yaml_list`: Lists with preload components.

Examples

```
sess_res = DW_test_mksession()
state = sess_res$state
res = DW_mk_preload(state)
```

DW_new_view

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, session)
```

Arguments

<code>state</code>	DW state from <code>DW_fetch_state()</code>
<code>session</code>	Shiny session variable

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

```
library(formods)
# The example requires a formods DW state object
sess_res = DW_test_mksession()
state = sess_res$state
session = sess_res$session

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
# elemets 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 = state,
    session = session,
    dwb_res = dwb_res,
    dwee_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, session, current_view)

# This will pull out the available sources for the current data view.
# This is used when doing merging:
avail_ds = DW_fetch_available_sources(state, session, dw_view=current_view)

# This will pull out the meta data for a given object
ds_obj = avail_ds[["catalog"]][["object"]][1]
ds_meta = DW_fetch_obj_ds_meta(
    state = state,
    session = session,
    dw_view=current_view,
    ds_obj=ds_obj)

```

DW_preload

Preload Data for DW Module

Description

Populates the supplied session variable with information from list of sources.

Usage

```

DW_preload(
    session,
    src_list,

```

```

    yaml_res,
    mod_ID = NULL,
    react_state = list(),
    quickload = FALSE
  )

```

Arguments

session	Shiny session variable (in app) or a list (outside of app)
src_list	List of preload data (all read together with module IDs at the top level)
yaml_res	List data from module yaml config
mod_ID	Module ID of the module being loaded
react_state	Reactive shiny object (in app) or a list (outside of app) used to trigger reactions.
quickload	Logical TRUE to load reduced analysis FALSE to load the full analysis

Value

list with the following elements

- isgood: Boolean indicating the exit status of the function.
- msgs: Messages to be passed back to the user.
- session: Session object
- input: The value of the shiny input at the end of the session initialization.
- state: App state.
- react_state: The react_state components.

DW_proc_pll	<i>Adding Preload List Component</i>
-------------	--------------------------------------

Description

Adds the wrangling component specified in the preload list

Usage

```
DW_proc_pll(state, session = NULL, pll)
```

Arguments

state	DW state from DW_fetch_state()
session	Shiny session variable
pll	Preload list either from a preload file or

Value

state with the preload list added. See the field `state[["DW"]][["res"]][["proc_pll"]]` for the exit status and any messages.

 DW_rebuild_current_view

Forces Rebuild of Current Data View

Description

Will attempt to rebuild data views when changes in the source datasets have been detected.

Usage

```
DW_rebuild_current_view(state, session)
```

Arguments

state	DW state object
session	Shiny session variable

Value

State with current data view rebuilt. Any messages will be stored in state[["DW"]][["res"]][["rebuild_current_view"]]

Examples

```
# JMH add examples
```

 DW_rectify

Rebuilds Data Views After Source Changes

Description

Will attempt to rebuild data views when changes in the source datasets have been detected

Usage

```
DW_rectify(
  state,
  session,
  view_ids = NULL,
  id = NULL,
  id_UD = NULL,
  id_DM = NULL,
  force = FALSE
)
```

Arguments

state	DW state object
session	Shiny session variable
view_ids	List of view ids to mend or NULL to attempt to mend all
id	Shiny id of the DW module
id_UD	Shiny id of the UD module
id_DM	Shiny id of the DM module
force	Boolean to force rebuild

Value

State with dataviews rebuilt if dataset changes were detected

Examples

```
sess_res = DW_test_mksession()
state    = sess_res$state
session  = sess_res$session
view_ids = names(state[["DW"]][["views"]])[1]
state = DW_rectify(
  state    = state,
  session  = session,
  view_ids = view_ids,
  id       = "DW", id_UD="UD", id_DM="DM")
```

DW_Server

Data Wrangling Server

Description

Server function for the data wrangling module

Usage

```
DW_Server(
  id,
  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
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()){
  if(interactive()){
    # These are suggested packages
    library(shinydashboard)
    library(ggpubr)
    library(plotly)
    library(shinybusy)
    library(prompter)
    library(utils)
    library(cli)
    library(formods)

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

    formods.yaml = system.file(package="formods", "templates", "formods.yaml")
    ASM.yaml     = system.file(package="formods", "templates", "ASM.yaml")
    UD.yaml      = system.file(package="formods", "templates", "UD.yaml")
    DW.yaml      = system.file(package="formods", "templates", "DW.yaml")
    DM.yaml      = system.file(package="formods", "templates", "DM.yaml")
    FG.yaml      = system.file(package="formods", "templates", "FG.yaml")

    # Name of file to indicate we need to load testing data
    ftmptest = file.path(tempdir(), "formods.test")

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

```

```

#https://fontawesome.com/icons?from=io
data_url =
"https://github.com/john-harrold/formods/raw/master/inst/test_data/TEST_DATA.xlsx"

ui <- dashboardPage(
  skin="black",
  dashboardHeader(title="formods"),
  dashboardSidebar(
    sidebarMenu(
      menuItem("Source Data",      tabName="upload",      icon=icon("table")),
      menuItem("Manage Data",      tabName="manage_data", icon=icon("folder-tree")),
      menuItem("Workflows",        tabName="workflows",  icon=icon("robot")),
      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",
                  src = "https://github.com/john-harrold/formods/raw/master/man/figures/logo.png",
                  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="workflows",
        box(title="Run Workflows",
          htmlOutput(NS("ASM", "ASM_ui_workflows")))),
      tabItem(tabName="wrangle",
        box(title="Transform and Create Views of Your Data", width=12,

```

```

        htmlOutput(NS("DW", "DW_ui_compact")))),
    tabItem(tabName="manage_data",
      box(title="Manage and Create Data Sources", width=12,
        htmlOutput(NS("DM", "DM_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()

  #react_FM[["UD"]] = list()
  #react_FM[["FG"]] = list()
  #react_FM[["DW"]] = list()
  #react_FM[["DM"]] = list()
  #react_FM[["ASM"]] = list()

  #Uncomment to populate with test data
  # If the ftmp test file is present we load test data
  if(file.exists(ftmptest)){

```

```

    sources = c(system.file(package="formods", "preload", "ASM_preload.yaml"),
                system.file(package="formods", "preload", "UD_preload.yaml"),
                system.file(package="formods", "preload", "DM_preload.yaml"),
                system.file(package="formods", "preload", "DW_preload.yaml"),
                system.file(package="formods", "preload", "FG_preload.yaml"))
    res = FM_app_preload(session=session, sources=sources, react_state = react_FM)
  } else if(file.exists("preload.yaml")){

    shinybusy::show_modal_spinner(text="Preloading analysis, be patient", session=session)
    res = FM_app_preload(session=session, sources="preload.yaml", react_state=react_FM)
    shinybusy::remove_modal_spinner(session = session)
  }

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

  # Module servers
  formods::ASM_Server(id="ASM",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = ASM.yaml,
    deployed      = deployed,
    react_state   = react_FM,
    mod_ids       = mod_ids)
  formods::UD_Server( id="UD",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = UD.yaml,
    deployed      = deployed,
    react_state   = react_FM)
  formods::DW_Server( id="DW",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = DW.yaml,
    deployed      = deployed,
    react_state   = react_FM)
  formods::DM_Server( id="DM",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = DM.yaml,
    deployed      = deployed,
    react_state   = react_FM)
  formods::FG_Server( id="FG",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = FG.yaml,
    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

```
DW_set_current_view(state, session, dw_view)
```

Arguments

state	DW state from DW_fetch_state()
session	Shiny session variable
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
sess_res = DW_test_mksession()
state     = sess_res$state
session   = sess_res$session

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
# elemets 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    = state,
  session  = session,
  dwb_res  = dwb_res,
  dwee_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, session, current_view)

# This will pull out the available sources for the current data view.
# This is used when doing merging:
avail_ds = DW_fetch_available_sources(state, session, dw_view=current_view)

# This will pull out the meta data for a given object
ds_obj = avail_ds[["catalog"]][["object"]][1]
ds_meta = DW_fetch_obj_ds_meta(
    state = state,
    session = session,
    dw_view=current_view,
    ds_obj=ds_obj)

```

DW_test_mksession	<i>Populate Session Data for Module Testing</i>
-------------------	---

Description

Populates the supplied session variable for testing.

Usage

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

Arguments

session Shiny session variable (in app) or a list (outside of app)

Value

The DW portion of the 'all_sess_res' returned from [FM_app_preload](#)

See Also

[FM_app_preload](#)

Examples

```

session = shiny::MockShinySession$new()
sess_res = DW_test_mksession(session=session)

```

DW_update_checksum	<i>Updates DW Module Checksum</i>
--------------------	-----------------------------------

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 = sess_res$session
input    = sess_res$input

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

state = DW_update_checksum(state)
```

dwrs_builder	<i>Builds a Data Wrangling R Statement From ui Elements:</i>
--------------	--

Description

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

Usage

```
dwrs_builder(state, session)
```

Arguments

state	DW state from DW_fetch_state()
session	Shiny session variable

Value

list containing the following elements

- isgood: Return status of the function
- cmd: Data wrangling R command
- action: The action being performed
- pll: Preload list (pll) containing components to save with mk_preload.
- res_obj: Resource objects beyond data view used by this wrangling statement (e.g. data source used when merging) or NULL if no resource is used.
- res_obj_DSchecksum: Checksum of the res_obj or NULL if no resource is used.
- res_obj_mod_id: Module ID of the res obj or NULL if no resource is used.
- 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
sess_res = DW_test_mksession()
state    = sess_res$state
session  = sess_res$session

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
# elemets 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    = state,
  session  = session,
  dwb_res  = dwb_res,
  dwee_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, session, current_view)

# This will pull out the available sources for the current data view.
# This is used when doing merging:
avail_ds = DW_fetch_available_sources(state, session, dw_view=current_view)

# This will pull out the meta data for a given object
ds_obj = avail_ds[["catalog"]][["object"]][1]
ds_meta = DW_fetch_obj_ds_meta(
  state = state,
  session = session,
  dw_view=current_view,
  ds_obj=ds_obj)

```

fers_builder

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

Takes the current ui elements and constructs the appropriate ggplot commands forom the user input. The plot commands assume the existance of a ggplot object p.

Usage

```
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
- pll: Preload list (pll) containing components to save with mk_preload.
- desc: Verbose description of the element
- msgs: Messages to be passed back to the user

Examples

```

sess_res = FG_test_mksession()
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

```
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	<i>Fetches the Current Version of Pacakge</i>
-----------------------	---

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
---------	-----------------

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')
```

fetch_resource	<i>Fetches Resource Object</i>
----------------	--------------------------------

Description

Resources such as datasets can be specified using the module ID and either identifier index or the resource label. This function will attempt to rectify those and give the "best" answer. If a resource label is supplied and exists uniquely for that module ID, that object will be returned. If not then an error will be returned. If the resource label is not specified and the index is specified and that exists uniquely, that object will be returned. Otherwise an error will be returned.

Usage

```
fetch_resource(catalog = NULL, id = NULL, idx = NULL, res_label = NULL)
```

Arguments

catalog	Catalog table returned from formods fetch functions (e.g. 'FM_fetch_ds()).
id	Module id.
idx	Element id from module, can be set to NULL or "" if the res_label option is set.
res_label	Resource label, must be set to NULL or "" to ignore the resource label and use idx.

Value

list containing the current dataset with the following format:

- isgood: Boolean indicating the whether a resource object was found.
- msgs: Any messages generated
- res_row: Row of catalog associated with resource or NULL if not found.
- res_obj: Name of resource object or NULL if not found.

Examples

```
# We need a module state and a Shiny session variable
# to use this function:
sess_res = UD_test_mksession()
session = sess_res$session
state    = sess_res$state

DSV = FM_fetch_ds(state=state, session=session, ids=c("UD", "DM", "DW"), meta_only=TRUE)

fetch_resource(catalog=DSV[["catalog"]], id = "UD", idx="1", res_label = "")
```

FG_append_report	<i>Append Report Elements</i>
------------------	-------------------------------

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()
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$hasrptele

# 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,
  pll = 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.
pll	Preload list for 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

Value

State with the build result stored in the current figure. Pull out the figure to inspect it:

```
current_fig = FG_fetch_current_fig(state)
```

Look at this logical object to test if the build worked.

```
current_fig[["add_isgood"]]
```

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 = 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"

# If this is a paginated figure, and we can access a specific
pg_1 = FG_extract_page(state, 1)

# 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

```
FG_extract_page(state, page)
```

Arguments

state	FG state from FG_fetch_state()
page	Page number to extract

Value

ggplot object with the specified page.

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 = 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"
```

```

# If this is a paginated figure, and we can access a specific
pg_1 = FG_extract_page(state, 1)

# 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_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()
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()

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

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

# If this is a paginated figure, and we can access a specific
pg_1 = FG_extract_page(state, 1)

# 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	<i>Fetch Figure Generation State</i>
----------------	--------------------------------------

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, 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.
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 referesh 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**: Boolean indicating the success/failure of adding the last figure elemnt.
 - * **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**: The object name that contains the figure in the generated code.
 - * **fig_dsview**: Name of the dataset view for the current figure (also the R object name of the dataset view).
 - * **fobj**: The ggplot object of the figure.
 - * **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**: Any messages generated when building the figure.
 - * **notes**: Figure notes (user editable)
 - * **num_pages**: Number of pages in the figure.
 - * **page**: The currently selected figure page.
- **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

```
# 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.

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

# Creating an empty state object
state = FG_fetch_state(id          = "FG",
                       input       = input,
                       session      = session,
                       FM_yaml_file = FM_yaml_file,
                       MOD_yaml_file = MOD_yaml_file,
                       react_state  = NULL)

state
```

FG_init_state	<i>Initialize FG Module State</i>
---------------	-----------------------------------

Description

Creates a list of the initialized module state

Usage

```
FG_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	Shiny module ID
session	Shiny session variable

Value

list containing an empty app state object

Examples

```
# 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",
  session      = session)

state
```

FG_mk_preload	<i>Make List of Current FG State</i>
---------------	--------------------------------------

Description

Converts the current FG state into a preload list.

Usage

```
FG_mk_preload(state)
```

Arguments

state	FG state object
-------	-----------------

Value

list with the following elements

- isgood: Boolean indicating the exit status of the function.
- msgs: Messages to be passed back to the user.
- yaml_list: Lists with preload components.

Examples

```
sess_res = FG_test_mksession()
state = sess_res$state
res = FG_mk_preload(state)
```

FG_new_fig	<i>Initialize New Figure</i>
------------	------------------------------

Description

Creates a new figure in a FG module

Usage

```
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

```

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

# If this is a paginated figure, and we can access a specific
pg_1 = FG_extract_page(state, 1)

# 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_preload

Preload Data for FG Module

Description

Populates the supplied session variable with information from list of sources.

Usage

```

FG_preload(
  session,
  src_list,
  yaml_res,
  mod_ID = NULL,
  react_state = list(),

```

```

    quickload = FALSE
  )

```

Arguments

<code>session</code>	Shiny session variable (in app) or a list (outside of app)
<code>src_list</code>	List of preload data (all read together with module IDs at the top level)
<code>yaml_res</code>	List data from module yaml config
<code>mod_ID</code>	Module ID of the module being loaded.
<code>react_state</code>	Reactive shiny object (in app) or a list (outside of app) used to trigger reactions.
<code>quickload</code>	Logical TRUE to load reduced analysis FALSE to load the full analysis

Value

list with the following elements

- `isgood`: Boolean indicating the exit status of the function.
- `msgs`: Messages to be passed back to the user.
- `session`: Session object
- `input`: The value of the shiny input at the end of the session initialization.
- `state`: App state.
- `react_state`: The `react_state` components.

FG_Server

Figure Generation Server

Description

Server function for the figure generation module

Usage

```

FG_Server(
  id,
  FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml"),
  MOD_yaml_file = system.file(package = "formods", "templates", "FG.yaml"),
  deployed = FALSE,
  react_state = NULL
)

```

Arguments

<code>id</code>	An ID string that corresponds with the ID used to call the module's UI function
<code>FM_yaml_file</code>	App configuration file with FM as main section.
<code>MOD_yaml_file</code>	Module configuration file with MC as main section.
<code>deployed</code>	Boolean variable indicating whether the app is deployed or not.
<code>react_state</code>	Variable passed to server to allow reaction outside of module (NULL)

Value

FG Server object

Examples

```

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

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

    formods.yaml = system.file(package="formods", "templates", "formods.yaml")
    ASM.yaml = system.file(package="formods", "templates", "ASM.yaml")
    UD.yaml = system.file(package="formods", "templates", "UD.yaml")
    DW.yaml = system.file(package="formods", "templates", "DW.yaml")
    DM.yaml = system.file(package="formods", "templates", "DM.yaml")
    FG.yaml = system.file(package="formods", "templates", "FG.yaml")

    # Name of file to indicate we need to load testing data
    ftmptest = file.path(tempdir(), "formods.test")

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

    #https://fontawesome.com/icons?from=io
    data_url =
    "https://github.com/john-harrold/formods/raw/master/inst/test_data/TEST_DATA.xlsx"

    ui <- dashboardPage(
      skin="black",
      dashboardHeader(title="formods"),
      dashboardSidebar(
        sidebarMenu(

```

```

menuItem("Source Data",      tabName="upload",      icon=icon("table")),
menuItem("Manage Data",     tabName="manage_data", icon=icon("folder-tree")),
menuItem("Workflows",       tabName="workflows",   icon=icon("robot")),
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",
              src = "https://github.com/john-harrold/formods/raw/master/man/figures/logo.png",
              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.')
```

```

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()

  #react_FM[["UD"]] = list()
  #react_FM[["FG"]] = list()
  #react_FM[["DW"]] = list()
  #react_FM[["DM"]] = list()
  #react_FM[["ASM"]] = list()

  #Uncomment to populate with test data
  # If the ftmp test file is present we load test data
  if(file.exists(ftmp test)){
    sources = c(system.file(package="formods", "preload", "ASM_preload.yaml"),
      system.file(package="formods", "preload", "UD_preload.yaml"),
      system.file(package="formods", "preload", "DM_preload.yaml"),
      system.file(package="formods", "preload", "DW_preload.yaml"),
      system.file(package="formods", "preload", "FG_preload.yaml"))
    res = FM_app_preload(session=session, sources=sources, react_state = react_FM)
  } else if(file.exists("preload.yaml")){

shinybusy::show_modal_spinner(text="Preloading analysis, be patient", session=session)
res = FM_app_preload(session=session, sources="preload.yaml", react_state=react_FM)

```

```

    shinybusy::remove_modal_spinner(session = session)
  }

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

  # Module servers
  formods::ASM_Server(id="ASM",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = ASM.yaml,
    deployed      = deployed,
    react_state   = react_FM,
    mod_ids       = mod_ids)
  formods::UD_Server( id="UD",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = UD.yaml,
    deployed      = deployed,
    react_state   = react_FM)
  formods::DW_Server( id="DW",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = DW.yaml,
    deployed      = deployed,
    react_state   = react_FM)
  formods::DM_Server( id="DM",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = DM.yaml,
    deployed      = deployed,
    react_state   = react_FM)
  formods::FG_Server( id="FG",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = FG.yaml,
    deployed      = deployed,
    react_state   = react_FM)
}

shinyApp(ui, server)
}
}

```

FG_set_current_fig	<i>Sets Current Figure</i>
--------------------	----------------------------

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 = 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"

# If this is a paginated figure, and we can access a specific
pg_1 = FG_extract_page(state, 1)

# 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_test_mksession	<i>Populate Session Data for Module Testing</i>
-------------------	---

Description

Populates the supplied session variable for testing.

Usage

```
FG_test_mksession(session = list(), full = FALSE)
```

Arguments

- session Shiny session variable (in app) or a list (outside of app) session = shiny::MockShinySession\$new().
- full Boolean indicating if the full test session should be created.

Value

The FG portion of the ‘all_sess_res’ returned from [FM_app_preload](#)

See Also

[FM_app_preload](#)

Examples

```
session = shiny::MockShinySession$new()
sess_res = FG_test_mksession(session=session)
```

FG_update_checksum	<i>Updates FG Module Checksum</i>
--------------------	-----------------------------------

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)
```

Arguments

- state FG state from FG_fetch_state()

Value

state with checksum updated.

Examples

```
# This will create a populated FG state object:

sess_res = FG_test_mksession()
state    = sess_res$state
state = FG_update_checksum(state)
```

FM_add_ui_tooltip	<i>Add Tooltip to UI Element</i>
-------------------	----------------------------------

Description

Adds a tool tip to a user element.

Usage

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

Arguments

state	Current module state after yaml file has been read.
ui ele	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:
sess_res = UD_test_mksession()
state = sess_res$state
uiele = shiny::textInput(inputId = "my input", label="example input")

uiele = FM_add_ui_tooltip(state, uiele)
}

```

FM_app_preload

*Preload Data Into App***Description**

Populates session data for testing or to load a specific analysis.

Usage

```

FM_app_preload(
  session,
  sources = NULL,
  react_state = list(),
  quickload = FALSE
)

```

Arguments

session	Shiny session variable (in app) or a list (outside of app)
sources	Vector of at corresponds with the ID used to call the modules UI elements
react_state	Reactive shiny object (in app) or a list (outside of app) used to trigger reactions
quickload	Logical TRUE to load reduced analysis FALSE to load the full analysis

Value

list with the following elements

- isgood: Boolean indicating the exit status of the function.
- msgs: Messages to be passed back to the user.
- all_sess_res: List containing the result for each module stored in the list name with the module ID.
- session: Returning the session variable to be used in scripting (not in app).

Examples

```

sources=c(system.file(package="formods", "preload", "ASM_preload.yaml"),
          system.file(package="formods", "preload", "UD_preload.yaml"))
res = FM_app_preload(session=list(), sources=sources)

```

FM_build_comment	Create RStudio Formatted Comments
------------------	-----------------------------------

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

```
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

```
# We need a Shiny session object to use this function:
sess_res = DW_test_mksession()
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 Informaiton About the App

Description

Returns diagnostic information about the app

Usage

```
FM_fetch_app_info(session)
```

Arguments

session Shiny session variable.

Value

List with information about the app with the following structure

- uiele: All system information as UI elements to be used in shiny apps.
- uiele_packages: UI element for installed packages to be used in shiny apps.
- uiele_options: UI element for current options.
- uiele_modules: UI element for loaded formods modules to be used in shiny apps.
- modules: List with formods module IDs used in the app as the element names. Each contains details about that module.
- msgs: System information as text to be used in a report/terminal.
- si_packages Dataframe with currently used packages.
- si_options Dataframe with current options

Examples

```
# We need a Shiny session object to use this function:
sess_res = UD_test_mksession()
session = sess_res$session
app_info = FM_fetch_app_info(session)
app_info$msgs
```

FM_fetch_app_state	<i>Fetches the App State</i>
--------------------	------------------------------

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

```
# We need a Shiny session object to use this function:
sess_res = UD_test_mksession()
session = sess_res$session
app_state = FM_fetch_app_state(session)
app_state
```

FM_fetch_current_mods	<i>Fetches Details About Current Modules</i>
-----------------------	--

Description

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

Usage

```
FM_fetch_current_mods()
```

Value

list with details about the currently supported modules.

Examples

```
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

```
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

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

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

df = rio::import(file=data_file_local, which=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	<i>Fetches Dependency Information</i>
---------------	---------------------------------------

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

```
# We need a Shiny session object to use this function:
sess_res = UD_test_mksession()
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

```
FM_fetch_ds(state, session, ids = NULL, meta_only = FALSE)
```

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
meta_only	Include only metadata and not the dataset (default FALSE) 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: Optional data frame with metadata about the columns of the dataset in DS.
 - code: Code to generate the dataset.
 - checksum: Module checksum when the dataset was pulled
 - DSchecksum: Checksum of the dataframe in DS
- choices: List of values to use with selectInput()/pickerInput()
 - plain: Raw vector of choices
 - labeled: Named vector of choices
 - grouped: Choices grouped by module
- 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
 - idx: Numerical identifier within the module

- res_label: optional label that can be defined by a user and used in workflows. Must be unique to the module.
- 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$my_id`.
- module_ids: List with an entry for each module_id. The element name is the module id. For each of these there is a checksum. For example to access the checksum of a module with id of 'my_id', you would use the following: `res$module_ids$my_id`.

Examples

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

<code>FM_fetch_log_path</code>	<i>Fetches the Path to the Log File</i>
--------------------------------	---

Description

Use this to get the path to the formods log file

Usage

```
FM_fetch_log_path(state)
```

Arguments

<code>state</code>	module state after yaml read
--------------------	------------------------------

Value

Character string with the path to the log file.

Examples

```
# Within shiny a session variable will exist,
# this creates one here for testing purposes:
sess_res = UD_test_mksession()
session = sess_res$session
state = sess_res$state
FM_fetch_log_path(state)
```

FM_fetch_md1

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

```
FM_fetch_md1(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_md1, see vignette("making_modules", package = "formods")
- catalog: Dataframe containing the a tabular catalog of the models found.
- choices: List of values to use with selectInput()/pickerInput()
 - plain: Raw vector of choices
 - labeled: Named vector of choices
 - grouped: Choices grouped by module
 - label: Text label for the model (e.g. one-compartment model).
 - MOD_TYPE: Type of module.
 - id: Module ID.
 - rx_obj: The rxode2 object.

- rx_obj_name: The rxode2 object name that holds the model.
- ts_obj: List of timescale information for the system and details of other timescales (`list(system="weeks", details = list(days=list(verb="days", conv=86400)))`)
- ts_obj_name: The object name that holds the timescale for this model.
- fcn_def: Text to define the model.
- MDLMETA: Notes about the model.
- code: Code to generate the model.
- checksum: Module checksum.
- MDLchecksum: Model checksum.

Examples

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

<code>FM_fetch_mod_state</code>	<i>Fetch the Module State</i>
---------------------------------	-------------------------------

Description

Fetches the module state from the userdata under the specified id

Usage

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

Arguments

<code>session</code>	Shiny session variable.
<code>id</code>	ID string for the module.

Value

module state or NULL if it's not defined.

Examples

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

`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

```
FM_fetch_user_files_path(state)
```

Arguments

<code>state</code>	module state after yaml read
--------------------	------------------------------

Value

Character string with the path to the log file.

Examples

```
# We need a state object to use this function:
sess_res = UD_test_mksession()
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

```
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:
```

```

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_has_ds_changed	<i>Compares DS Summary with Current State for Changes</i>
-------------------	---

Description

Takes the result of `FM_fetch_ds()` and compares that to the current `react_state` to determine if any datasets have changed for the given module IDs.

Usage

```
FM_has_ds_changed(state, fdres, ids, react_state)
```

Arguments

<code>state</code>	module state after yaml read
<code>fdres</code>	Output of <code>FM_fetch_ds()</code> .
<code>ids</code>	IDs the current module is
<code>react_state</code>	Reactive shiny object (in app) or a list (outside of app) used to trigger reactions

Value

Logical indicating if a change has been detected (TRUE) or not (FALSE)

Examples

```

session = shiny::MockShinySession$new()
sess_res = FG_test_mksession(session=session)
state = sess_res$state
FM_has_ds_changed(state = state, fdres = NULL, ids = ("DW"), react_state=list())

```

FM_init_state	<i>Initialize a formods State Object</i>
---------------	--

Description

Initializes a formods state object with common elements.

Usage

```
FM_init_state(
  FM_yaml_file,
  MOD_yaml_file,
  id,
  dep_mod_ids = c(),
  MT,
  button_counters,
  ui_ids,
  ui_hold,
  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.
dep_mod_ids	Vector of module ids this module depends on.
MT	Type of module using the short name (e.g. "UD", "FG", etc.).
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 = 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

*Adds 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

```

# We need a module state to use this function:
sess_res = UD_test_mksession()
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	<i>Show Message to User</i>
------------	-----------------------------

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_app_preload	<i>Create Preload List from App</i>
-------------------	-------------------------------------

Description

Creates the preload list from the current app state

Usage

```
FM_mk_app_preload(session)
```

Arguments

session Shiny session variable (in app) or a list (outside of app)

Value

- list with the following elements
- isgood: Boolean indicating the exit status of the function.
 - msgs: Messages to be passed back to the user.
 - yaml_list: Lists with preload components.
 - session: Returning the session variable to be used in scripting (not in app).

Examples

```
sources=system.file(package="formods", "preload", "UD.yaml")
sess_res = FM_app_preload(session=list(), sources=sources)
mkp_res = FM_mk_app_preload(sess_res$session)
```

FM_mk_error_fig	<i>Generates ‘ggplot’ Object with Error Message</i>
-----------------	---

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

```
FM_mk_error_fig(msgs)
```

Arguments

msgs Vector of error messages

Value

ggplot object

Examples

```
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

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

Arguments

state	Module state generating the notification
session	Shiny session variable

Value

Boolean variable indicating if the notification was triggered

Examples

```
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_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 = sess_res$session
state = sess_res$state
FM_pause_screen(state, session, "pausing")
FM_resume_screen(state, session)
```

FM_pretty_sort	<i>Centralized Sorting Function</i>
----------------	-------------------------------------

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 = rio::import(file=data_file_local, which=sheet)
# Regular sorting:
sort(unique(df$Cohort))
FM_pretty_sort(unique(df$Cohort))
```

FM_proc_include	<i>Process Include Files in formods.yaml</i>
-----------------	--

Description

Reads in the formods.yaml file and copies any include files that are needed into the user directory.

Usage

```
FM_proc_include(state, session)
```

Arguments

state	formods State object.
session	Shiny session variable.

Value

No return value, sets include flag to indicate everything has been set.

Examples

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

FM_read_yaml	<i>Fetch Yaml Contents</i>
--------------	----------------------------

Description

Wrapper for read_yaml() to make this function available to modules dependent on formods.

Usage

```
FM_read_yaml(file)
```

Arguments

file	yaml file.
------	------------

Value

List containing the contents of the yaml file.

Examples

```
res = FM_read_yaml(  
  system.file(package = "formods",  
               "templates",  
               "formods.yaml"))  
  
res
```

FM_reset_app*Resets the App State*

Description

Removes formods data from the app.

Usage

```
FM_reset_app(session)
```

Arguments

`session` Shiny session variable.

Value

session variable with app data removed.

Examples

```
# We need a Shiny session object to use this function:  
sess_res = UD_test_mksession()  
session = sess_res$session  
app_state = FM_reset_app(session)  
app_state
```

FM_resume_screen*Stops Modal Screen Pause*

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

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

FM_set_app_state	<i>Set the App State</i>
------------------	--------------------------

Description

Takes a loaded app state and overwrites the current 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.

Value

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

Examples

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

FM_set_mod_state	<i>Set the Module State</i>
------------------	-----------------------------

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:
sess_res = UD_test_mksession()
session = sess_res$session
state = sess_res$state
FM_set_mod_state(session, "UD", state)
```

FM_set_notification	<i>Shiny Notification</i>
---------------------	---------------------------

Description

Generates a notification that should only show once.

Usage

```
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

```

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

```
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 messaages.

Value

state with ui message set.

Examples

```
# We need a module state object to use this function:
sess_res = UD_test_mksession()
state = sess_res$state
state = FM_set_ui_msg(state, "Something happend.")
```

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

cmd	Character object containing the R command to evaluate in the try/catch block
tc_env	list of with names corresponding to object names and corresponding Values to define in the try/catch environment
capture	Character vector of values to capture after the command is successfully captured. If capture is NULL (e.g., c()), then all objects generated by cmd will be captured.

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	<i>formods: Shiny modules for common tasks.</i>
---------	---

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	<i>Checks 'formods' Dependencies</i>
---------------	--------------------------------------

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	<i>Deprecated: Detect if a UI element has changed</i>
-------------	---

Description

Deprecated please use has_updated instead: Takes a UI element value and an older value and determines if it has been modified

Usage

```
has_changed(ui_val = NULL, old_val = NULL, init_value = c(""))
```

Arguments

- ui_val Current value from the UI.
- old_val Last value of of the element.
- init_value Default value for reading in UI data when it has not been defined.

Value

Boolean result of the comparison

Examples

```
changed_true = has_changed(ui_val = "a", old_val = "")
changed_true
changed_false = has_changed(ui_val = "a", old_val = "a")
changed_false
```

has_updated	<i>Detect if a UI element has updated</i>
-------------	---

Description

Takes a UI element value and an older value and determines if it has been modified

Usage

```
has_updated(ui_val = NULL, old_val = NULL, init_val = NULL)
```

Arguments

- ui_val Current value from the UI.
- old_val Last value of of the element. defined.
- init_val List of values to skip. These are values expected to be assigned on initialization. For buttons it may be 0. For others it may be "".

Value

Boolean result of the comparison

Examples

```
changed_true = has_updated(ui_val = "a", old_val = "")
changed_true
changed_false = has_updated(ui_val = "a", old_val = "a")
changed_false
```

icon_link	<i>Creates Icon Link</i>
-----------	--------------------------

Description

Creates a link to a Shiny icon

Usage

```
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

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

is_excel	<i>Determines if a File is Excel</i>
----------	--------------------------------------

Description

Check to see if a file is an excel file

Usage

```
is_excel(file_name)
```

Arguments

file_name	Name of file to check.
-----------	------------------------

Value

Logical indicating if a file is an excel file or not.

Examples

```
is_excel("file.xls")  
is_excel("file.xlsx")  
is_excel("file.csv")
```

is_importable	<i>Determines if a File is Importable</i>
---------------	---

Description

Check to see if a file can be imported

Usage

```
is_importable(file_name)
```

Arguments

file_name	Name of file to check.
-----------	------------------------

Value

Logical indicating if a file is an excel file or not.

Examples

```
is_importable("file.xls")  
is_importable("file.csv")  
is_importable("file.bob")
```

is_installed	<i>Determines if a Package is Installed</i>
--------------	---

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')
```

is_shiny	<i>Determine if Object is Shiny Session Object</i>
----------	--

Description

Determines if the specified object is a shiny session object.

Usage

```
is_shiny(session = list())
```

Arguments

session	Session object
---------	----------------

Value

Logical indicating if the object is a shiny session object or not

Examples

```
is_shiny(session = list())
```

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

Examples

```
linspace(0,100, 20)
```

mk_preload_dir

Construct Preload Directory

Description

Given module configuration files, datasets, etc. this will construct a preload directory generally used for testing purposes.

Usage

```
mk_preload_dir(
  directory = tempfile(pattern = "preload_"),
  preload = NULL,
  rpt_templates = c(system.file(package = "onbrand", "templates", "report.docx"),
    system.file(package = "onbrand", "templates", "report.pptx"), system.file(package =
      "onbrand", "templates", "report.docx")),
  mod_yaml = c(system.file(package = "formods", "templates", "formods.yaml"),
    system.file(package = "formods", "templates", "ASM.yaml"), system.file(package =
      "formods", "templates", "UD.yaml"), system.file(package = "formods", "templates",
      "DM.yaml"), system.file(package = "formods", "templates", "DW.yaml"),
    system.file(package = "formods", "templates", "FG.yaml")),
  include = NULL
)
```

Arguments

directory	Target directory to contain the preload file structure.
preload	One or more preload yaml files to include. Multiple files will be combined into a single preload.yaml file.
rpt_templates	Vector containing PowerPoint, Word and the onbrand yaml mapping file.
mod_yaml	Vector of module yaml files to include
include	List of files to include each element of the list should include a 'from' option with the absolute path to the source file and a 'to' with the name of the file in the preload directory. If you want to place the in a subdirectory of the preload directory include the optional path option. (e.g. <code>list(UD = list(from="/full/path/to/source/file.csv", path="data/stuff", to="newfile.csv"))</code>)

Value

List with the following elements:

- isgood: Return status of the function.
- msgs: Error or warning messages if any issues were encountered.

Examples

```
pldir = file.path(tempdir(), "preload_test")

mpd_res = mk_preload_dir(
  directory = pldir,
  preload = c(system.file(package="formods", "preload", "ASM_preload.yaml"),
              system.file(package="formods", "preload", "UD_preload.yaml")),
  mod_yaml = c(
    system.file(package="formods", "templates", "formods.yaml"),
    system.file(package="formods", "templates", "ASM.yaml"),
    system.file(package="formods", "templates", "UD.yaml"),
    system.file(package="formods", "templates", "DM.yaml"),
    system.file(package="formods", "templates", "DW.yaml"),
    system.file(package="formods", "templates", "FG.yaml")),
)

dir(pldir)
```

new_module_template	<i>Makes Template Files for formods New Module</i>
---------------------	--

Description

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

Usage

```
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.
- server: Server.R file.
- 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

```
new_module_template()
```

remove_hold	<i>Removes Hold on UI Element</i>
-------------	-----------------------------------

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")
```

render_str	<i>Evaluate R Code in String</i>
------------	----------------------------------

Description

Attempts to evaluate a string as a chunk of R code. If that succeeds it will return the result. If not it will return the original text.

Usage

```
render_str(estr = "")
```

Arguments

estr	String to render.
------	-------------------

Value

String containing the eval'd as a character or the original string

Examples

```
res = render_str(estr="ls()")
```

run_formods	<i>Run the 'formods' Shiny App</i>
-------------	------------------------------------

Description

Runs the test formods. app.

Usage

```
run_formods(
  host = "127.0.0.1",
  port = 3838,
  server_opts = list(shiny.maxRequestSize = 30 * 1024^2),
  devmode = FALSE,
  mksession = FALSE
)
```

Arguments

host	Hostname of the server ("127.0.0.1")
port	Port number for the app (3838)
server_opts	List of options (names) and their vlues (value) e.g. <code>list(shiny.maxRequestSize = 30 * 1024^2)</code> .
devmode	Boolean value, when TRUE will run formods with development modules.
mksession	Boolean value, when TRUE will load test session data for app testing.

Value

Nothing is returned, this function just runs the built-in formods app.

Examples

```
if (interactive()) {  
  run_formods()  
}
```

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")
```

UD_attach_ds

Attach Data Set to UD State

Description

Attaches a dataset to the UD state supplied.

Usage

```
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)

state

```

UD_ds_read

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"

# 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	<i>Fetch Module Code</i>
---------------	--------------------------

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)
```

UD_fetch_ds	<i>Fetch Module Datasets</i>
-------------	------------------------------

Description

Fetches the datasets contained in the module.

Usage

UD_fetch_ds(state, meta_only = FALSE)

Arguments

state UD state from UD_fetch_state()
meta_only Include only metadata and not the dataset (default FALSE)

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
 - idx: unique numerical ID to identify this dataset in the module.
 - res_label: optional label that can be defined by a user and used in workflows. Must be unique to the module.
 - 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

```
# 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())
state = sess_res$state

ds_res = UD_fetch_ds(state)
```

UD_fetch_state

Fetch Upload Data State

Description

Merges default app options with the changes made in the UI

Usage

```
UD_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

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.
 - sheets: If the uploaded file is an excel file, this is a character vector of the sheets present in that file.
 - contents: Data frame containing the contents of the data file.
 - checksum: This is an MD5 sum of the contents element and can be used to detect changes in the loaded file.
- MOD_TYPE: Character data containing the type of module "UD"
- 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

```
# 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 )

```

UD_init_state

*Initialize UD Module 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

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_mk_preload	<i>Make List of Current ASM State</i>
---------------	---------------------------------------

Description

Converts the current ASM state into a preload list.

Usage

```
UD_mk_preload(state)
```

Arguments

state	UD state object
-------	-----------------

Value

list with the following elements

- isgood: Boolean indicating the exit status of the function.
- msgs: Messages to be passed back to the user.
- yaml_list: Lists with preload components.

Examples

```
sess_res = UD_test_mksession()
state = sess_res$state
res = UD_mk_preload(state)
```

UD_preload	<i>Preload Data for UD Module</i>
------------	-----------------------------------

Description

Populates the supplied session variable with information from list of sources.

Usage

```
UD_preload(
  session,
  src_list,
  yaml_res,
  mod_ID = NULL,
  react_state = list(),
  quickload = FALSE
)
```

Arguments

session	Shiny session variable (in app) or a list (outside of app)
src_list	List of preload data (all read together with module IDs at the top level)
yaml_res	List data from module yaml config
mod_ID	Module ID of the module being loaded.
react_state	Reactive shiny object (in app) or a list (outside of app) used to trigger reactions.
quickload	Logical TRUE to load reduced analysis FALSE to load the full analysis

Value

list with the following elements

- isgood: Boolean indicating the exit status of the function.
- msgs: Messages to be passed back to the user.
- session: Session object
- input: The value of the shiny input at the end of the session initialization.
- state: App state.
- react_state: The react_state components.

UD_Server

*Data Upload Server***Description**

Server function for the Data Upload Shiny Module

Usage

```
UD_Server(
  id,
  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
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()){
  if(interactive()){
    # These are suggested packages
    library(shinydashboard)
    library(ggpubr)
    library(plotly)
    library(shinybusy)
    library(prompter)
    library(utils)
    library(cli)
    library(formods)

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

    formods.yaml = system.file(package="formods", "templates", "formods.yaml")
    ASM.yaml     = system.file(package="formods", "templates", "ASM.yaml")
    UD.yaml      = system.file(package="formods", "templates", "UD.yaml")
    DW.yaml      = system.file(package="formods", "templates", "DW.yaml")
    DM.yaml      = system.file(package="formods", "templates", "DM.yaml")
    FG.yaml      = system.file(package="formods", "templates", "FG.yaml")

    # Name of file to indicate we need to load testing data
    ftmptest = file.path(tempdir(), "formods.test")

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

    #https://fontawesome.com/icons?from=io
    data_url =
    "https://github.com/john-harrold/formods/raw/master/inst/test_data/TEST_DATA.xlsx"

    ui <- dashboardPage(
      skin="black",
      dashboardHeader(title="formods"),
      dashboardSidebar(
        sidebarMenu(

```

```

menuItem("Source Data",      tabName="upload",      icon=icon("table")) ,
menuItem("Manage Data",     tabName="manage_data", icon=icon("folder-tree")),
menuItem("Workflows",       tabName="workflows",  icon=icon("robot")),
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",
              src = "https://github.com/john-harrold/formods/raw/master/man/figures/logo.png",
              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="workflows",
    box(title="Run Workflows",
      htmlOutput(NS("ASM", "ASM_ui_workflows")))),
  tabItem(tabName="wrangle",
    box(title="Transform and Create Views of Your Data", width=12,
      htmlOutput(NS("DW", "DW_ui_compact")))),
  tabItem(tabName="manage_data",
    box(title="Manage and Create Data Sources", width=12,
      htmlOutput(NS("DM", "DM_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()

  #react_FM[["UD"]] = list()
  #react_FM[["FG"]] = list()
  #react_FM[["DW"]] = list()
  #react_FM[["DM"]] = list()
  #react_FM[["ASM"]] = list()

  #Uncomment to populate with test data
  # If the ftmp test file is present we load test data
  if(file.exists(ftmp test)){
    sources = c(system.file(package="formods", "preload", "ASM_preload.yaml"),
                system.file(package="formods", "preload", "UD_preload.yaml"),
                system.file(package="formods", "preload", "DM_preload.yaml"),
                system.file(package="formods", "preload", "DW_preload.yaml"),
                system.file(package="formods", "preload", "FG_preload.yaml"))
    res = FM_app_preload(session=session, sources=sources, react_state = react_FM)
  } else if(file.exists("preload.yaml")){

    shinybusy::show_modal_spinner(text="Preloading analysis, be patient", session=session)
    res = FM_app_preload(session=session, sources="preload.yaml", react_state=react_FM)
  }
}

```

```

    shinybusy::remove_modal_spinner(session = session)
  }

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

  # Module servers
  formods::ASM_Server(id="ASM",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = ASM.yaml,
    deployed      = deployed,
    react_state   = react_FM,
    mod_ids       = mod_ids)
  formods::UD_Server( id="UD",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = UD.yaml,
    deployed      = deployed,
    react_state   = react_FM)
  formods::DW_Server( id="DW",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = DW.yaml,
    deployed      = deployed,
    react_state   = react_FM)
  formods::DM_Server( id="DM",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = DM.yaml,
    deployed      = deployed,
    react_state   = react_FM)
  formods::FG_Server( id="FG",
    FM_yaml_file = formods.yaml,
    MOD_yaml_file = FG.yaml,
    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 = list())
```

Arguments

session Shiny session variable (in app) or a list (outside of app)

Value

The UD portion of the ‘all_sess_res’ returned from [FM_app_preload](#)

See Also

[FM_app_preload](#)

Examples

```
session = shiny::MockShinySession$new()
sess_res = UD_test_mksession(session=session)
```

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

```
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

```
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()

Examples

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

Index

ASM_check_workflow, [5](#)
ASM_fetch_code, [6](#)
ASM_fetch_dlfm, [7](#)
ASM_fetch_state, [7](#)
ASM_init_state, [8](#)
ASM_load_state, [9](#)
ASM_mk_preload, [10](#)
ASM_onload, [11](#)
ASM_preload, [11](#)
ASM_read_app_state, [12](#)
ASM_run_workflow, [13](#)
ASM_save_state, [14](#)
ASM_Server, [15](#)
ASM_test_mksession, [19](#)
ASM_test_preload, [20](#)
autocast, [21](#)

DM_add_file, [22](#)
DM_add_url, [23](#)
DM_append_report, [25](#)
DM_del_current_element, [26](#)
DM_delete_source, [27](#)
DM_fetch_code, [29](#)
DM_fetch_current_element, [30](#)
DM_fetch_ds, [31](#)
DM_fetch_source, [32](#)
DM_fetch_state, [34](#)
DM_hasds, [35](#)
DM_init_state, [36](#)
DM_mk_preload, [37](#)
DM_new_element, [37](#)
DM_onload, [38](#)
DM_preload, [39](#)
DM_run_code, [40](#)
DM_Server, [42](#)
DM_set_current_element, [46](#)
DM_test_mksession, [47](#)
DM_update_checksum, [48](#)
DM_update_element_code, [49](#)
DW_add_wrangling_element, [50](#)

DW_append_report, [52](#)
DW_attach_ds, [53](#)
dw_eval_element, [54](#)
DW_fetch_available_sources, [55](#)
DW_fetch_code, [57](#)
DW_fetch_current_view, [58](#)
DW_fetch_ds, [59](#)
DW_fetch_obj_ds_meta, [60](#)
DW_fetch_state, [62](#)
DW_fetch_view_deps, [64](#)
DW_hasds, [65](#)
DW_init_state, [65](#)
DW_mk_preload, [66](#)
DW_new_view, [67](#)
DW_preload, [68](#)
DW_proc_pll, [69](#)
DW_rebuild_current_view, [70](#)
DW_rectify, [70](#)
DW_Server, [71](#)
DW_set_current_view, [76](#)
DW_test_mksession, [77](#)
DW_update_checksum, [78](#)
dwrs_builder, [78](#)

fers_builder, [80](#)
fetch_hold, [81](#)
fetch_package_version, [82](#)
fetch_resource, [82](#)
FG_append_report, [83](#)
FG_build, [84](#)
FG_extract_page, [86](#)
FG_fetch_code, [87](#)
FG_fetch_current_fig, [88](#)
FG_fetch_state, [89](#)
FG_init_state, [91](#)
FG_mk_preload, [92](#)
FG_new_fig, [92](#)
FG_preload, [93](#)
FG_Server, [94](#)
FG_set_current_fig, [98](#)

FG_test_mksession, 100
FG_update_checksum, 100
FM_add_ui_tooltip, 101
FM_app_preload, 19, 47, 77, 100, 102, 154
FM_build_comment, 103
FM_fetch_app_code, 103
FM_fetch_app_info, 104
FM_fetch_app_state, 105
FM_fetch_current_mods, 105
FM_fetch_data_format, 106
FM_fetch_deps, 107
FM_fetch_ds, 108
FM_fetch_log_path, 109
FM_fetch_md1, 110
FM_fetch_mod_state, 111
FM_fetch_user_files_path, 112
FM_generate_report, 52, 83, 84, 112
FM_has_ds_changed, 114
FM_init_state, 115
FM_le, 116
FM_message, 117
FM_mk_app_preload, 117
FM_mk_error_fig, 118
FM_notify, 119
FM_pause_screen, 120
FM_pretty_sort, 121
FM_proc_include, 122
FM_read_yaml, 122
FM_reset_app, 123
FM_resume_screen, 123
FM_set_app_state, 124
FM_set_mod_state, 125
FM_set_notification, 125
FM_set_ui_msg, 127
FM_tc, 128
formods, 129
formods-package (formods), 129
formods::FM_generate_report(), 26
formods_check, 129

has_changed, 130
has_updated, 130

icon_link, 131
is_excel, 132
is_importable, 132
is_installed, 133
is_shiny, 133

linspace, 134

mk_preload_dir, 134

new_module_template, 135

remove_hold, 137
render_str, 138
run_formods, 138

set_hold, 139

template_details, 25

UD_attach_ds, 140
UD_ds_read, 142
UD_fetch_code, 144
UD_fetch_ds, 144
UD_fetch_state, 145
UD_init_state, 147
UD_mk_preload, 148
UD_preload, 148
UD_Server, 149
UD_test_mksession, 153
unfactor, 154
use_formods, 155