Package ‘ruminate’

December 2, 2023

Title  A Pharmacometrics Data Transformation and Analysis Tool
Version  0.2.0
Description  Exploration of pharmacometrics data involves both general tools (transformation and plotting) and specific techniques (non-compartmental analysis). This kind of exploration is generally accomplished by utilizing different packages. The purpose of ‘ruminate’ is to create a ‘shiny’ interface to make these tools more broadly available while creating reproducible results.
License  BSD_2_clause + file LICENSE
BugReports  https://github.com/john-harrold/ruminate/issues
URL  https://ruminate.ubiquity.tools/
Encoding  UTF-8
Depends  R (>= 4.2.0)
Imports  digest, dplyr, DT, flextable, formods (>= 0.1.2), ggplot2, onbrand (>= 1.0.3), PKNCA (>= 0.10.2), plotly, rHandsontable, rlang, shiny, shinyAce, shinyWidgets, stringr, tidyr, utils, yaml
Suggests  clipr, gridExtra, knitr, nlmixr2lib, nonmem2rx, prompter, rmarkdown, readxl, rxode2, shinydashboard, testthat (>= 3.0.0), ubiquity
RoxygenNote  7.2.3
Config/testthat/edition  3
VignetteBuilder  knitr
NeedsCompilation  no
Author  John Harrold [aut, cre] (<https://orcid.org/0000-0003-2052-4373>)
Maintainer  John Harrold <john.m.harrold@gmail.com>
Repository  CRAN
Date/Publication  2023-12-02 05:50:02 UTC
R topics documented:

apply_route_map .................................................. 3

dose_records_builder ........................................... 4
MB_append_report ................................................ 6
MB_build_code .................................................... 6
MB_del_current_element ........................................ 8
MB_fetch_catalog ............................................... 11
MB_fetch_code .................................................. 13
MB_fetch_component .......................................... 14
MB_fetch_current_element ................................. 16
MB_fetch_mdl ................................................... 18
MB_fetch_state ............................................... 19
MB_init_state .................................................. 20
MB_new_element .............................................. 21
MB_Server ....................................................... 23
MB_set_current_element ..................................... 24
MB_test_catalog .............................................. 26
MB_test_mksession .......................................... 28
MB_update_checksum ....................................... 29
MB_update_model ............................................ 30
mk_figure_ind_obs ........................................... 32
mk_rx_obj ....................................................... 33
mk_table_ind_obs ........................................... 35
mk_table_nca_params ...................................... 37
NCA_add_int .................................................. 39
NCA_append_report ......................................... 39
nca_builder ................................................... 40
NCA_fetch_ana_ds ............................................ 41
NCA_fetch_ana_pknca ....................................... 43
NCA_fetch_code ................................................ 44
NCA_fetch_current_ana ..................................... 46
NCA_fetch_current_obj .................................. 47
NCA_fetch_data_format .................................. 49
NCA_fetch_ds ................................................ 49
NCA_fetch_np_meta ......................................... 50
NCA_fetch_PKNCA_meta ................................... 51
NCA_fetch_state ............................................. 52
NCA_find_col .................................................. 55
NCA_init_state .............................................. 57
NCA_load_scenario .......................................... 58
NCA_mkactive_ana .......................................... 58
NCA_new_ana .................................................... 60
NCA_process_current_ana .................................. 61
NCA_Server ..................................................... 63
NCA_set_current_ana ....................................... 69
NCA_test_mksession ....................................... 71
ruminate ....................................................... 72
apply_route_map

Applies Route Mapping to Dataset

Description

Used to convert nonstandard dose route values (i.e. "IV") to standard values ("intravascular").

Usage

apply_route_map(route_map = list(), route_col = NULL, DS = NULL)

Arguments

route_map List with names corresponding to the route replacement and a vector of regular expressions to match.
route_col Column name with the route data.
DS Dataframe containing the dataset.

Value

Dataset with the route mapping applied.

Examples

if(system.file(package="readxl") !="\n""){
library(readxl)
#loading a dataset
data_file = system.file(package="formods","test_data","TEST_DATA.xlsx")
myDS = readxl::read_excel(path=data_file, sheet="DATA")

route_map = list(
  intravascular = c("(?i)iv$"),
  extravascular = c("(?i)sc$", "(?i)oral")
)

utils::head(myDS[['ROUTE']])

myDS = apply_route_map(route_map = route_map,
  route_col = "ROUTE",
  DS = myDS)

utils::head(myDS[['ROUTE']])
}
dose_records_builder  Builds Dose Records Dataframe

Description

Takes information about columns in dataset and constructs the dosing records.

Usage

dose_records_builder(
    NCA_DS = NULL,
    dose_from = NULL,
    col_id = NULL,
    col_time = NULL,
    col_ntime = NULL,
    col_route = NULL,
    col_dose = NULL,
    col_cycle = NULL,
    col_dur = NULL,
    col_evid = NULL,
    col_analyte = NULL,
    col_group = NULL
)

Arguments

NCA_DS            Dataset containing dosing records.
dose_from        Method of dose extraction either "cols" or "rows".
col_id            Name of column with subject ID.
col_time          Name of column with time since first dose.
col_ntime         Name of column with time since the last dose (required with dose_from="cols").
col_route         Name of column with route information.
col_dose          Name of column with last dose given.
col_cycle         Name of column with dose cycle (required with dose_from="cols").
col_dur           Name of column with dose duration.
col_evid          Name of column with event ID (required with dose_from="rows").
col_analyte       Name of column with analyte (optional).
col_group         Names of columns with grouping information (optional).

Value

list containing the following elements

• isgood: Return status of the function.
• msgs: Messages to be passed back to the user.
• dose_rec:
Examples

```r
if(system.file(package="readxl") !=""){
  library(dplyr)
  library(readxl)
  library(stringr)

  # Example data file:
  data_file = system.file(package="formods","test_data","TEST_DATA.xlsx")

  # Dataset formatted to extract dosing from columns
  DS_cols = readxl::read_excel(path=data_file, sheet="DATA") |>
    dplyr::filter(EVID == 0) |>
    dplyr::filter(DOSE %in% c(3)) |>
    dplyr::filter(str_detect(string=Cohort, "^MD")) |>
    dplyr::filter(CMT == "C_ng_ml")
  
  drb_res = dose_records_builder(
    NCA_DS = DS_cols,
    dose_from = "cols",
    col_id = "ID",
    col_time = "TIME_DY",
    col_ntime = "NTIME_DY",
    col_route = "ROUTE",
    col_cycle = "DOSE_NUM",
    col_dose = "DOSE",
    col_group = "Cohort")
  
  utils::head(drb_res$dose_rec)

  # Dataset formatted to extract dosing from rows (records)
  DS_rows = readxl::read_excel(path=data_file, sheet="DATA") |>
    dplyr::filter(DOSE %in% c(3)) |>
    dplyr::filter(str_detect(string=Cohort, "^MD")) |>
    dplyr::filter(CMT %in% c("Ac", "C_ng_ml"))
  
  drb_res = dose_records_builder(
    NCA_DS = DS_rows,
    dose_from = "rows",
    col_id = "ID",
    col_time = "TIME_DY",
    col_ntime = "NTIME_DY",
    col_route = "ROUTE",
    col_dose = "AMT",
    col_evid = "EVID",
    col_group = "Cohort")
  
  utils::head(drb_res$dose_rec)
}
```
### MB_append_report  
*Append Report Elements*

**Description**

Appends report elements to a formods report.

**Usage**

```r
MB_append_report(state, rpt, rpttype, gen_code_only = FALSE)
```

**Arguments**

- `state`  
  MB state from `MB_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`: 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`

---

### MB_build_code  
*Build Code to Generate Model*

**Description**

Takes the function definition from an rxode object, a function object name and an rxode object name and creates the code to build those objects.

**Usage**

```r
MB_build_code(state, session, fcn_def, fcn_obj_name, rx_obj_name)
```
Arguments

- **state**: MB state from `MB_fetch_state()`
- **session**: Shiny session variable
- **fcn_def**: Character string containing the function definition for the model
- **fcn_obj_name**: Object name of the function to create.
- **rx_obj_name**: Object name of the rxode2 object to create.

Value

List with the following elements

- **model_code**: Block of code to create the model in the context of a larger script.
- **model_code_sa**: Same as the **model_code** element but meant to stand alone.

Examples

```r
library(ruminate)
# This will get the full session:
sess_res = MB_test_mksession(session=list(), full_session=TRUE)
# This is just for CRAN
# sess_res = MB_test_mksession(session=list())
session = sess_res$session
input = sess_res$input

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

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

# This will provide a list of the available models
models = MB_fetch_catalog(state)
# This is a summary of the tables in the model:
models$summary

# This will test the models in the catalog, set as_cran to FALSE to test all the models.
mtres = MB_test_catalog(state, as_cran=TRUE)
mtres$isgood

# Creates a new empty element
state = MB_new_element(state)
```
# Delete the current element
state = MB_del_current_element(state)

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

# This will attach a model to it:
# Pulling the first model from the catalog
fcn_def = models["summary"][:1, "Model"]
fcn_obj = models["summary"][:1, "Object"]
mdl_type = models["summary"][:1, "Type"]
fcn_desc = models["summary"][:1, "Description"]

# This will build the rxODE2 object from the model
mk_rx_res = mk_rx_obj(
    type = mdl_type,
    model = list(fcn_def = fcn_def,
                 fcn_obj = fcn_obj))

# This will attach the model to the current element
element = MB_update_model(
    state = state,
    session = session,
    current_ele = element,
    rx_obj = mk_rx_res["capture"]["rx_obj"],
    note = fcn_desc,
    reset = True)

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

# This will fetch the current component
component = MB_fetch_component(state, element)

# You can use the component to build the code to generate the model:
gen_code =
    MB_build_code(state = state, session = session,
                   fcn_def = component["fcn_def"],
                   fcn_obj_name = "my_fcn_obj",
                   rx_obj_name = "my_obj_name")

# Model code to be included in a larger script
cat(paste0(gen_code$model_code, collapse="\n"))

# Stand-alone model code
cat(paste0(gen_code$model_code_sa, collapse="\n"))

# JMH to add:
# MB_fetch_code
**MB_del_current_element**

*Deletes Current model*

**Description**

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

**Usage**

```
MB_del_current_element(state)
```

**Arguments**

- `state` : MB state from `MB_fetch_state()`

**Value**

MB state object with the current model deleted.

**Examples**

```r
library(ruminate)
# This will get the full session:
sess_res = MB_test_mksession(session=list(), full_session=TRUE)
# This is just for CRAN
#sess_res = MB_test_mksession(session=list())
session = sess_res$session
input = sess_res$input

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

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

# This will provide a list of the available models
models = MB_fetch_catalog(state)
# This is a summary of the tables in the model:
models$summary

# This will test the models in the catalog, set as_cran
# to FALSE to test all the models.
mtres = MB_test_catalog(state, as_cran=TRUE)
```
mtres$isgood

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

# Delete the current element
state = MB_del_current_element(state)

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

# This will attach a model to it:
# Pulling the first model from the catalog
fcn_def = models["summary"][1,][["Model"]]
fcn_obj = models["summary"][1,][["Object"]]
mdl_type = models["summary"][1,][["Type"]]
fcn_desc = models["summary"][1,][["Description"]]

# This will build the rxode2 object from the model
mk_rx_res = mk_rx_obj(
    type = mdl_type,
    model = list(fcn_def = fcn_def,
                 fcn_obj = fcn_obj))

# This will attach the model to the current element
element = MB_update_model(
    state = state,
    session = session,
    current_ele = element,
    rx_obj = mk_rx_res["capture"]["rx_obj"],
    note = fcn_desc,
    reset = TRUE)

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

# This will fetch the current component
component = MB_fetch_component(state, element)

# You can use the component to build the code to generate the model:
gen_code =
    MB_build_code(state = state, session = session,
                   fcn_def = component["fcn_def"],
                   fcn_obj_name = "my_fcn_obj",
                   rx_obj_name = "my_obj_name")

# Model code to be included in a larger script
cat(paste0(gen_code$model_code, collapse="\n"))

# Stand-alone model code
cat(paste0(gen_code$model_code_sa, collapse="\n"))
# JMH to add:
# MB_fetch_code

## MB_fetch_catalog

**Fetches List of Available Models**

### Description

Creates a catalog of the models available in the system file.

### Usage

```r
MB_fetch_catalog(state)
```

### Arguments

- **state**: MB state from `MB_fetch_state()`

### Value

List with the following attributes:

- `summary`: Dataframe with a summary of the models in the catalog
- `sources`: Same information as that found in the summary table but in list form.
- `select_group`: List with the models grouped by source.
- `select_plain`: Flat list with the models (ungrouped).
- `select_subtext`: Subtext for pulldown menus.
- `msgs`: Messages to be passed back to the user.
- `hasmdl`: Boolean value indicating if any models were found.
- `isgood`: Boolean variable indicating success or failure.

### Examples

```r
library(ruminate)
# This will get the full session:
sess_res = MB_test_mksession(session=list(), full_session=TRUE)
# This is just for CRAN
# sess_res = MB_test_mksession(session=list())
session = sess_res$session
input  = sess_res$input

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

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

# This will provide a list of the available models
models = MB_fetch_catalog(state)
# This is a summary of the tables in the model:
models$summary

# This will test the models in the catalog, set as_cran
# to FALSE to test all the models.
mtres = MB_test_catalog(state, as_cran=TRUE)
mtres$isegood

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

# Delete the current element
state = MB_del_current_element(state)

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

# This will attach a model to it:
# Pulling the first model from the catalog
fcn_def = models[['summary']][1,]['Model']
fcn_obj = models[['summary']][1,]['Object']
mdl_type = models[['summary']][1,]['Type']
fcn_desc = models[['summary']][1,]['Description']

# This will build the rxode2 object from the model
mk_rx_res = mk_rx_obj(
  type = mdl_type,
  model = list(fcn_def = fcn_def,
               fcn_obj = fcn_obj))

# This will attach the model to the current element
element = MB_update_model(
  state = state,
  session = session,
  current_ele = element,
  rx_obj = mk_rx_res['capture']['rx_obj'],
  note = fcn_desc,
  reset = TRUE)

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

# This will fetch the current component
component = MB_fetch_component(state, element)
# You can use the component to build the code to generate the model:

```r
gen_code =
    MB_build_code(state = state, session = session,
                   fcn_def = component[["fcn_def"]],
                   fcn_obj_name = "my_fcn_obj",
                   rx_obj_name = "my_obj_name")
```

# Model code to be included in a larger script

```r
cat(paste0(gen_code$model_code, collapse="\n"))
```

# Stand-alone model code

```r
cat(paste0(gen_code$model_code_sa, collapse="\n"))
```

# JMH to add:
# MB_fetch_code

---

**MB_fetch_code**  
**Fetch Module Code**

### Description

Fetches the code to generate results seen in the app

### Usage

```r
MB_fetch_code(state)
```

### Arguments

- **state**: MB state from `MB_fetch_state()`

### Value

Character object vector with the lines of code

### Examples

```r
# We need a module state:
.sess_res = MB_test_mksession(session=list(), full_session=FALSE)
state = sess_res$state

code = MB_fetch_code(state)

cat(code)
```
**MB_fetch_component**  
*Fetch Selected Current Model Component*

**Description**

Fetches the selected component of the provided model.

**Usage**

```r
MB_fetch_component(state, current_ele, component_id = NULL)
```

**Arguments**

- `state`: MB state from `MB_fetch_state()`
- `current_ele`: MB model element from `MB_fetch_current_element()`
- `component_id`: The numeric component id to select (default NULL) will return the selected ID.

**Value**

A list with the current component with the following attributes:

- `isgood`: Boolean object indicating success.
- `rx_obj`: rxode2 object for the model.
- `fcn_def`: Just the model function definition.
- `note`: Note field from the components_table
- `model_code`: Code to generate model.
- `model_code_sa`: Stand-alone code to generate model with
- `msgs`: Messages to be passed back to the user.

**Examples**

```r
library(ruminate)
# This will get the full session:
sess_res = MB_test_mksession(session=list(), full_session=TRUE)
# This is just for CRAN
# sess_res = MB_test_mksession(session=list())
session = sess_res$session
input = sess_res$input

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

# Creating an empty state object
state = MB_fetch_state(id = "MB",
           input = input,
```

```r
```
session = session,
FM_yaml_file = FM_yaml_file,
MOD_yaml_file = MOD_yaml_file,
react_state = NULL)

# This will provide a list of the available models
models = MB_fetch_catalog(state)
# This is a summary of the tables in the model:
models$summary

# This will test the models in the catalog, set as_cran
# to FALSE to test all the models.
mtres = MB_test_catalog(state, as_cran=TRUE)
mtres$isgood

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

# Delete the current element
state = MB_del_current_element(state)

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

# This will attach a model to it:
# Pulling the first model from the catalog
fcn_def = models[["summary"]][1, ][["Model"]]
fcn_obj = models[["summary"]][1, ][["Object"]]
mdl_type = models[["summary"]][1, ][["Type"]]
fcn_desc = models[["summary"]][1, ][["Description"]]

# This will build the rxode2 object from the model
mk_rx_res = mk_rx_obj(
  type = mdl_type,
  model = list(fcn_def = fcn_def,
               fcn_obj = fcn_obj))

# This will attach the model to the current element
element = MB_update_model(
  state = state,
  session = session,
  current_ele = element,
  rx_obj = mk_rx_res[["capture"]][["rx_obj"]],
  note = fcn_desc,
  reset = TRUE)

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

# This will fetch the current component
component = MB_fetch_component(state, element)
# You can use the component to build the code to generate the model:

gen_code =
    MB_build_code(state = state, session = session,
                  fcn_def = component["fcn_def"],
                  fcn_obj_name = "my_fcn_obj",
                  rx_obj_name = "my_obj_name")

# Model code to be included in a larger script
cat(paste0(gen_code$model_code, collapse="\n"))

# Stand-alone model code
cat(paste0(gen_code$model_code_sa, collapse="\n"))

# JMH to add:
# MB_fetch_code

---

**MB_fetch_current_element**

*Fetches Current model*

**Description**

Takes a MB state and returns the current active model object.

**Usage**

```r
MB_fetch_current_element(state)
```

**Arguments**

- `state` MB state from `MB_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$MB$elements` in the output of `MB_fetch_state()`.

**Examples**

```r
library(ruminate)
# This will get the full session:
sess_res = MB_test_mksession(session=list(), full_session=TRUE)
# This is just for CRAN
#sess_res = MB_test_mksession(session=list())
session = sess_res$session
input = sess_res$input

# Configuration files
FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml")
MOD_yaml_file = system.file(package = "ruminate", "templates", "MB.yaml")
```
# Creating an empty state object
state = MB_fetch_state(id = "MB",
                      input = input,
                      session = session,
                      FM_yaml_file = FM_yaml_file,
                      MOD_yaml_file = MOD_yaml_file,
                      react_state = NULL)

# This will provide a list of the available models
models = MB_fetch_catalog(state)
# This is a summary of the tables in the model:
models$summary

# This will test the models in the catalog, set as_cran
# to FALSE to test all the models.
mtres = MB_test_catalog(state, as_cran=TRUE)
mtres$isgood

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

# Delete the current element
state = MB_del_current_element(state)

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

# This will attach a model to it:
# Pulling the first model from the catalog
fcn_def = models[["summary"]][1,][["Model"]]
fcn_obj = models[["summary"]][1,][["Object"]]
mdl_type = models[["summary"]][1,][["Type"]]
fcn_desc = models[["summary"]][1,][["Description"]]

# This will build the rxode2 object from the model
mk_rx_res = mk_rx_obj(
  type = mdl_type,
  model = list(fcn_def = fcn_def,
               fcn_obj = fcn_obj))

# This will attach the model to the current element
element = MB_update_model(
  state = state,
  session = session,
  current_ele = element,
  rx_obj = mk_rx_res[["capture"]][["rx_obj"]],
  note = fcn_desc,
  reset = TRUE)

# You can now place element back in the state
state = MB_set_current_element(state, element)
# This will fetch the current component
component = MB_fetch_component(state, element)

# You can use the component to build the code to generate the model:
gen_code =
    MB_build_code(state = state, session = session,
                   fcn_def = component[["fcn_def"]],
                   fcn_obj_name = "my_fcn_obj",
                   rx_obj_name = "my_obj_name")

# Model code to be included in a larger script
cat(paste0(gen_code$model_code, collapse="\n"))

# Stand-alone model code
cat(paste0(gen_code$model_code_sa, collapse="\n"))

# JMH to add:
# MB_fetch_code

---

**MB_fetch_mdl**  
*Fetch Model Builder Module Models*

**Description**

Fetches the models contained in the module.

**Usage**

```
MB_fetch_mdl(state)
```

**Arguments**

- `state`: MB state from `MB_fetch_state()`

**Value**

list containing the following elements

- `isgood`: Return status of the function.
- `hasmdl`: Boolean indicator if the module has any models
- `msgs`: Messages to be passed back to the user.
- `mdl`: List with models. Each list element has the name of the R-object for that dataset. Each element has the following structure:
  - `label`: Text label for the model (e.g. one-compartment model).
  - `MOD_TYPE`: Type of module.
  - `id`: Module ID.
  - `rx_obj`: The rxode2 object name that holds the model.
MB_fetch_state

- 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

```r
# We need a module state:
sess_res = MB_test_mksession(session=list(), full_session=FALSE)
state = sess_res$state

mdls = MB_fetch_mdl(state)

names(mdls)
```

### MB_fetch_state

**Fetch Model Builder State**

**Description**

Merges default app options with the changes made in the UI

**Usage**

```r
MB_fetch_state(
    id,
    id_ASM,
    input,
    session,
    FM_yaml_file,
    MOD_yaml_file,
    react_state
)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Shiny module ID</td>
</tr>
<tr>
<td>id_ASM</td>
<td>ID string for the app state management module used to save and load app states</td>
</tr>
<tr>
<td>input</td>
<td>Shiny input variable</td>
</tr>
<tr>
<td>session</td>
<td>Shiny session variable</td>
</tr>
<tr>
<td>FM_yaml_file</td>
<td>App configuration file with FM as main section.</td>
</tr>
<tr>
<td>MOD_yaml_file</td>
<td>Module configuration file with MC as main section.</td>
</tr>
<tr>
<td>react_state</td>
<td>Variable passed to server to allow reaction outside of module (NULL)</td>
</tr>
</tbody>
</table>
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:

- `MC`: Module components of the yaml file.
- `MB`:
  - `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 "MB"
- `id`: Character data containing the module id module in the session variable.
- `FM_yaml_file`: App configuration file with FM as main section.
- `MOD_yaml_file`: Module configuration file with MC as main section.

Examples

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

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

# Creating an empty state object
state = MB_fetch_state(id = "MB",
                       id_ASM = "ASM",
                       input = input,
                       session = session,
                       FM_yaml_file = FM_yaml_file,
                       MOD_yaml_file = MOD_yaml_file,
                       react_state = NULL)
```

---

**MB_init_state**

Initialize MB Module State

Description

Creates a list of the initialized module state

Usage

`MB_init_state(FM_yaml_file, MOD_yaml_file, id, session)`
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FM_yaml_file</td>
<td>App configuration file with FM as main section.</td>
</tr>
<tr>
<td>MOD_yaml_file</td>
<td>Module configuration file with MC as main section.</td>
</tr>
<tr>
<td>id</td>
<td>ID string for the module.</td>
</tr>
<tr>
<td>session</td>
<td>Shiny session variable</td>
</tr>
</tbody>
</table>

Value

- list containing an empty MB state

Examples

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

state = MB_init_state(
  FM_yaml_file = system.file(package = "formods",
    "templates",
    "formods.yaml"),
  MOD_yaml_file = system.file(package = "ruminate",
    "templates",
    "MB.yaml"),
  id = "MB",
  session = session)

state
```

**MB_new_element**  
New Model Building Model

Description

Appends a new empty model to the MB state object and makes this new model the active model.

Usage

```r
MB_new_element(state)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>MB state from MB_fetch_state()</td>
</tr>
</tbody>
</table>

Value

- MB state object containing a new model and that model is set as the current active model. See the help for MB_fetch_state() for model format.
Examples

library(ruminate)
# This will get the full session:
sess_res = MB_test_mksession(session=list(), full_session=TRUE)
# This is just for CRAN
# sess_res = MB_test_mksession(session=list())
session = sess_res$session
input = sess_res$input

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

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

# This will provide a list of the available models
models = MB_fetch_catalog(state)
# This is a summary of the tables in the model:
models$summary

# This will test the models in the catalog, set as_cran
# to FALSE to test all the models.
mtres = MB_test_catalog(state, as_cran=TRUE)
mtres$isgood

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

# Delete the current element
state = MB_del_current_element(state)

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

# This will attach a model to it:
# Pulling the first model from the catalog
fcn_def = models[["summary"]][1, ][["Model"],
fcn_obj = models[["summary"]][1, ][["Object"],
mdl_type = models[["summary"]][1, ][["Type"],
fcn_desc = models[["summary"]][1, ][["Description"]]

# This will build the rxode2 object from the model
mk_rx_res = mk_rx_obj(
    type = mdl_type,
    model = list(fcn_def = fcn_def,
fcn_obj = fcn_obj))

# This will attach the model to the current element
element = MB_update_model(
    state = state,
    session = session,
    current_ele = element,
    rx_obj = mk_rx_res["capture"]["rx_obj"],
    note = fcn_desc,
    reset = TRUE)

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

# This will fetch the current component
component = MB_fetch_component(state, element)

# You can use the component to build the code to generate the model:
gen_code =
    MB_build_code(state = state, session = session,
                   fcn_def = component["fcn_def"],
                   fcn_obj_name = "my_fcn_obj",
                   rx_obj_name = "my_obj_name")

# Model code to be included in a larger script
cat(paste0(gen_code$model_code, collapse="\n"))

# Stand-alone model code
cat(paste0(gen_code$model_code_sa, collapse="\n"))

# JMH to add:
# MB_fetch_code

---

**MB_Server**

**Model Builder State Server**

**Description**

Server function for the Model Builder Shiny Module

**Usage**

MB_Server(
    id,
    id_ASM = "ASM",
    FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml"),
    MOD_yaml_file = system.file(package = "ruminate", "templates", "MB.yaml"),
    deployed = FALSE,
    react_state = NULL
)
MB_set_current_element

**Arguments**

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

**Value**

- MB Server object

---

**Description**

Takes a MB state and returns the current active model.

**Usage**

```r
MB_set_current_element(state, element)
```

**Arguments**

- **state**: MB state from `MB_fetch_state()`.
- **element**: Element list from `MB_fetch_current_element()`.

**Value**

- MB state object with the current model set using the supplied value.

**Examples**

```r
library(ruminate)
# This will get the full session:
sess_res = MB_test_mksession(session=list(), full_session=TRUE)
# This is just for CRAN
# sess_res = MB_test_mksession(session=list())
session = sess_res$session
input = sess_res$input

# Configuration files
FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml")
MOD_yaml_file = system.file(package = "ruminate", "templates", "MB.yaml")
```
# Creating an empty state object
state = MB_fetch_state(id = "MB",
                       input = input,
                       session = session,
                       FM_yaml_file = FM_yaml_file,
                       MOD_yaml_file = MOD_yaml_file,
                       react_state = NULL)

# This will provide a list of the available models
models = MB_fetch_catalog(state)
# This is a summary of the tables in the model:
models$summary

# This will test the models in the catalog, set as_cran
# to FALSE to test all the models.
mtres = MB_test_catalog(state, as_cran=TRUE)
mtres$isgood

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

# Delete the current element
state = MB_del_current_element(state)

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

# This will attach a model to it:
# Pulling the first model from the catalog
fcn_def = models[["summary"]][1, ][["Model"]]
fcn_obj = models[["summary"]][1, ][["Object"]]
mdl_type = models[["summary"]][1, ][["Type"]]
fcn_desc = models[["summary"]][1, ][["Description"]]

# This will build the rxode2 object from the model
mk_rx_res = mk_rx_obj(
    type = mdl_type,
    model = list(fcn_def = fcn_def,
                 fcn_obj = fcn_obj))

# This will attach the model to the current element
element = MB_update_model(
    state = state,
    session = session,
    current_ele = element,
    rx_obj = mk_rx_res[["capture"]][["rx_obj"]],
    note = fcn_desc,
    reset = TRUE)

# You can now place element back in the state
state = MB_set_current_element(state, element)
# This will fetch the current component
component = MB_fetch_component(state, element)

# You can use the component to build the code to generate the model:
gen_code =
    MB_build_code(state = state, session = session,
                   fcn_def = component["fcn_def"],
                   fcn_obj_name = "my_fcn_obj",
                   rx_obj_name = "my_obj_name")

# Model code to be included in a larger script
cat(paste0(gen_code$model_code, collapse="\n"))

# Stand-alone model code
cat(paste0(gen_code$model_code_sa, collapse="\n"))

# JMH to add:
# MB_fetch_code

---

**MB_test_catalog**  
*Tests the Model Catalog*

---

**Description**

Reads in models in the catalog and attempts to build them.

**Usage**

```r
MB_test_catalog(state, as_cran = FALSE, verbose = TRUE)
```

**Arguments**

- **state**  
  MB state from `MB_fetch_state()`
- **as_cran**  
  Boolean to indicate if you’re running this on CRAN
- **verbose**  
  Boolean to indicate if messages should be displayed.

**Value**

List with the following attributes:

- **isgood**: Boolean variable indicating if all the models in the catalog passed the test.
- **msgs**: Messages indicating if the test was successful or not.
library(ruminate)
# This will get the full session:
sess_res = MB_test_mksession(session=list(), full_session=TRUE)
# This is just for CRAN
# sess_res = MB_test_mksession(session=list())
session = sess_res$session
input = sess_res$input

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

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

# This will provide a list of the available models
models = MB_fetch_catalog(state)
# This is a summary of the tables in the model:
models$summary

# This will test the models in the catalog, set as_cran
# to FALSE to test all the models.
mtres = MB_test_catalog(state, as_cran=TRUE)
mtres$isgood

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

# Delete the current element
state = MB_del_current_element(state)

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

# This will attach a model to it:
# Pulling the first model from the catalog
fcn_def = models["summary"][[1, ][["Model"]]
fcn_obj = models["summary"][[1, ][["Object"]]
mdl_type = models["summary"][[1, ][["Type"]]
fcn_desc = models["summary"][[1, ][["Description"]]

# This will build the rxode2 object from the model
mk_rx_res = mk_rx_obj(
  type = mdl_type,
  model = list(fcn_def = fcn_def,
fcn_obj = fcn_obj))

# This will attach the model to the current element
element = MB_update_model(
    state = state,
    session = session,
    current_ele = element,
    rx_obj = mk_rx_res["capture"]["rx_obj"],
    note = fcn_desc,
    reset = TRUE)

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

# This will fetch the current component
component = MB_fetch_component(state, element)

# You can use the component to build the code to generate the model:
gen_code =
    MB_build_code(state = state, session = session,
    fcn_def = component["fcn_def"],
    fcn_obj_name = "my_fcn_obj",
    rx_obj_name = "my_obj_name")

# Model code to be included in a larger script
    cat(paste0(gen_code$model_code, collapse="\n"))

# Stand-alone model code
    cat(paste0(gen_code$model_code_sa, collapse="\n"))

# JMH to add:
# MB_fetch_code

---

### MB_test_mksession

**Populate Session Data for Module Testing**

#### Description

Populates the supplied session variable for testing.

#### Usage

```r
MB_test_mksession(session, id = "MB", full_session = TRUE)
```

#### Arguments

- **session**
  - Shiny session variable (in app) or a list (outside of app)
- **id**
  - An ID string that corresponds with the ID used to call the modules UI elements
- **full_session**
  - Boolean to indicate if the full test session should be created (default TRUE).
\textit{MB_update_checksum}

\textbf{Value}

list with the following elements

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

\textbf{Examples}

\begin{verbatim}
(sess_res = MB_test_mksession(session=list(), full_session=FALSE))
\end{verbatim}

\begin{verbatim}
MB_update_checksum(state)
\end{verbatim}

\textbf{Description}

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

\textbf{Usage}

MB_update_checksum(state)

\textbf{Arguments}

- \texttt{state} MB state from MB\_fetch\_state()

\textbf{Value}

MB state object with the checksum updated

\textbf{Examples}

\begin{verbatim}
# Within shiny both session and input variables will exist,
# this creates examples here for testing purposes:
sess_res = MB_test_mksession(session=list())
session = sess_res$session
input = sess_res$input

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

state = MB_update_checksum(state)
\end{verbatim}
**MB_update_model**

*Updates Current Element with rxode2 Model*

**Description**

Takes an rxode2 object and updates the model components of the current element.

**Usage**

```r
MB_update_model(state, session, current_ele, rx_obj, note, reset = FALSE)
```

**Arguments**

- `state`: MB state from `MB_fetch_state()`
- `session`: Shiny session variable
- `current_ele`: MB model element from `MB_fetch_current_element()`
- `rx_obj`: rxode2 model from `rxode2::rxode2()`
- `note`: text indicating what this update does (e.g. "added parameter")
- `reset`: boolean indicating that the element needs to be reset (i.e. if you change the base model) default: `FALSE`.

**Value**

`current_element` with model attached

**Examples**

```r
library(ruminate)
# This will get the full session:
sess_res = MB_test_mksession(session=list(), full_session=TRUE)
# This is just for CRAN
#sess_res = MB_test_mksession(session=list())
session = sess_res$session
input = sess_res$input

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

# Creating an empty state object
state = MB_fetch_state(id = "MB",
    input = input,
    session = session,
    FM_yaml_file = FM_yaml_file,
    MOD_yaml_file = MOD_yaml_file,
    react_state = NULL)
```
# This will provide a list of the available models
models = MB_fetch_catalog(state)

# This is a summary of the tables in the model:
models$summary

# This will test the models in the catalog, set as_cran
# to FALSE to test all the models.
mtres = MB_test_catalog(state, as_cran=TRUE)
mtres$isgood

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

# Delete the current element
state = MB_del_current_element(state)

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

# This will attach a model to it:
# Pulling the first model from the catalog
fcn_def = models[["summary"]][1][["Model"]]
fcn_obj = models[["summary"]][1][["Object"]]
mdl_type = models[["summary"]][1][["Type"]]
fcn_desc = models[["summary"]][1][["Description"]]

# This will build the rxode2 object from the model
mk_rx_res = mk_rx_obj(
    type = mdl_type,
    model = list(fcn_def = fcn_def,
                 fcn_obj = fcn_obj))

# This will attach the model to the current element
element = MB_update_model(
    state = state,
    session = session,
    current_ele = element,
    rx_obj = mk_rx_res[["capture"]][["rx_obj"]],
    note = fcn_desc,
    reset = TRUE)

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

# This will fetch the current component
component = MB_fetch_component(state, element)

# You can use the component to build the code to generate the model:
gen_code =
    MB_build_code(state = state, session = session,
                   fcn_def = component[["fcn_def"]],
                   fcn_obj_name = "my_fcn_obj"),
# mk_figure_ind_obs

Creates Figures of Individual Observations from PKNCA Result

## Description

Takes the output of PKNCA and creates ggplot figures faceted by subject id highlighting of certain NCA aspects (e.g. points used for half-life)

## Usage

```r
mk_figure_ind_obs(
  nca_res,
  OBS_LAB = "Concentration ===CONCUNIT===",
  TIME_LAB = "Time ===TIMEUNIT===",
  OBS_STRING = "Observation",
  BLQ_STRING = "BLQ",
  NA_STRING = "Missing",
  log_scale = TRUE,
  scales = "fixed",
  nfrows = 4,
  nfcols = 3
)
```

## Arguments

- **nca_res**: Output of PKNCA.
- **OBS_LAB**: Label of the observation axis with optional ===CONCUNIT=== placeholder for units.
- **TIME_LAB**: Label of the time axis with optional ===TIMEUNIT=== placeholder for units.
- **OBS_STRING**: Label for observation data.
- **BLQ_STRING**: Label for BLQ data.
- **NA_STRING**: Label for missing data.
- **log_scale**: Boolean variable to control y-scale (TRUE: Log 10, FALSE: linear).
- **scales**: String to determine the scales used when faceting. Can be either "fixed", "free", "free_x", or "free_y".
- **nfrows**: Number of facet rows per page.
- **nfcols**: Number of facet cols per page.
**mk_rx_obj**

**Value**

List containing the element `figures` which is a list of figure pages ("Figure 1", "Figure 2", etc.). Each of these is also a list containing two elements:

- `gg`: A ggplot object for that page.
- `notes`: Placeholder for future notes, but NULL now.

**Examples**

```r
id = "NCA"
id_UD = "UD"
id_DW = "DW"
id_ASM = "ASM"

# We need a state variable to be define
sess_res = NCA_test_mksession(session=list(),
                               id = id,
                               id_UD = id_UD,
                               id_DW = id_DW,
                               id_ASM = id_ASM,
                               full_session=FALSE)

state = sess_res$state

# Pulls out the active analysis
current_ana = NCA_fetch_current_ana(state)

# This is the raw PKNCA output
pknca_res = NCA_fetch_ana_pknca(state, current_ana)

# Building the figure
mk_res = mk_figure_ind_obs(nca_res = pknca_res)

mk_res$figures$Figure_1$gg
```

---

**Description**

Makes an `rxode2 Object`

**Usage**

`mk_rx_obj(type, model)`
Arguments

- **type**: Type of supplied model can be "rxode2", "NONMEM".
- **model**: List containing the relevant information about the model. This will depend on the model types.
  - rxode2: The supplied model is in the rxode2 format.
    - `fcn_def`: Character string containing function definition.
    - `fcn_obj`: Name of the function object created in `fcn_def`.
  - NONMEM: The supplied model is in NONMEM format (either a control
    - `model_file`: Character string containing the NONMEM model file.

Value

Results of `FM_tc()` when running the model. This will include a field `isgood` which is a boolean variable indicating success or failure. See the documentation for `FM_tc()` for the format returned when evaluation results in a failure and how to address those. When successful the `capture` field will contain the following:

- `fcn_obj`: The function name.
- `rx_obj`: The built rxode2 object.

Examples

```r
fcn_def = 'my_func = function ()
{
  description <- "One compartment PK model with linear clearance"
  ini({
    1ka <- 0.45
    label("Absorption rate (Ka)")
    1cl <- 1
    label("Clearance (CL)")
    1vc <- 3.45
    label("Central volume of distribution (V)")
    propSd <- c(0, 0.5)
    label("Proportional residual error (fraction)")
  })
  model({
    ka <- exp(1ka)
    cl <- exp(1cl)
    vc <- exp(1vc)
    cp <- linCmt()
    cp ~ prop(propSd)
  })
}
'`  

`fcn_obj = "my_func"
model = list(fcn_def = fcn_def,
             fcn_obj = fcn_obj)

rx_res = mk_rx_obj("rxode2", model)`
# mk_table_ind_obs

## Description

Takes the output of PKNCA and creates a tabular view of the individual observation data. This can be spread out of over several tables (pages) if necessary.

## Usage

```r
mk_table_ind_obs(
  nca_res,
  obnd = NULL,
  not_sampled = "NS",
  blq = "BLQ",
  digits = 3,
  text_format = "text",
  max_height = 7,
  max_width = 6.5,
  max_row = NULL,
  max_col = 9,
  notes_detect = NULL,
  rows_by = "time"
)
```

## Arguments

- `nca_res`: Output of PKNCA.
- `obnd`: onbrand reporting object.
- `not_sampled`: Character string to use for missing data when pivoting.
- `blq`: Character string to use to indicate data below the level of quantification (value of 0 in the dataset).
- `digits`: Number of significant figures to report (set to `NULL` to disable rounding).
- `text_format`: Either "md" for markdown or "text" (default) for plain text.
- `max_height`: Maximum height of the final table in inches (A value of `NULL` will use 100 inches).
- `max_width`: Maximum width of the final table in inches (A value of `NULL` will use 100 inches).
max_row
Maximum number of rows to have on a page. Spillover will hang over the side of the page.

max_col
Maximum number of columns to have on a page. Spillover will be wrapped to multiple pages.

notes_detect
Vector of strings to detect in output tables (example c("NC", "BLQ").

rows_by
Can be either "time" or "id". If it is "time", there will be a column for time and separate column for each subject ID. If rows_by is set to "id" there will be a column for ID and a column for each individual time.

Value

List containing the following elements

• isgood: Boolean indicating the exit status of the function.
• one_table: Dataframe of the entire table with the first lines containing the header.
• one_body: Dataframe of the entire table (data only).
• one_header: Dataframe of the entire header (row and body, no data).
• tables: Named list of tables. Each list element is of the output
• msgs: Vector of text messages describing any errors that were found. format from build_span.

Examples

id = "NCA"
id_UD = "UD"
id_DW = "DW"
id_ASM = "ASM"

# We need a state variable to be define
sess_res = NCA_test_mksession(session=list(),
id = id,
id_UD = id_UD,
id_DW = id_DW,
id_ASM = id_ASM,
full_session=FALSE)

state = sess_res$state

# Pulls out the active analysis
current_ana = NCA_fetch_current_ana(state)

# This is the raw PKNCA output
pknca_res = NCA_fetch_ana_pknca(state, current_ana)

# Building the figure
mk_res = mk_table_ind_obs(nca_res = pknca_res)
mk_res$tables[["Table 1"]]+$ft
**mk_table_nca_params**

Create Tabular Output from PKNCA Results

**Description**

Create paginated tables from PKNCA to use in reports and Shiny apps.

**Usage**

```r
mk_table_nca_params(
  nca_res,  # Output of PKNCA.
  type = "individual",  # Type of table to generate. Can be either "individual" or "summary". [195x297]
  grouping = "interval",  # How to group columns in tables. Can be either "interval" or "parameter". [355x297]
  not_calc = "NC",  # Text string to replace NA values with to indicated values were not calculated. [418x297]
  obnd = NULL,  # onbrand reporting object. [507x297]
  nps = NULL,  # NCA parameter summary table with the following columns. [584x174]
  mult_str = "x",  # Text string to replace * values in units. [384x297]
  infinity = "inf",  # Text string to replace infinity in time intervals in column headers. [436x297]
  digits = NULL,  # Number of significant figures to report (set to NULL to disable rounding) [478x297]
  text_format = "text",  # Name used in text output. [379x112]
  notes_detect = NULL,  # Name used markdown output. [355x174]
  max_height = 7,  # Name used in latex output. [418x297]
  max_width = 6.5,  # Description: Verbose textual description of the parameter. [507x160]
  max_row = NULL,  # NCA parameter summary table with the following columns. [584x174]
  max_col = NULL   # onbrand reporting object. [6.5]
)
```

**Arguments**

- `nca_res` Output of PKNCA.
- `type` Type of table to generate. Can be either "individual" or "summary".
- `grouping` How to group columns in tables. Can be either "interval" or "parameter".
- `not_calc` Text string to replace NA values with to indicated values were not calculated.
- `obnd` onbrand reporting object.
- `nps` NCA parameter summary table with the following columns.
  - `parameter`: PKNCA Parameter name.
  - `text`: Name used in text output.
  - `md`: Name used markdown output.
  - `latex`: Name used in latex output.
  - `description`: Verbose textual description of the parameter.
- `mult_str` Text string to replace * values in units.
- `infinity` Text string to replace infinity in time intervals in column headers.
- `digits` Number of significant figures to report (set to NULL to disable rounding).
text_format

Either "md" for markdown or "text" (default) for plain text.

notes_detect

Vector of strings to detect in output tables (example c("NC", "BLQ").

max_height

Maximum height of the final table in inches (A value of NULL will use 100 inches).

max_width

Maximum width of the final table in inches (A value of NULL will use 100 inches).

max_row

Maximum number of rows to have on a page. Spillover will hang over the side of the page.

max_col

Maximum number of columns to have on a page. Spillover will be wrapped to multiple pages.

Value

list containing the following elements

- raw_nca: Raw PKNCA output.
- isgood: Boolean indicating the exit status of the function.
- one_table: Dataframe of the entire table with the first lines containing the header.
- one_body: Dataframe of the entire table (data only).
- one_header: Dataframe of the entire header (row and body, no data).
- tables: Named list of tables. Each list element is of the output.
- msgs: Vector of text messages describing any errors that were found. format from build_span.

Examples

```r
id = "NCA"
id_UD = "UD"
id_DW = "DW"
id_ASM = "ASM"

# We need a state variable to be define
sess_res = NCA_test_mksession(session=list(),
   id = id,
   id_UD = id_UD,
   id_DW = id_DW,
   id_ASM = id_ASM,
   full_session=FALSE)

state = sess_res$state

# Pulls out the active analysis
current_ana = NCA_fetch_current_ana(state)

# This is the raw PKNCA output
pknca_res = NCA_fetch_ana_pknca(state, current_ana)

# Parameter reporting details from the ruminate configuration
nps = state["NCA"]["nca_parameters"]["summary"]
```
# Building the figure
```
mk_res = mk_table_nca_params(nca_res = pknca_res, nps=nps, digits=3)
mk_res$tables[["Table 1"]]*$ft
```

---

**NCA_add_int**  
*Adds Analysis Interval to Current Analysis*

**Description**
Takes the start time, stop time, and NCA parameters and adds them to the intervals table

**Usage**
```
NCA_add_int(state, interval_start, interval_stop, nca_parameters)
```

**Arguments**
- **state**  
  NCA state from `NCA_fetch_state()`
- **interval_start**  
  Interval start time (numeric).
- **interval_stop**  
  Interval stop time (numeric).
- **nca_parameters**  
  list of NCA parameters in the interval

**Value**
State with interval added to the current analysis.

---

**NCA_append_report**  
*Append Report Elements*

**Description**
Takes an NCA state object and appends any reportable elements for the specified report type. On NCA analyses that are in a "good" state will be reported. Those not in a good state will be ignored.

**Usage**
```
NCA_append_report(state, rpt, rpttype, gen_code_only = FALSE)
```

**Arguments**
- **state**  
  NCA state from `NCA_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: Code to create report 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 = NCA_test_mksession(session=list(), full_session=FALSE)
state = sess_res$state

# here we need an empty report object for tabular data
rpt = list(summary = list(), sheets=list())

# Now we append the report indicating we want
# Excel output:
rpt_res = NCA_append_report(state,
rpt = rpt,
rpttype = "xlsx",
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"))
NCA_fetch_ana_ds

Value

NCA state with the NCA for the current analysis built.

Examples

```r
# Module IDs
id = "NCA"
id_UD = "UD"
id_DW = "DW"
id_ASM = "ASM"

# We need a module variables to be defined
sess_res = NCA_test_mksession(session=list(),
    id = id,
    id_UD = id_UD,
    id_DW = id_DW,
    id_ASM = id_ASM,
    full_session=FALSE)

state = sess_res$state

state = nca_builder(state)
```

---

NCA_fetch_ana_ds    Fetch Analysis Dataset

Description

Fetches the dataset used for the specified analysis

Usage

`NCA_fetch_ana_ds(state, current_ana)`

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>NCA state from <code>NCA_fetch_state()</code></td>
</tr>
<tr>
<td>current_ana</td>
<td>Current value in the analysis</td>
</tr>
</tbody>
</table>

Value

Dataset from the ds field of `FM_fetch_ds()`
Examples

library(ruminate)
# Module IDs
id = "NCA"
id_UD = "UD"
id_DW = "DW"
id_ASM = "ASM"
# We need session and input variables to be defined
sess_res = NCA_test_mksession(session=list(),
    id = id,
id_UD = id_UD,
id_DW = id_DW,
id_ASM = id_ASM,
    full_session=FALSE)

# Extracting the session and input variables
session = sess_res$session
input = sess_res$input
react_state = list()

# We also need configuration files
FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml")
MOD_yaml_file = system.file(package = "ruminate", "templates", "NCA.yaml")

# Getting the current module state
state = NCA_fetch_state(id = id,
    input = input,
    session = session,
    FM_yaml_file = FM_yaml_file,
    MOD_yaml_file = MOD_yaml_file,
    id_ASM = id_ASM,
    id_UD = id_UD,
    id_DW = id_DW,
    react_state = react_state)

# Pulls out the active analysis
current_ana = NCA_fetch_current_ana(state)

# This will get the dataset associated with this analysis
ds = NCA_fetch_ana_ds(state, current_ana)

# After making changes you can update those in the state
state = NCA_set_current_ana(state, current_ana)

# You can use this to check the current analysis
current_ana = NCA_process_current_ana(state)

# This will pull out the code for the module
fc_res = NCA_fetch_code(state)

# This will use patterns defined for the site to detect
# columns. In this example we are detecting the id column:
id_col = NCA_find_col(
    patterns = state[["MC"]][["detect_col"]][["id"],
    dscols = names(ds$DS))

# This creates a new analysis
state = NCA_new_ana(state)

NCA_fetch_ana_pknca  Fetch PKNCA Results Object

Description

Fetches the PKNCA output for a specified analysis

Usage

NCA_fetch_ana_pknca(state, current_ana)

Arguments

state  NCA state from NCA_fetch_state()
current_ana  Current value in the analysis

Value

Dataset from the ds field of FM_fetch_ds()

Examples

library(ruminate)
# Module IDs
id = "NCA"
id_UD = "UD"
id_DW = "DW"
id_ASM = "ASM"
# We need session and input variables to be define
sess_res = NCA_test_mksession(session=list(),
    id = id,
    id_UD = id_UD,
    id_DW = id_DW,
    id_ASM = id_ASM,
    full_session=FALSE)

# Extracting the session and input variables
session = sess_res$session
input = sess_res$input
react_state = list()

# We also need configuration files
NCA_fetch_code

Fetch Module Code

Description

Fetches the code to generate results seen in the app

Usage

NCA_fetch_code(state)
Arguments

state  NCA state from NCA_fetch_state()

Value

Character object vector with the lines of code

Examples

library(ruminate)
# Module IDs
id = "NCA"
id_UD = "UD"
id_DW = "DW"
id_ASM = "ASM"
# We need session and input variables to be define
sess_res = NCA_test_mksession(session=list(),
id = id,
id_UD = id_UD,
id_DW = id_DW,
id_ASM = id_ASM,
full_session=FALSE)

# Extracting the session and input variables
session = sess_res$session
input = sess_res$input
react_state = list()

# We also need configuration files
FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml")
MOD_yaml_file = system.file(package = "ruminate", "templates", "NCA.yaml")

# Getting the current module state
state = NCA_fetch_state(id = id,
                        input = input,
                        session = session,
                        FM_yaml_file = FM_yaml_file,
                        MOD_yaml_file = MOD_yaml_file,
                        id_ASM = id_ASM,
                        id_UD = id_UD,
                        id_DW = id_DW,
                        react_state = react_state)

# Pulls out the active analysis
current_ana = NCA_fetch_current_ana(state)

# This will get the dataset associated with this analysis
ds = NCA_fetch_ana_ds(state, current_ana)

# After making changes you can update those in the state
state = NCA_set_current_ana(state, current_ana)
# You can use this to check the current analysis

```r
current_ana = NCA_process_current_ana(state)
```

# This will pull out the code for the module

```r
fc_res = NCA_fetch_code(state)
```

# This will use patterns defined for the site to detect # columns. In this example we are detecting the id column:

```r
id_col = NCA_find_col(
  patterns = state[["MC"]][["detect_col"]][["id"],
  dscols = names(ds$DS))
```

# This creates a new analysis

```r
state = NCA_new_ana(state)
```

---

### NCA_fetch_current_ana  Fetches Current Analysis

**Description**

Takes an NCA state and returns the current active analysis

**Usage**

```r
NCA_fetch_current_ana(state)
```

**Arguments**

- `state`: NCA state from `NCA_fetch_state()`

**Value**

List containing the details of the current analysis. The structure of this list is the same as the structure of `state$NCA$anas` in the output of `NCA_fetch_state()`.

**Examples**

```r
library(ruminate)
# Module IDs
id = "NCA"
id_UD = "UD"
id_DW = "DW"
id_ASM = "ASM"
# We need session and input variables to be define
sess_res = NCA_test_mksession(session=list(),
  id = id,
  id_UD = id_UD,
  id_DW = id_DW,
  id_ASM = id_ASM,
  full_session=FALSE)
```
# Extracting the session and input variables
session = sess_res$session
input = sess_res$input
react_state = list()

# We also need configuration files
FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml")
MOD_yaml_file = system.file(package = "ruminate", "templates", "NCA.yaml")

# Getting the current module state
state = NCA_fetch_state(id = id,
                        input = input,
                        session = session,
                        FM_yaml_file = FM_yaml_file,
                        MOD_yaml_file = MOD_yaml_file,
                        id_ASM = id_ASM,
                        id_UD = id_UD,
                        id_DW = id_DW,
                        react_state = react_state)

# Pulls out the active analysis
current_ana = NCA_fetch_current_ana(state)

# This will get the dataset associated with this analysis
ds = NCA_fetch_ana_ds(state, current_ana)

# After making changes you can update those in the state
state = NCA_set_current_ana(state, current_ana)

# You can use this to check the current analysis
current_ana = NCA_process_current_ana(state)

# This will pull out the code for the module
fc_res = NCA_fetch_code(state)

# This will use patterns defined for the site to detect
# columns. In this example we are detecting the id column:
id_col = NCA_find_col(
      patterns = state[["MC"],["detect_col"],["id"],[
                        dscols = names(ds$DS)])

# This creates a new analysis
state = NCA_new_ana(state)
Description
Takes the current state and object type and returns the currently selected object. For example if you have specified figure, it will look at the output figure selected and the figure number of that figure and return the ggplot object for that. By subject id highlighting of certain NCA aspects (e.g. points used for half-life)

Usage
NCA_fetch_current_obj(state, obj_type)

Arguments
state NCA state from NCA_fetch_state()
obj_type Type of object to return (either "table" or "figure").

Value
List with a format that depends on the obj_type. For figures:
• ggplot: ggplot object of the figure.
• isgood: Return status of the function.
• msgs: Messages to be passed back to the user.
For tables:
• df: Dataframe of the current table.
• ft: Flextable object of the current table.
• notes: Any table notes to be included.
• isgood: Return status of the function.
• msgs: Messages to be passed back to the user.

Examples
# We need a state object to use below
sess_res = NCA_test_mksession(session=list(), full_session=FALSE)
state = sess_res$state

# Current active table:
res = NCA_fetch_current_obj(state, "table")
res$ft

# Current active figure:
res = NCA_fetch_current_obj(state, "figure")
res$ggplot
NCA_fetch_data_format  Fetches Details About Data Requirements

Description

Use this to get information about data formats.

Usage

NCA_fetch_data_format(
    MOD_yaml_file = system.file(package = "ruminate", "templates", "NCA.yaml")
)

Arguments

MOD_yaml_file  Module configuration file with MC as main section.

Value

List with details about the data formats

Examples

NCA_fetch_data_format()

NCA_fetch_ds  Fetch Module Datasets

Description

Fetches the datasets contained in the module

Usage

NCA_fetch_ds(state)

Arguments

state  NCA state from NCA_fetch_state()
Value

list containing the following elements

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

Examples

# We need a state object to use below
sess_res = NCA_test_mksession(session=list(), full_session=FALSE)
state = sess_res$state

myDs = NCA_fetch_ds(state)
NCA_fetch_PKNCA_meta

Value

List with the following elements:

- choices: List parameter choices grouped by values specified in the module configuration file.
- summary: Data frame with meta data about the NCA parameters with the following columns:
  - parameter: Name of parameter in PKNCA.
  - text: Name of parameter in plain text.
  - md: Parameter name formatted in Markdown.
  - latex: Parameter name formatted using LaTeX.
  - description: Verbose description in plain text for the parameter.

Examples

NCA_fetch_np_meta()

NCA_fetch_PKNCA_meta  Fetches PKNCA Metadata

Description

Compiles Metadata from PKNCA

Usage

NCA_fetch_PKNCA_meta()

Value

Dataframe containing PKCNA metadata for NCA parameters.

Examples

PKCNA_meta = NCA_fetch_PKNCA_meta()
utils::head(PKCNA_meta)
**NCA_fetch_state**  
*Fetch ruminate State*

**Description**

Merges default app options with the changes made in the UI

**Usage**

```r
NCA_fetch_state(
  id,
  input,
  session,
  FM_yaml_file,
  MOD_yaml_file,
  id_ASM,
  id_UD,
  id_DW,
  react_state
)
```

**Arguments**

- `id`: Shiny module ID
- `input`: Shiny input variable
- `session`: Shiny session variable
- `FM_yaml_file`: App configuration file with FM as main section.
- `MOD_yaml_file`: Module configuration file with MC as main section.
- `id_ASM`: ID string for the app state management module used to save and load app states
- `id_UD`: ID string for the upload data module used to save and load app states
- `id_DW`: ID string for the data wrangling module used to save and load app states
- `react_state`: Variable passed to server to allow reaction outside of module (NULL)

**Value**

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

- `MC`: Module components of the yaml file.
- `NCA`:
  - `ana_cntr`: Analysis counter.
  - `anas`: List of analyses: Each analysis has the following structure:
- ana_dsview: Dataset view/ID (name from DSV) selected as a data source for this analysis.
- ana_scenario: Analysis scenario selected in the UI
- checksum: checksum of the analysis (used to detect changes in the analysis).
- code: Code to generate analysis from start to finish or error messages if code generation/analysis failed.
- code_components: List containing the different components from code
- col_conc: Column from ana_dsview containing the concentration data.
- col_dose: Column from ana_dsview containing the dose amount.
- col_dur: Column from ana_dsview containing the infusion duration or N/A if unused.
- col_group: Columns from ana_dsview containing other grouping variables.
- col_id: Column from ana_dsview containing the subject IDs.
- col_netime: Column from ana_dsview containing the nominal time values
- col_route: Column from ana_dsview containing the dosing route.
- col_time: Column from ana_dsview containing the time values.
- id: Character id (ana_idx).
- idx: Numeric id (1).
- include_units: Boolean variable indicating in units should included in the analysis.
- interval_range: Vector with the first element representing the beginning of the interval and the second element containing the end of the interval.
- intervals: List of the intervals to include.
- isgood: Current status of the analysis.
- key: Analysis key acts as a title/caption (user editable)
- msgs: Messages generated when checking configuration and analysis options.
- nca_config: List of NCA configuration options for this analysis.
- nca_object_name: Prefix for NCA objects associated with this analysis.
- nca_parameters: NCA parameters selected for calculation in the UI.
- notes: Analysis notes (user editable)
- objs: List of names and values for objects created with generated code.
- sampling: Sampling method either "sparse" or "serial"
- units_amt: Amount units.
- units_conc: Concentration units.
- units_dose: Dosing units.
- units_time: Time units.
- current_ana: Currently selected analysis (list name element from anas).
- DSV: Available data source views (see FM_fetch_ds)
- checksum: This is an MD5 sum of the module (checksum of the analysis checksums).
- nca_config: List of PKNCA configuration options for this analysis.
- nca_parameters: List with two elements
  * choices: List consisting of "Common Parameters" and "Other" (used for grouping in the UI). Each of these is a list of text parameter names with a value of the PKNCA parameter name.
- summary: Summary table with the following columns:
- parameter: PKNCA Parameter name.
- text: Name used in text output.
- md: Name used markdown output.
- latex: Name used in latex output.
- description: Verbose textual description of the parameter.
- ui: Current value of form elements in the UI.
- ui_ana_map: Map between UI element names and analysis in the object you get from
  \texttt{NCA\_fetch\_current\_ana}
- ui_ids: Vector of UI elements for the module.
- ui_hold: List of hold elements to disable updates before a full ui refresh is complete.

- **MOD\_TYPE**: Character data containing the type of module "NCA"
- **id**: Character data containing the module id module in the session variable.
- **FM\_yaml\_file**: App configuration file with FM as main section.
- **MOD\_yaml\_file**: Module configuration file with MC as main section.

### Examples

```r
library(ruminate)
# Module IDs
id = "NCA"
id_UD = "UD"
id_DW = "DW"
id_ASM = "ASM"
# We need session and input variables to be define
sess_res = NCA_test_mksession(session=list(),
id = id,
id_UD = id_UD,
id_DW = id_DW,
id_ASM = id_ASM,
full_session=FALSE)

# Extracting the session and input variables
session = sess_res$session
input = sess_res$input
react_state = list()

# We also need configuration files
FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml")
MOD_yaml_file = system.file(package = "ruminate", "templates", "NCA.yaml")

# Getting the current module state
state = NCA_fetch_state(id = id,
input = input,
session = session,
FM_yaml_file = FM_yaml_file,
MOD_yaml_file = MOD_yaml_file,
id_ASM = id_ASM,
id_UD = id_UD,
id_DW = id_DW,
...)```
# Pulls out the active analysis
current_ana = NCA_fetch_current_ana(state)

# This will get the dataset associated with this analysis
ds = NCA_fetch_ana_ds(state, current_ana)

# After making changes you can update those in the state
state = NCA_set_current_ana(state, current_ana)

# You can use this to check the current analysis
current_ana = NCA_process_current_ana(state)

# This will pull out the code for the module
fc_res = NCA_fetch_code(state)

# This will use patterns defined for the site to detect
# columns. In this example we are detecting the id column:
id_col = NCA_find_col(
    patterns = state["MC"][["detect_col"]]["id"],
dscols = names(ds$DS))

# This creates a new analysis
state = NCA_new_ana(state)

---

**NCA_find_col**

*Determines Default Column Name*

**Description**

Based on the current analysis, value from the UI, an optional list of patterns to search, and column names from a dataset this function tries to find a default value for a column in the analysis (e.g. subject id, dose, concentration, etc).

Generally the following is done:

- If `curr_ui` has a non-NULL, non-"" value it is compared to `dscols`. If it is found there that value is returned.
- If not then the patterns are considered. If the patterns from the YAML file are not NULL they are compared sequentially to the columns names. The first match found is returned.
- If nothing is found then the first value of `dscols` is returned.

**Usage**

```r
NCA_find_col(
    curr_ana = NULL,
    curr_ui = NULL,
    patterns = NULL,
```
Arguments

curr_ana | Current value in the analysis
curr_ui  | Current value in UI
patterns | List of regular expression patterns to consider.
dscols   | Columns from the dataset.
null_ok  | Logical value indicating if a null result (nothing found) is OK (default: FALSE)

Value

Name of column found based on the rules above.

Examples

library(ruminate)
# Module IDs
id    = 'NCA'
id_UD  = 'UD'
id_DW  = 'DW'
id_ASM = 'ASM'
# We need session and input variables to be define
sess_res = NCA_test_mksession(session=list(),
   id    = id,
   id_UD  = id_UD,
   id_DW  = id_DW,
   id_ASM = id_ASM,
   full_session=FALSE)

# Extracting the session and input variables
session = sess_res$session
input   = sess_res$input
react_state = list()

# We also need configuration files
FM_yaml_file = system.file(package = 'formods', 'templates', 'formods.yaml')
MOD_yaml_file = system.file(package = 'ruminate', 'templates', 'NCA.yaml')

# Getting the current module state
state = NCA_fetch_state(id    = id,
                        input  = input,
                        session = session,
                        FM_yaml_file = FM_yaml_file,
                        MOD_yaml_file = MOD_yaml_file,
                        id_ASM   = id_ASM,
                        id_UD    = id_UD,
                        id_DW    = id_DW,
                        react_state = react_state)
# Pulls out the active analysis
current_ana = NCA_fetch_current_ana(state)

# This will get the dataset associated with this analysis
ds = NCA_fetch_ana_ds(state, current_ana)

# After making changes you can update those in the state
state = NCA_set_current_ana(state, current_ana)

# You can use this to check the current analysis
current_ana = NCA_process_current_ana(state)

# This will pull out the code for the module
fc_res = NCA_fetch_code(state)

# This will use patterns defined for the site to detect
# columns. In this example we are detecting the id column:
id_col = NCA_find_col(
    patterns = state["MC"]['detect_col']["id"],
    dscols = names(ds$DS))

# This creates a new analysis
state = NCA_new_ana(state)

---

**NCA_init_state**

**Initialize NCA Module State**

**Description**

Creates a list of the initialized module state

**Usage**

NCA_init_state(FM_yaml_file, MOD_yaml_file, id, id_UD, id_DW, session)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FM_yaml_file</td>
<td>App configuration file with FM as main section.</td>
</tr>
<tr>
<td>MOD_yaml_file</td>
<td>Module configuration file with MC as main section.</td>
</tr>
<tr>
<td>id</td>
<td>ID string for the module.</td>
</tr>
<tr>
<td>id_UD</td>
<td>ID string for the upload data module used to handle uploads or the name of</td>
</tr>
<tr>
<td></td>
<td>the list element in react_state where the data set is stored.</td>
</tr>
<tr>
<td>id_DW</td>
<td>ID string for the data wrangling module to process any uploaded data</td>
</tr>
<tr>
<td>session</td>
<td>Shiny session variable (in app) or a list (outside of app)</td>
</tr>
</tbody>
</table>

**Value**

list containing an empty NCA state
**NCA_load_scenario**  *Loads Pre-Defined Scenario*

**Description**

Loads a pre-defined analysis scenario from the NCA YAML config file.

**Usage**

```python
NCA_load_scenario(state, ana_scenario)
```

**Arguments**

- `state`: NCA state from `NCA_fetch_state()`
- `ana_scenario`: Short name of the analysis scenario to load from the config file.

**Value**

NCA state object with the scenario loaded and relevant notifications set.

**NCA_mkactive_ana**  *Fetch PKNCA Results Object*

**Description**

Fetches the PKNCA output for a specified analysis.

**Usage**

```python
NCA_mkactive_ana(state, ana_id)
```

**Arguments**

- `state`: NCA state from `NCA_fetch_state()`
- `ana_id`: Analysis ID to make active.

**Value**

State with the analysis ID made active. JMH add to example script below
Examples

```r
call(your_function)
```

```r
library(ruminate)
# Module IDs
id = "NCA"
id_UD = "UD"
id_DW = "DW"
id_ASM = "ASM"
# We need session and input variables to be define
sess_res = NCA_test_mksession(session=list(),
                   id = id,
                   id_UD = id_UD,
                   id_DW = id_DW,
                   id_ASM = id_ASM,
                   full_session=FALSE)

# Extracting the session and input variables
session = sess_res$session
input = sess_res$input
react_state = list()

# We also need configuration files
FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml")
MOD_yaml_file = system.file(package = "ruminate", "templates", "NCA.yaml")

# Getting the current module state
state = NCA_fetch_state(id = id,
                   input = input,
                   session = session,
                   FM_yaml_file = FM_yaml_file,
                   MOD_yaml_file = MOD_yaml_file,
                   id_ASM = id_ASM,
                   id_UD = id_UD,
                   id_DW = id_DW,
                   react_state = react_state)

# Pulls out the active analysis
current_ana = NCA_fetch_current_ana(state)

# This will get the dataset associated with this analysis
ds = NCA_fetch_ana_ds(state, current_ana)

# After making changes you can update those in the state
state = NCA_set_current_ana(state, current_ana)

# You can use this to check the current analysis
current_ana = NCA_process_current_ana(state)

# This will pull out the code for the module
fc_res = NCA_fetch_code(state)

# This will use patterns defined for the site to detect
# columns. In this example we are detecting the id column:
```
id_col = NCA_find_col(
    patterns = state[["MC"]]][["detect_col"]][["id"],
    dscols = names(ds$DS))

# This creates a new analysis
state = NCA_new_ana(state)

### NCA_new_ana

**Initialize New Analysis**

**Description**

Creates a new NCA analysis in an NCA module

**Usage**

NCA_new_ana(state)

**Arguments**

- **state**: NCA state from NCA_fetch_state()

**Value**

NCA state object containing a new empty analysis and that analysis is set as the current active analysis

**Examples**

library(ruminate)
# Module IDs
id = "NCA"
id_UD = "UD"
id_DW = "DW"
id_ASM = "ASM"
# We need session and input variables to be define
sess_res = NCA_test_mksession(session=list(),
    id = id,
    id_UD = id_UD,
    id_DW = id_DW,
    id_ASM = id_ASM,
    full_session=FALSE)

# Extracting the session and input variables
session = sess_res$session
input = sess_res$input
react_state = list()

# We also need configuration files
FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml")
MOD_yaml_file = system.file(package = "ruminate", "templates", "NCA.yaml")

# Getting the current module state
state = NCA_fetch_state(id = id,
                        input = input,
                        session = session,
                        FM_yaml_file = FM_yaml_file,
                        MOD_yaml_file = MOD_yaml_file,
                        id_ASM = id_ASM,
                        id_UD = id_UD,
                        id_DW = id_DW,
                        react_state = react_state)

# Pulls out the active analysis
current_ana = NCA_fetch_current_ana(state)

# This will get the dataset associated with this analysis
ds = NCA_fetch_ana_ds(state, current_ana)

# After making changes you can update those in the state
state = NCA_set_current_ana(state, current_ana)

# You can use this to check the current analysis
current_ana = NCA_process_current_ana(state)

# This will pull out the code for the module
fc_res = NCA_fetch_code(state)

# This will use patterns defined for the site to detect
# columns. In this example we are detecting the id column:
id_col = NCA_find_col(
    patterns = state[["MC"]]["detect_col"]["id"],
    dscols = names(ds$DS))

# This creates a new analysis
state = NCA_new_ana(state)

---

NCA_process_current_ana

Processes Current Analysis to be Run

Description

Takes the current analysis and checks different aspects to for any issues to make sure it’s good to go.

Usage

NCA_process_current_ana(state)
Arguments

state NCA state from NCA_fetch_state()

Value

Current analysis list with isgood and msgs set

Examples

library(ruminate)
# Module IDs
id = "NCA"
id_UD = "UD"
id_DW = "DW"
id_ASM = "ASM"
# We need session and input variables to be define
sess_res = NCA_test_mksession(session=list(),
    id = id,
    id_UD = id_UD,
    id_DW = id_DW,
    id_ASM = id_ASM,
    full_session=FALSE)

# Extracting the session and input variables
session = sess_res$session
input = sess_res$input
react_state = list()

# We also need configuration files
FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml")
MOD_yaml_file = system.file(package = "ruminate", "templates", "NCA.yaml")

# Getting the current module state
state = NCA_fetch_state(id = id,
    input = input,
    session = session,
    FM_yaml_file = FM_yaml_file,
    MOD_yaml_file = MOD_yaml_file,
    id_ASM = id_ASM,
    id_UD = id_UD,
    id_DW = id_DW,
    react_state = react_state)

# Pulls out the active analysis
current_ana = NCA_fetch_current_ana(state)

# This will get the dataset associated with this analysis
ds = NCA_fetch_ana_ds(state, current_ana)

# After making changes you can update those in the state
state = NCA_set_current_ana(state, current_ana)
# You can use this to check the current analysis
current_ana = NCA_process_current_ana(state)

# This will pull out the code for the module
fc_res = NCA_fetch_code(state)

# This will use patterns defined for the site to detect columns. In this example we are detecting the id column:
id_col = NCA_find_col(
    patterns = state["MC"]["detect_col"]["id"],
    dscols = names(ds$DS))

# This creates a new analysis
state = NCA_new_ana(state)

---

**NCA_Server**

*Fetch Non-Compartmental Analysis State*

**Description**

Merges default app options with the changes made in the UI

**Usage**

```r
NCA_Server(
    id,
    FM_yaml_file = system.file(package = "formods", "templates", "formods.yaml"),
    MOD_yaml_file = system.file(package = "ruminate", "templates", "NCA.yaml"),
    id_ASM = "ASM",
    id_UD = "UD",
    id_DW = "DW",
    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.

- **id_ASM**  
  ID string for the app state management module used to save and load app states

- **id_UD**  
  ID string for the upload data module used to save and load app states

- **id_DW**  
  ID string for the data wrangling module used to save and load app states

- **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

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:

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

Examples

```r
if(interactive()){
  # original file: inst/templates/ruminate.R
  library(formods)
  library(ruminate)

  # These are suggested packages
  library(shinydashboard)
  #library(ggpubr)
  #library(plotly)
  #library(shinybusy)
  library(prompter)
  #library-utils()

tags$style("@import url(https://use.fontawesome.com/releases/v6.4.0/css/all.css);")

  # You can copy these locally and customize them for your own needs. Simply change the assignment to the local copy you've modified.
  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")
  FG.yaml = system.file(package="formods", "templates", "FG.yaml")
  MB.yaml = system.file(package="ruminate", "templates", "MB.yaml")
  NCA.yaml = system.file(package="ruminate", "templates", "NCA.yaml")

  # Making sure that the deployed object is created
  if(!exists("deployed")){
    deployed = FALSE
    }

  # Making sure that the run_dev object is created
  if(file.exists(file.path(tempdir(), "RUMINTE_DEVELOPMENT"))){
```
run_dev = TRUE
} else{
    run_dev = FALSE
}

# If the SETUP.R file exists we source it
if(file.exists("SETUP.R")){
    source("SETUP.R")
}

# If the DEPLOYED file marker exists we set deployed to TRUE
if(file.exists("DEPLOYED")){
    deployed = TRUE
}

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

#https://fontawesome.com/icons?from=io
logo_url = "https://raw.githubusercontent.com/john-harrold/ruminate/main/man/figures/logo.png"
run_url = "https://runruminate.ubiquity.tools/
use_url = "https://useruminate.ubiquity.tools/
main_url = "https://ruminate.ubiquity.tools/
issue_url = "https://github.com/john-harrold/ruminate/issues"

intro_text = tags$p("Ruminate is a shiny module for pharmacometric data processing, visualization, and analysis. It consists of separate shiny modules that provide interfaces into common R packages and provides the underlying code. This is done to facilitate usage of those packages and to provide reproducible analyses.",
    tags$li("To find out more visit ", tags$a("ruminate.ubiquity.tools", href=main_url),""),
    tags$li("To give it a try you can download a test dataset ",
        tags$a("here", href=data_url),""),
    tags$li("Go to ", tags$a("useruminate.ubiquity.tools", href=use_url)," for a video demonstrating how to use ruminate"),
    tags$li("If you run into any problems, have questions, or want a feature please")"
NCA_Server

visit the ",
tag$a("issues", href=issue_url)," page")

ftmp = file.path(tempdir(), "ruminate.test")
# If the ftmp file is present we load the development modules
if(run_dev){
    dev_modules = shinydashboard::menuItem("Models",
        tabName = "model",
        icon = icon("trowel-bricks"))
} else {
    dev_modules = NULL
}

ui <- shinydashboard::dashboardPage(
    skin="black",
    shinydashboard::dashboardHeader(title="ruminate"),
    shinydashboard::dashboardSidebar(
        shinydashboard::sidebarMenu(
            shinydashboard::menuItem("Load/Save",
                tabName="loadsave",
                icon=icon("arrow-down-up-across-line")),
            shinydashboard::menuItem("Transform Data", tabName="wrangle", icon=icon("shuffle")),
            shinydashboard::menuItem("Visualize", tabName="plot", icon=icon("chart-line")),
            shinydashboard::menuItem("NCA", tabName="nca", icon=icon("chart-area")),
            dev_modules,
            #shinydashboard::menuItem("Models", tabName="model", icon=icon("trowel-bricks")),
            shinydashboard::menuItem("App Info", tabName="sysinfo", icon=icon("book-medical"))
        ),
    shinydashboard::dashboardBody(
        tags$head(
            tags$style(HTML(CSS))
        ),
        shinydashboard::tabItems(
            shinydashboard::tabItem(tabName="nca",
                shinydashboard::box(title="Run Non-Compartmental Analysis", width=12,
                fluidRow(prompter::use_prompt(),
                column(width=12,
                htmlOutput(NS("NCA", "NCA_ui_compact"))))
            ),
            shinydashboard::tabItem(tabName="model",
                shinydashboard::box(title="Build PK/PD Models", width=12,
                fluidRow(
                column(width=12,
                htmlOutput(NS("MB", "MB_ui_compact"))))
            ),
            shinydashboard::tabItem(tabName="loadsave",
                # shinydashboard::box(title=NULL, width=12,
                shinydashboard::tabBox(
                    width = 12,
                    title = NULL,
                    })
            )
        )
    )
)
```r
shiny::tabPanel(id="load_data",
    title=tagList(shiny::icon("file-arrow-up"),
    "Load Data"),
    fluidRow(
        column(width=6,
            div(style="display:inline-block;width:100%",
                htmlOutput(NS("UD", "ui_ud_load_data")),
                htmlOutput(NS("UD", "ui_ud_clean")),
                htmlOutput(NS("UD", "ui_ud_select_sheets")),
                htmlOutput(NS("UD", "ui_ud_text_load_result"))),
        column(width=6,
            tags$p(
                tags$img(
                    class = "wrapfig",
                    src = logo_url,
                    width = 150,
                    alt = "formods logo" ),
                intro_text
            ))
    ),
    fluidRow(
        column(width=12,
            div(style="display:inline-block;vertical-align:top",
                htmlOutput(NS("UD", "ui_ud_data_preview")))
    )))
),
shiny::tabPanel(id="save_state",
    title=tagList(shiny::icon("arrow-down-up-across-line"),
    "Save or Load Analysis"),
    fluidRow(
        column(width=5,
            div(style="display:inline-block;vertical-align:top",
                htmlOutput(NS("ASM", "ui_asm_compact")))
    )))
)
)
# ),
)
shinydashboard::tabItem(tabName="wrangle",
    shinydashboard::box(title="Transform and Create Views of Your Data", width=12,
        fluidRow(
            column(width=12,
                htmlOutput(NS("DW", "DW_ui_compact"))))
    ),
shinydashboard::tabItem(tabName="plot",
    shinydashboard::box(title="Visualize Data", width=12,
        htmlOutput(NS("FG", "FG_ui_compact"))))
shinydashboard::tabItem(tabName="sysinfo",
    shinydashboard::box(title="System Details", width=12,
        width = 12,
        title = NULL,
        fluidRow(
```
shiny::tabPanel(id="sys_packages",
    title=tagList(shiny::icon("box-open"),
        "Installed Packages"),
    htmlOutput(NS("ASM", "ui_asm_sys_packages"))
),
shiny::tabPanel(id="sys_modules",
    title=tagList(shiny::icon("cubes"),
        "Loaded Modules"),
    htmlOutput(NS("ASM", "ui_asm_sys_modules"))
),
shiny::tabPanel(id="sys_log",
    title=tagList(shiny::icon("clipboard-list"),
        "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()

    # Module IDs and the order they are needed for code generation
    mod_ids = c("UD", "DW", "FG", "NCA", "MB")

    # If the ftmptest file is present we load test data
    if(file.exists(ftmptest)){
        NCA_test_mkSession(
            session,
            id = "NCA",
            id_UD = "UD",
            id_DW = "DW",
            id_ASM = "ASM"
        )
        MB_test_mkSession(
            session,
            full_session=TRUE
        )
    }

    # Module servers
    formods::ASM_Server(id="ASM",}
NCA_set_current_ana

Sets Current Analysis

Description

Takes an NCA state and an analysis list and sets that figure list as the value for the active figure

Usage

NCA_set_current_ana(state, ana)
Arguments

state \hspace{1cm} \text{NCA state from NCA\_fetch\_state()}
ana \hspace{1cm} \text{Analysis list from NCA\_fetch\_current\_ana}

Value

State with the current analysis updated

Examples

library(ruminate)
# Module IDs
id = "NCA"
id_UD = "UD"
id_DW = "DW"
id_ASM = "ASM"
# We need session and input variables to be define
sess_res = NCA\_test\_mksession(session=list(),
id = id,
id_UD = id_UD,
id_DW = id_DW,
id_ASM = id_ASM,
full_session=FALSE)

# Extracting the session and input variables
session = sess_res$session
input = sess_res$input
react\_state = list()

# We also need configuration files
FM\_yaml\_file = system.file(package = "formods", "templates", "formods.yaml")
MOD\_yaml\_file = system.file(package = "ruminate", "templates", "NCA.yaml")

# Getting the current module state
state = NCA\_fetch\_state(id = id,
input = input,
session = session,
FM\_yaml\_file = FM\_yaml\_file,
MOD\_yaml\_file = MOD\_yaml\_file,
id_ASM = id_ASM,
id_UD = id_UD,
id_DW = id_DW,
react\_state = react\_state)

# Pulls out the active analysis
current\_ana = NCA\_fetch\_current\_ana(state)

# This will get the dataset associated with this analysis
ds = NCA\_fetch\_ana\_ds(state, current\_ana)

# After making changes you can update those in the state
state = NCA\_set\_current\_ana(state, current\_ana)
# You can use this to check the current analysis
current_ana = NCA_process_current_ana(state)

# This will pull out the code for the module
fc_res = NCA_fetch_code(state)

# This will use patterns defined for the site to detect
# columns. In this example we are detecting the id column:
id_col = NCA_find_col(
    patterns = state["MC"]["detect_col"]["id"],
    dscols = names(ds$DS))

# This creates a new analysis
state = NCA_new_ana(state)

---

NCA_test_mksession  
**Populate Session Data for Module Testing**

### Description

Populates the supplied session variable for testing.

### Usage

NCA_test_mksession(
  session,
  id = "NCA",
  id_UD = "UD",
  id_DW = "DW",
  id_ASM = "ASM",
  full_session = TRUE
)

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>Shiny session variable (in app) or a list (outside of app)</td>
</tr>
<tr>
<td>id</td>
<td>An ID string that corresponds with the ID used to call the modules UI elements</td>
</tr>
<tr>
<td>id_UD</td>
<td>An ID string that corresponds with the ID used to call the UD modules UI elements</td>
</tr>
<tr>
<td>id_DW</td>
<td>An ID string that corresponds with the ID used to call the DW modules UI elements</td>
</tr>
<tr>
<td>id_ASM</td>
<td>An ID string that corresponds with the ID used to call the ASM modules UI elements</td>
</tr>
<tr>
<td>full_session</td>
<td>Boolean to indicate if the full test session should be created (default TRUE).</td>
</tr>
</tbody>
</table>
Value

list with the following elements

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

Examples

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

Description

This is done by creating a Shiny interface to different tools for data transformation (dplyr and tidyr), plotting (ggplot2), and noncompartmental analysis (PKNCA). These results can be reported in Excel, Word or PowerPoint. The state of the app can be saved and loaded at a later date. When saved, a script is generated to reproduce the different actions in the Shiny interface.

Runs the pharmacometrics ruminate app.

Usage

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

Arguments

- host: Hostname of the server ("127.0.0.1")
- port: Port number for the app (3838)
- development: Boolean variable indicating
- server_opts: List of options (names) and their values (value) e.g. list(shiny.maxRequestSize = 30 * 1024^2).
- mksession: Boolean value, when TRUE will load test session data for app testing and will also load development modules (FALSE).
run_nca_components

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

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

See Also
https://ruminate.ubiquity.tools/

Examples
if (interactive()) {
  ruminate()
}

run_nca_components  Runs NCA for the Current Analysis

Description
Takes the current state and runs the current analysis in that state.

Usage
run_nca_components(
  state,
  components = c("nca", "fg_ind_obs", "tb_ind_obs", "tb_ind_params")
)

Arguments
state  NCA state from NCA_fetch_state()
components  List of components to run. By default it will run all of the following. If you just need to regenerate a figure based on the current nca results you can just specify that component. These are the valid components:
  • nca: Run NCA analysis
  • fg_ind_obs: Build the figure(s) with the individual observations.
  • tb_ind_obs: Build the table(s) with the individual observations.
  • tb_ind_params: Build the table(s) with the individual parameters.

Value
List with the following components:
  • isgood: Return status of the function.
  •msgs: Error messages if any issues were encountered.
  • nca_res: PKNCA results if run was successful.
Examples

# We need a state object to use below
sess_res = NCA_test_mksession(session=list(), full_session=FALSE)
state = sess_res$state

state = run_nca_components(state, components="tb_ind_params")
Index

apply_route_map, 3
build_span, 36, 38
dose_records_builder, 4
FM_fetch_ds, 53
FM_generate_report, 6, 39, 40
MB_append_report, 6
MB_build_code, 6
MB_del_current_element, 8
MB_fetch_catalog, 11
MB_fetch_code, 13
MB_fetch_component, 14
MB_fetch_current_element, 16
MB_fetch_mdl, 18
MB_fetch_state, 19
MB_init_state, 20
MB_new_element, 21
MB_Server, 23
MB_set_current_element, 24
MB_test_catalog, 26
MB_test_mksession, 28
MB_update_checksum, 29
MB_update_model, 30
mk_figure_ind_obs, 32
mk_rx_obj, 33
mk_table_ind_obs, 35
mk_table_nca_params, 37
NCA_add_int, 39
NCA_append_report, 39
nca_builder, 40
NCA_fetch_ana_ds, 41
NCA_fetch_ana_pknca, 43
NCA_fetch_code, 44
NCA_fetch_current_ana, 46, 54
NCA_fetch_current_obj, 47
NCA_fetch_data_format, 49
NCA_fetch_ds, 49
NCA_fetch_np_meta, 50
NCA_fetch_PKNCA_meta, 51
NCA_fetch_state, 52
NCA_find_col, 55
NCA_init_state, 57
NCA_load_scenario, 58
NCA_mkactive_ana, 58
NCA_new_ana, 60
NCA_process_current_ana, 61
NCA_Server, 63
NCA_set_current_ana, 69
NCA_test_mksession, 71
ruminate, 72
ruminate-package (ruminate), 72
run_nca_components, 73