Package 'ubiquity'

January 7, 2025

Type Package

Title PKPD, PBPK, and Systems Pharmacology Modeling Tools

Version 2.1.0

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

Description Complete work flow for the analysis of pharmacokinetic pharmacodynamic (PKPD), physiologically-based pharmacokinetic (PBPK) and systems pharmacology models including: creation of ordinary differential equation-based models, pooled parameter estimation, individual/population based simulations, rule-based simulations for clinical trial design and modeling assays, deployment with a customizable 'Shiny' app, and non-compartmental analysis. System-specific analysis templates can be generated and each element includes integrated reporting with 'PowerPoint' and 'Word'.

URL https://r.ubiquity.tools

SystemRequirements Perl

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

License BSD_2_clause + file LICENSE

Encoding UTF-8

Depends R (>= 4.2.0)

Imports cli, deSolve, dplyr (>= 1.0.0), digest, doParallel, flextable, foreach, ggplot2, knitr, MASS, onbrand (>= 1.0.2), optimx, PKNCA, pso, readxl, rmarkdown, rhandsontable, scales, stats, stringr, shiny,

Suggests babelmixr2, GA, GGally, gridGraphics, gridExtra, grid, officer, rxode2, webshot, ggrepel, rstudioapi, testthat

VignetteBuilder knitr

RoxygenNote 7.3.2

NeedsCompilation no

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

Repository CRAN

Date/Publication 2025-01-07 14:30:02 UTC

Contents

`build_system` *Build the System File*

Description

Builds the specified system file creating the targets for R and other languages as well as the templates for performing simulations and estimations.

Usage

```
build_system(
  system_file = "system.txt",
  distribution = "automatic",
  perlcmd = "perl",
  output_directory = file.path(".", "output"),
  temporary_directory = file.path(".", "transient"),
  verbose = TRUE,
  ubiquity_app = FALSE,
  debug = TRUE
)
```

Arguments

system_file	name of the file defining the system in the ubiquity format (default = 'system.txt'), if the file does not exist a template will be created and compiled.
distribution	indicates weather you are using a 'package' or a 'stand alone' distribution of ubiquity. If set to 'automatic' the build script will first look to see if the ubiquity R package is installed. If it is installed it will use the package. Otherwise, it will assume a "sand alone" distribution.
perlcmd	system command to run perl ("perl")
output_directory	location to store analysis outputs (file.path(".", "output"))
temporary_directory	location to templates and otehr files after building the system (file.path(".", "transient"))
verbose	enable verbose messaging (TRUE)
ubiquity_app	set to TRUE when building the system to be used with the ubiquty App (FALSE)
debug	Boolean variable indicating if debugging information should be displayed (TRUE)

Value

initialized ubiquity system object

Examples

```
fr = system_new(file_name        = "system.txt",
                system_file      = "mab_pk",
                overwrite        = TRUE,
                output_directory = tempdir())
cfg = build_system(system_file  = file.path(tempdir(), "system.txt"),
      output_directory          = file.path(tempdir(), "output"),
      temporary_directory       = tempdir())
```

---

calculate_halflife *Calculate the halflife of data*

Description

Determines the terminal halflife of a sequence of corresponding times and values with optional minimum and maximum times to censor data.

Usage

```
calculate_halflife(times = NULL, values = NULL, tmin = NULL, tmax = NULL)
```

Arguments

times	- sequence of times
values	- corresponding sequence of values
tmin	- minimum time to include (NULL)
tmax	- maximum time to include (NULL)

Value

List with the following names

- thalf Halflife in units of times above
- mod Result of lm used to fit the log transformed data
- df Dataframe with the data and predicted values at the time within tmin and tmax

Examples

```
x     = c(0:100)
y     = exp(-.1*x)
th    = calculate_halflife(times=x, values=y)
thalf = th$thalf
```

---

gg_axis *Make Pretty ggplot x- or y-Axis Log 10 Scale*

Description

used to convert the x and y-axis of a ggplot to a log 10 scale that is more visually satisfying than the ggplot default.

Usage

```
gg_axis(
  fo,
  yaxis_scale = TRUE,
  xaxis_scale = TRUE,
  ylim_min = NULL,
  ylim_max = NULL,
  xlim_min = NULL,
  xlim_max = NULL,
  x_tick_label = TRUE,
  y_tick_label = TRUE
)
```

Arguments

fo	ggplot figure object
yaxis_scale	TRUE indicates that the y-axis should be log10 scaled
xaxis_scale	TRUE indicates that the x-axis should be log10 scaled
ylim_min	set to a number to define the lower bound of the y-axis
ylim_max	set to a number to define the upper bound of the y-axis
xlim_min	set to a number to define the lower bound of the x-axis
xlim_max	set to a number to define the upper bound of the x-axis
x_tick_label	TRUE to show x tick labels, FALSE to hide the x tick labels
y_tick_label	TRUE to show y tick labels, FALSE to hide the y tick labels

Value

ggplot object with formatted axis

See Also

gg_log10_xaxis and gg_log10_yaxis

Examples

```
library("ggplot2")
df = data.frame(x = seq(0.01,10,.01),
                y = seq(0.01,10,.01)^2)
p       = ggplot(df, aes(x=x, y=y)) + geom_line()
# pretty up the axes
p       = prepare_figure(fo=p, purpose="print")
# pretty log10 y-axis
p_logy  = gg_log10_yaxis(fo=p)
# pretty log10 x-axis
p_logx  = gg_log10_xaxis(fo=p)
# pretty log10 yx-axis
p_logxy = gg_axis(fo=p)
```

---

gg_log10_xaxis *Make Pretty ggplot x-Axis Log 10 Scale*

Description

Wrapper for gg_axis to create a log 10 x-axis

Usage

```
gg_log10_xaxis(
  fo,
  xlim_min = NULL,
  xlim_max = NULL,
  y_tick_label = TRUE,
  x_tick_label = TRUE
)
```

Arguments

fo	ggplot figure object
xlim_min	set to a number to define the lower bound of the x-axis
xlim_max	set to a number to define the upper bound of the x-axis
y_tick_label	TRUE to show y tick labels, FALSE to hide the y tick labels
x_tick_label	TRUE to show x tick labels, FALSE to hide the x tick labels

Value

ggplot object with formatted axis

See Also

gg_axis and gg_log10_xaxis

Examples

```
library("ggplot2")
df = data.frame(x = seq(0.01,10,.01),
                y = seq(0.01,10,.01)^2)
p       = ggplot(df, aes(x=x, y=y)) + geom_line()
# pretty up the axes
p       = prepare_figure(fo=p, purpose="print")
# pretty log10 y-axis
p_logy  = gg_log10_yaxis(fo=p)
# pretty log10 x-axis
p_logx  = gg_log10_xaxis(fo=p)
# pretty log10 yx-axis
p_logxy = gg_axis(fo=p)
```

---

gg_log10_yaxis *Make Pretty ggplot y-Axis Log 10 Scale*

Description

Wrapper for gg_axis to create a log 10 y-axis

Usage

```
gg_log10_yaxis(
  fo,
  ylim_min = NULL,
  ylim_max = NULL,
  y_tick_label = TRUE,
  x_tick_label = TRUE
)
```

Arguments

fo	ggplot figure object
ylim_min	set to a number to define the lower bound of the y-axis
ylim_max	set to a number to define the upper bound of the y-axis
y_tick_label	TRUE to show y tick labels, FALSE to hide the y tick labels
x_tick_label	TRUE to show x tick labels, FALSE to hide the x tick labels

Value

ggplot object with formatted axis

See Also

gg_axis and gg_log10_xaxis

Examples

```
library("ggplot2")
df = data.frame(x = seq(0.01,10,.01),
                y = seq(0.01,10,.01)^2)
p       = ggplot(df, aes(x=x, y=y)) + geom_line()
# pretty up the axes
p       = prepare_figure(fo=p, purpose="print")
# pretty log10 y-axis
p_logy  = gg_log10_yaxis(fo=p)
# pretty log10 x-axis
p_logx  = gg_log10_xaxis(fo=p)
# pretty log10 yx-axis
p_logxy = gg_axis(fo=p)
```

---

linspace *Implementation of the* linspace *Function from Matlab*

Description

Creates a vector of n elements equally spaced apart.

Usage

```
linspace(a, b, n = 100)
```

Arguments

a	initial number
b	final number
n	number of elements (integer >= 2)

Value

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

Examples

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

---

logspace *Implementation of the* logspace *Function from Matlab*

Description

Creates a vector of n elements logarithmically spaced apart.

Usage

```
logspace(a, b, n = 100)
```

Arguments

a	initial number
b	final number
n	number of elements (integer >=2)

Value

vector of numbers from a to b with n logarithmically (base 10) spaced apart

Examples

```
logspace(-2, 3,20)
```

pad_string *Pad String with Spaces*

Description

Adds spaces to the beginning or end of strings until it reaches the maxlength. Used for aligning text.

Usage

```
pad_string(str, maxlength = 1, location = "beginning")
```

Arguments

str	string
maxlength	length to pad to
location	either "beginning" to pad the left or "end" to pad the right

Value

Padded string

Examples

```
pad_string("bob", maxlength=10)
pad_string("bob", maxlength=10, location="end")
```

prepare_figure *Make ggplot Figure Pretty*

Description

Takes a ggplot object and alters the line thicknesses and makes other cosmetic changes to make it more appropriate for exporting.

Usage

```
prepare_figure(
  purpose = "present",
  fo,
  y_tick_minor = FALSE,
  y_tick_major = FALSE,
  x_tick_minor = FALSE,
  x_tick_major = FALSE
)
```

Arguments

purpose	either "present" (default), "print" or "shiny"
fo	ggplot figure object
y_tick_minor	Boolean value to control grid lines
y_tick_major	Boolean value to control grid lines
x_tick_minor	Boolean value to control grid lines
x_tick_major	Boolean value to control grid lines

Value

ggplot object

Examples

```
library("ggplot2")
df = data.frame(x = seq(0.01,10,.01),
                y = seq(0.01,10,.01)^2)
p       = ggplot(df, aes(x=x, y=y)) + geom_line()
# pretty up the axes
p       = prepare_figure(fo=p, purpose="print")
# pretty log10 y-axis
p_logy  = gg_log10_yaxis(fo=p)
# pretty log10 x-axis
p_logx  = gg_log10_xaxis(fo=p)
# pretty log10 yx-axis
p_logxy = gg_axis(fo=p)
```

---

run_simulation_titrate

Simulate With Titration or Rule-Based Inputs

Description

Provides an interface to run_simulation_ubiquity to start and stop simulations and apply rules to control dosing and state-resets.

Usage

```
run_simulation_titrate(SIMINT_p, SIMINT_cfg, SIMINT_dropfirst = TRUE)
```

Arguments

SIMINT_p	list of system parameters
SIMINT_cfg	ubiquity system object
SIMINT_dropfirst	when TRUE it will drop the first sample point (prevents bolus doses from starting at 0)

Value

som

See Also

system_new_tt_rule, system_set_tt_cond and the titration vignette (vignette("Titration", package = "ubiquity"))

`run_simulation_ubiquity` *Simulate Individual Response*

Description

Controls the execution of individual simulations with deSolve using either R scripts or loadable C libraries.

Usage

```
run_simulation_ubiquity(SIMINT_parameters, SIMINT_cfg, SIMINT_dropfirst = TRUE)
```

Arguments

SIMINT_parameters	vector of parameters
SIMINT_cfg	ubiquity system object
SIMINT_dropfirst	when TRUE it will drop the first sample point (prevents bolus doses from starting at 0)

Value

The simulation output is mapped (som) is a list. time-course is stored in the simout element.

- The first column (time) contains the simulation time in the units of the simulation.
- Next there is a column for each: State, output and system parameter
- Models with covariate will contain the initial value (prefix: SIMINT_CVIC_) as well as the values at each time point
- Each static and dynamic system parameter is also passed through
- A column for each timescale is returned with a "ts." prefix.

See Also

Simulation vignette (vignette("Simulation", package = "ubiquity"))

`simulate_subjects` *Run Population Simulations*

Description

Used to run Population/Monte Carlo simulations with subjects generated from either provided variance/covariance information or a dataset.

Usage

```
simulate_subjects(
  parameters,
  cfg,
  show_progress = TRUE,
  progress_message = "Simulating Subjects:"
)
```

Arguments

`parameters`	list containing the typical value of parameters
`cfg`	ubiquity system object
`show_progress`	Boolean value controlling the display of a progress indicator (`TRUE`)
`progress_message`	text string to prepend when called from the ShinyApp

Details

Failures due to numerical instability or other integration errors will be captured within the function. Data for those subjects will be removed from the output. Their IDs will be displayed as messages and stored in the output.

For more information on setting options for population simulation see the stochastic section of the `system_set_option` help file.

Value

Mapped simulation output with individual predictions, individual parameters, and summary statistics of the parameters. The Vignettes below details on the format of the output.

See Also

Vignette on simulation (`vignette("Simulation", package = "ubiquity")`) titration (`vignette("Titration", package = "ubiquity")`) as well as `som_to_df`

som_to_df *Converts the Wide/Verbose Output Simulation Functions into Data Frames*

Description

The functions run_simulation_ubiquity, simulate_subjects, or run_simulation_titrate provide outputs in a more structured format, but it may be useful to convert this "wide" format to a tall/skinny format.

Usage

```
som_to_df(cfg, som)
```

Arguments

cfg	ubiquity system object
som	simulation output from run_simulation_ubiquity, simulate_subjects, or run_simulation_titrate

Value

Data frame of the format:

When applied to the output of run_simulation_ubiquity or run_simulation_titrate

- ts.time - timescale of the system
- ts.ts1, ... ts.tsn - timescales defined in the system (<TS>)
- pred - predicted/simulated response
- tt.ti1.x - titration event information (*)
- name - state or output (<O>) name corresponding to the prediction

When applied to the output of simulate_subjects

- ID - subject ID
- ts.time - timescale of the system
- ts.ts1, ... ts.tsn - timescales defined in the system (<TS>)
- pred - predicted/simulated response
- tt.ti1.x - titration event information (*)
- P1, P2, ... Pn - system parameters for the subject (<P>)
- name - state or output (<O>) name corresponding to the prediction

(* - field present when titration is enabled)

See Also

run_simulation_titrate internally when running simulations.

system_check_requirements *Check For Perl and C Tools*

Description

Check the local installation for perl and verify C compiler is installed and working.

Usage

```
system_check_requirements(
 checklist = list(perl = list(check = TRUE, perlcmd = "perl"), C = list(check = TRUE)),
  verbose = TRUE
)
```

Arguments

checklist	list with names corresponding to elements of the system to check.
verbose	enable verbose messaging

Value

List fn result of all packages

Examples

```
invisible(system_check_requirements())
```

system_check_steady_state *Verify System Steady State*

Description

Takes the ubiquity system object and other optional inputs to verify the system is running at steady state. This also provides information that can be helpful in debugging systems not running at steady state.

Usage

```
system_check_steady_state(
  cfg,
  parameters = NULL,
  zero_rates = TRUE,
  zero_bolus = TRUE,
  output_times = seq(0, 100, 1),
  offset_tol = .Machine$double.eps * 100,
  derivative_tol = .Machine$double.eps * 100,
  derivative_time = 0
)
```

Arguments

cfg	ubiquity system object
parameters	optional set of parameters (NULL) to check at steady state (if set to NULL then the parameters for the currently selected parameter set will be used)
zero_rates	Boolean value to control removing all rate inputs (TRUE)
zero_bolus	Boolean value to control removing all bolus inputs (TRUE)
output_times	sequence of output times to simulate for offset determination (seq(0,100,1))
offset_tol	maximum percent offset to be considered zero (.Machine$double.eps*100)
derivative_tol	maximum derivative value to be considered zero (.Machine$double.eps*100)
derivative_time	time to evaluate derivatives to identify deviations (0), set to NULL to skip derivative evaluation

Value

List with the following names

- steady_state Boolean indicating weather the system was at steady state
- states_derivative Derivatives that had values greater than the derivative_tol
- states_simulation States that had values greater than the offset_tol
- som Simulated output
- derivatives Derivatives
- states_derivative_NA_NaN States that had derivatives that evaluated as either NA or NaN
- states_simulation_NA_NaN States with simulation values that had either NA or NaN
- derivative_tc Data frame with the timecourse of states where the derivative was found to be greater than tolerance (states_derivative)

system_clear_cohorts *Clear all Cohorts*

Description

Clear previously defined cohorts

Usage

```
system_clear_cohorts(cfg)
```

Arguments

cfg	ubiquity system object

Value

ubiquity system object with no cohorts defined

system_define_cohort *Define Estimation Cohort*

Description

Define a cohort to include in a parameter estimation

Usage

```
system_define_cohort(cfg, cohort)
```

Arguments

cfg	ubiquity system object
cohort	list with cohort information

Details

Each cohort has a name (eg d5mpk), and the dataset containing the information for this cohort is identified (the name defined in system_load_data)

```
cohort = list(
  name         = "d5mpk",
  dataset      = "pm_data",
  inputs       = NULL,
  outputs      = NULL)
```

Next if only a portion of the dataset applies to the current cohort, you can define a filter (cf field). This will be applied to the dataset to only return values relevant to this cohort. For example, if we only want records where the column DOSE is 5 (for the 5 mpk cohort). We can use the following:

```
cohort[["cf"]]   = list(DOSE   = c(5))
```

If the dataset has the headings ID, DOSE and SEX and cohort filter had the following format:

```
cohort[["cf"]]   = list(ID    = c(1:4),
                        DOSE  = c(5,10),
                        SEX   = c(1))
```

It would be translated into the boolean filter:

```
(ID==1) | (ID==2) | (ID==3) | (ID==4)) & ((DOSE == 5) | (DOSE==10)) & (SEX == 1)
```

Optionally you may want to fix a system parameter to a different value for a given cohort. This can be done using the cohort parameter (cp) field. For example if you had the body weight defined as a system parameter (BW), and you wanted to fix the body weight to 70 for the current cohort you would do the following:

```
cohort[["cp"]]   = list(BW        = c(70))
```

Note that you can only fix parameters that are not being estimated.

By default the underlying simulation output times will be taken from the general output_times option (see system_set_option). However It may also be necessary to specify simulation output times for a specific cohort. The output_times field can be used for this. Simply provide a vector of output times:

```
cohort[["output_times"]]   = seq(0,100,2)
```

Next we define the dosing for this cohort. It is only necessary to define those inputs that are non-zero. So if the data here were generated from animals given a single 5 mpk IV at time 0. Bolus dosing is defined using <B:times> and <B:events>. If Cp is the central compartment, you would pass this information to the cohort in the following manner:

```
cohort[["inputs"]][["bolus"]] = list()
cohort[["inputs"]][["bolus"]][["Cp"]] = list(TIME=NULL, AMT=NULL)
cohort[["inputs"]][["bolus"]][["Cp"]][["TIME"]] = c( 0)
cohort[["inputs"]][["bolus"]][["Cp"]][["AMT"]]  = c( 5)
```

Inputs can also include any infusion rates (infusion_rates) or covariates (covariates). Covariates will have the default value specified in the system file unless overwritten here. The units here are the same as those in the system file

Next we need to map the outputs in the model to the observation data in the dataset. Under the outputs field there is a field for each output. Here the field ONAME can be replaced with something more useful (like PK).

```
cohort[["outputs"]][["ONAME"]] = list()
```

If you want to further filter the dataset. Say for example you have two outputs and the `cf` applied above reduces your dataset down to both outputs. Here you can use the "of" field to apply an "output filter" to further filter the records down to those that apply to the current output ONAME.

```
cohort[["outputs"]][["ONAME"]][["of"]] = list(
       COLNAME          = c(),
       COLNAME          = c())
```

If you do not need further filtering of data, you can you can just omit the field.

Next you need to identify the columns in the dataset that contain your times and observations. This is found in the `obs` field for the current observation:

```
cohort[["outputs"]][["ONAME"]][["obs"]] = list(
         time           = "TIMECOL",
         value          = "OBSCOL",
         missing        = -1)
```

The times and observations in the dataset are found in the `'TIMECOL'` column and the `'OBSCOL'` column (optional missing data option specified by -1).

These observations in the dataset need to be mapped to the appropriate elements of your model defined in the system file. This is done with the `model` field:

```
cohort[["outputs"]][["ONAME"]][["model"]] = list(
         time           = "TS",
         value          = "MODOUTPUT",
         variance       = "PRED^2")
```

First the system time scale indicated by the `TS` placeholder above must be specfied. The time scale must correspond to the data found in `TIMECOL` above. Next the model output indicated by the `MODOUTPUT` placeholder needs to be specified. This is defined in the system file using `<O>` and should correspond to `OBSCOL` from the dataset. Lastly the `variance` field specifies the variance model. You can use the keyword `PRED` (the model predicted output) and any variance parameters. Some examples include:

- `variance = "1"` - Least squares
- `variance = "PRED^2"` - Weighted least squares proportional to the prediction squared
- `variance = "(SLOPE*PRED)^2"` Maximum likelihood estimation where `SLOPE` is defined as a variance parameter (`<VP>`)

The following controls the plotting aspects associated with this output. The color, shape and line values are the values used by ggplot functions.

```
cohort[["outputs"]][["ONAME"]][["options"]] = list(
        marker_color   = "black",
        marker_shape   = 16,
        marker_line    = 1 )
```

If the cohort has multiple outputs, simply repeat the process above for the. additional cohorts. The estimation vignettes contains examples of this.

Note: Output names should be consistent between cohorts so they will be grouped together when plotting results.

Value

ubiquity system object with cohort defined

See Also

Estimation vignette (`vignette("Estimation", package = "ubiquity")`)

`system_define_cohorts_nm`

Define Cohorts from NONMEM Input File

Description

This function allows the user to define cohorts automatically from a NONMEM dataset

Usage

```
system_define_cohorts_nm(
  cfg,
  DS = "DSNAME",
  col_ID = "ID",
  col_CMT = "CMT",
  col_DV = "DV",
  col_TIME = "TIME",
  col_AMT = "AMT",
  col_RATE = "RATE",
  col_EVID = "EVID",
  col_GROUP = NULL,
  filter = NULL,
  INPUTS = NULL,
  OBS = NULL
)
```

Arguments

`cfg`	ubiquity system object
`DS`	Name of the dataset loaded using `system_load_data`
`col_ID`	Column of unique subject identifier
`col_CMT`	Compartment column
`col_DV`	Column with observations or `'.'` for input

col_TIME	Column with system time of each record
col_AMT	Infusion/dose amounts (these need to be in the same units specified in the system.txt file)
col_RATE	Rate of infusion or '.' for bolus
col_EVID	EVID (0 - observation, 1 dose)
col_GROUP	Column name to use for defining similar cohorts when generating figures.
filter	List used to filter the dataset or NULL if the whole dataset is to be used (see filter rules or nm_select_records or a description of how to use this option)
INPUTS	List mapping input information in the dataset to names used in the system.txt file
OBS	List mapping obseravation information in the dataset to nams used in the system.txt file

Details

NOTE: to use this function it is necessary that a timescale be define for the system time scale. For example, if the system time scale was days, something like the following is needed:

```
<TS:days> 1
```

Include all records in the dataset

```
filter = NULL
```

Include only records matching the following filter

```
filter = list()
filter$COLNAME = c()
```

Mapping information:

The inputs mapping information (INPUTMAP) is alist with a field for each type of input: input:

- bolus List with a name for each bolus state in the dataset (<B:?>): each bolus name should have a CMT_NUM field indicating the compartment number for that state
- infusion_rates List with a name for each rate in the dataset (<R:?>): each rate name should have a CMT_NUM field indicating the compartment number for that state
- covariates List with for each covariate in the dataset (<CV:?>): each covariate name should have a col_COV indicating the column in the database that contains that covariate

From a coding perspective it looks like this:

```
INPUTMAP = list()
INPUTMAP$bolus$SPECIES$CMT_NUM           =  1
INPUTMAP$infusion_rates$RATE$CMT_NUM     =  1
INPUTMAP$covariates$CVNAME$col_COV       = 'CNAME'
```

The observation mapping information (OBSMAP) is a list with elements for each output as described in for system_define_cohort. Each output is a list with the following names:

- variance Variance model for this output
- CMT Compartment number mapping observations for this output
- output Name of the output (<O>) corresponding with the observations
- missing Value indicating a missing observation or NULL

From a coding perspective it looks like this:

```
OBSMAP = list()
OBSMAP$ONAME=list(variance     = 'PRED^2',
                  CMT          =  1,
                  output       = '<O>',
                  missing      =  NULL )
```

Value

ubiquity system object with cohorts defined.

See Also

Estimation vignette (vignette("Estimation", package = "ubiquity"))

---

system_estimate_parameters

Control Estimation Process

Description

Manages the flow of parameter estimation using data specified with system_define_cohort.

Usage

```
system_estimate_parameters(
  cfg,
  flowctl = "plot guess",
  analysis_name = "my_analysis",
  archive_results = TRUE
)
```

Arguments

cfg	ubiquity system object
flowctl	string to control what the flow of the function
analysis_name	string containing the name of the analysis
archive_results	boolean variable to control whether results will be archived

Details

The `flowctl` argument can have the following values

- `"plot guess"` return the initial guess
- `"estimate"` perform estimation
- `"previous estimate as guess"` load previous estimate for `analysis_name` and use that as the initial guess
- `"plot previous estimate"` return the previous estimate for `analysis_name`

Value

parameter estimates

---

system_fetch_guess *Fetch Current Parameter Guesses*

Description

Fetch a list of the guesses for the current parameter set and parameters selected for estimation

Usage

```
system_fetch_guess(cfg)
```

Arguments

`cfg`	ubiquity system object

Value

list of current parameter gauesses

---

system_fetch_iiv *Fetch Variability Terms*

Description

Extract elements of the current variance/covariance matrix specified in the system file with `<IIV:?:?>` ?, `<IIVCOR:?:?>`?, `<IIVSET:?:?>` ?, `<IIVCORSET:?:?>`?

Usage

```
system_fetch_iiv(cfg, IIV1, IIV2)
```

Arguments

cfg	ubiquity system object
IIV1	row name of the variance/covariance matrix
IIV2	column name of the variance/covariance matrix

Value

Value from the variance/covariance matrix

See Also

system_set_iiv

Examples

```
# Creating a system file from the mab_pk example
fr = system_new(file_name        = "system.txt",
                system_file      = "mab_pk",
                overwrite        = TRUE,
                output_directory = tempdir())

# Building the system
cfg = build_system(system_file  = file.path(tempdir(), "system.txt"),
      output_directory          = file.path(tempdir(), "output"),
      temporary_directory       = tempdir())

# Covariance term for ETACL and ETAVc
val = system_fetch_iiv(cfg, IIV1="ETACL", IIV2="ETAVc")
```

---

system_fetch_nca *Fetch NCA Results*

Description

Fetches the NCA summary from the ubiquity system object.

Usage

```
system_fetch_nca(cfg, analysis_name = "analysis")
```

Arguments

cfg	ubiquity system object
analysis_name	string containing the name of the NCA analysis (default 'analysis')

Value

List with a data frame of the NCA results (NCA_summary), the raw output from PKNCA (PKNCA_results), and also a list element indicating the overall success of the function call (isgood)

See Also

Vignette on NCA (vignette("NCA", package = "ubiquity"))

---

system_fetch_nca_columns

Columns in NCA Analysis

Description

Show the columns available in a given NCA analysis

Usage

```
system_fetch_nca_columns(cfg, analysis_name = "analysis")
```

Arguments

cfg	ubiquity system object
analysis_name	string containing the name of the NCA analysis (default 'analysis')

Value

list with the following elements:

- isgood Boolean variable to identify if the function executed properly (TRUE) or if there were any errors (FALSE)
- NCA_col_summary dataframe with the columns from the analysis in analysis_name (col_name - NCA short name, from - where the parameter was derived from, label - verbose text label for the column, and description, verbose text description of the parameter.
- len_NCA_col maximum length of the col_name column
- len_from maximum length of the from column
- len_label maximum length of the label column
- len_description maximum length of the description column

See Also

Vignette on NCA (system_nca_parameters_meta)

system_fetch_parameters *Fetch System Parameters*

Description

Fetch the parameters of the currently selected parameter set. To switch between parameter sets use system_select_set

Usage

```
system_fetch_parameters(cfg)
```

Arguments

cfg	ubiquity system object

Value

List of parameters for the selected parameter set

Examples

```
# Creating a system file from the mab_pk example
fr = system_new(file_name        = "system.txt",
                system_file      = "mab_pk",
                overwrite        = TRUE,
                output_directory = tempdir())

# Building the system
cfg = build_system(system_file  = file.path(tempdir(), "system.txt"),
      output_directory          = file.path(tempdir(), "output"),
      temporary_directory       = tempdir())

# Fetching the default parameter set
parameters = system_fetch_parameters(cfg)
```

system_fetch_rpt_officer_object *Extracts the officer Object From the Specified ubiquity Report*

Description

This will extract an officer object from the ubiqiuty system object for the specified report name.

Usage

```
system_fetch_rpt_officer_object(cfg, rptname = "default")
```

Arguments

cfg	ubiquity system object
rptname	ubiquity report name

Value

officer report object

See Also

system_set_rpt_officer_object

---

system_fetch_rpt_onbrand_object

Extracts the onbrand Object From the Specified ubiquity Report

---

Description

This will extract an onbrand object from the ubiqiuty system object for the specified report name.

Usage

```
system_fetch_rpt_onbrand_object(cfg, rptname = "default")
```

Arguments

cfg	ubiquity system object
rptname	ubiquity report name

Value

onbrand report object

See Also

system_set_rpt_onbrand_object

system_fetch_set *Fetch Mathematical Set*

Description

Fetch the elements of the specified mathematical set that was defined in the system file.

Usage

```
system_fetch_set(cfg, set_name = NULL)
```

Arguments

cfg	ubiquity system object
set_name	name of mathematical set

Value

A sequence containing the elements of the parameter set or NULL if if there was a problem.

Examples

```
# Creating a system file from the pbpk example
fr = system_new(file_name        = "system.txt",
                system_file      = "pbpk",
                overwrite        = TRUE,
                output_directory = tempdir())

# Building the system
cfg = build_system(system_file  = file.path(tempdir(), "system.txt"),
      output_directory          = file.path(tempdir(), "output"),
      temporary_directory       = tempdir())

# Fetching the contents of the ORG mathematical set
ORG_elements = system_fetch_set(cfg, "ORG")
```

system_fetch_template *Create New Analysis Template*

Description

Building a system file will produce templates for R and other languages. This function provides a method to make local copies of these templates.

Usage

```
system_fetch_template(
  cfg,
  template = "Simulation",
  overwrite = FALSE,
  output_directory = getwd()
)
```

Arguments

cfg	ubiquity system object
template	template type
overwrite	if TRUE the new system file will overwrite any existing files present
output_directory	directory where workshop files will be placed (getwd())

Details

The template argument can have the following values for the R workflow:

- "Simulation" produces `analysis_simulate.R`: R-Script named with placeholders used to run simulations
- "Estimation" produces `analysis_estimate.R`: R-Script named with placeholders used to perform naive-pooled parameter estimation
- "NCA" produces `analysis_nca.R`: R-Script to perform non-compartmental analysis (NCA) and report out the results
- "ShinyApp" produces `ubiquity_app.R`, `server.R` and `ui.R`: files needed to run the model through a Shiny App either locally or on a Shiny Server
- "Model Diagram" produces `system.svg`: SVG template for producing a model diagram (Goto https://inkscape.org for a free SVG editor)
- "Shiny Rmd Report" produces `system_report.Rmd` and `test_system_report.R`: R-Markdown file used to generate report tabs for the Shiny App and a script to test it

And this will create files to use in other software:

- "Adapt" produces `system_adapt.for` and `system_adapt.prm`: Fortran and parameter files for the currently selected parameter set in Adapt format.
- "Berkeley Madonna" produces `system_berkeley_madonna.txt`: text file with the model and the currently selected parameter set in Berkeley Madonna format
- "nlmixr" produces `system_nlmixr.R` For the currently selected parameter set to define the system in the ‘nlmixr‘ format.
- "NONMEM" produces `system_nonmem.ctl` For the currently selected parameter set as a NONMEM conntrol stream.
- "Monolix" produces `system_monolix.txt` and `system_monolix.mlxtran` For the currently selected parameter set.
- "mrgsolve" produces `system_mrgsolve.cpp`: text file with the model and the currently selected parameter set in mrgsolve format

Value

List with vectors of template `sources`, `destinations` and corresponding write success (`write_file`), also a list element indicating the overall success of the function call (`isgood`)

Examples

```
# Creating a system file from the mab_pk example
fr = system_new(file_name        = "system.txt",
                system_file      = "mab_pk",
                overwrite        = TRUE,
                output_directory = tempdir())

# Building the system
cfg = build_system(system_file  = file.path(tempdir(), "system.txt"),
      output_directory          = file.path(tempdir(), "output"),
      temporary_directory       = tempdir())

# Creating a simulation template
fr =  system_fetch_template(cfg,
      template         = "Simulation",
      output_directory = tempdir())
```

---

system_fetch_TSsys *Fetch System Timescale*

Description

Reads through the system information and tries to determine the system time scale (the timescale that has a value of 1)

Usage

```
system_fetch_TSsys(cfg)
```

Arguments

`cfg`	ubiquity system object

Value

Name of the system timescale or `NULL` if it was not found

system_glp_init *Initialize GLP study design*

Description

Creates a new GLP study design

Usage

```
system_glp_init(cfg, study_title = "Study Title", study_name = "default")
```

Arguments

cfg	ubiquity system object
study_title	String containing descriptive information about the study
study_name	short name used to identify the study in other functions ("default")

Value

cfg ubiquity system object with the study initialized

system_glp_scenario *Design GLP Study For a Scenario*

Description

Identifies the top dose required in a GLP tox study in order to match human metrics (Cmax and AUCs) within a specified multiplier.

For a given set of human parameters the human doses required to hit the target Cmin and AUC (both or one) will be identified. The Cmax and AUC associated with the largest of those doses will be determined and the corresponding doses for a tox species (and provided parameters) will be determined for specific tox multipliers.

Optionally, simulations can be be run by specifying doses for either/or the human or tox species. Sample times can also be specified to generate annotated figures and tables to be given to analysts to facilitate assay design.

The system file requires the following components:

- Output for the drug concentration - Output for the cumulative AUC - Bolus dosing defined in a specific compartment - Timescale specified for the system timescale (e.g. if the timescale is hours then you need <TS> hours = 1.0)

Usage

```
system_glp_scenario(
  cfg,
  output_Conc = NULL,
  output_AUC = NULL,
  timescale = NULL,
  units_Conc = "",
  units_AUC = "",
  study_scenario = "Tox Study",
  human_sim_times = NULL,
  study_name = "default",
  human_parameters = NULL,
  human_bolus = NULL,
  human_ndose = 1,
  human_dose_interval = 1,
  human_Cmin = NULL,
  human_AUC = NULL,
  human_sample_interval = NULL,
  human_sim_doses = NULL,
  human_sim_samples = NULL,
  tox_species = "Tox",
  tox_sim_times = NULL,
  tox_parameters = NULL,
  tox_bolus = NULL,
  tox_ndose = 1,
  tox_dose_interval = 1,
  tox_Cmax_multiple = 10,
  tox_AUC_multiple = 10,
  tox_sample_interval = NULL,
  tox_sim_doses = NULL,
  tox_sim_samples = NULL,
  annotate_plots = TRUE
)
```

Arguments

cfg	ubiquity system object
output_Conc	model output specified with <O> containing the concentration associated with drug exposure.
output_AUC	model output specified with <O> containing the cumulative exposure
timescale	system timescale specified with <TS> used for AUC comparisons and plotting
units_Conc	units of concentration ('')
units_AUC	units of AUC ('')
study_scenario	string containing a descriptive name for the tox study
human_sim_times	user-specified simulation output times for humans (same timescale as the system)

study_name — name of the study to append the scenario to set with 'system_glp_init()' ('default'): When a report is initialized using system_rpt_read_template the report name is 'default' unless otherwise specified. To disable reporting set this to NULL, and to use a different report specify the name here.

human_parameters — list containing the human parameters

human_bolus — string containing the dosing state for human doses (specified with <B:?>)

human_ndose — number of human doses to simulate

human_dose_interval — dosing interval in humans (time units specified with <B:?>)

human_Cmin — target Cmin in humans (corresponding to output_Conc above)

human_AUC — target AUC in humans (corresponding to output_AUC above)

human_sample_interval — time interval in units specified by timescale above to evaluate the trough concentration and AUC (e.g c(1.99, 4.001) would consider the interval between 2 and 4)

human_sim_doses — optional list of doses into human_bolus to simulate (see Details below)

human_sim_samples — optional list of sample times in units specified by timescale above to label on plots of simulated doses (the default NULL will disable labels)

tox_species — optional name of the tox species ("Tox")

tox_sim_times — user-specified simulation output times for the tox species (same timescale as the system)

tox_parameters — list containing the parameters for the tox species

tox_bolus — string containing the dosing state for tox species doses (specified with <B:?>)

tox_ndose — number of tox doses to simulate

tox_dose_interval — dosing interval in the tox species (time units specified with <B:?>)

tox_Cmax_multiple — for each target (Cmin and AUC) the dose in the tox species will be found to cover this multiple over the projected Cmax in humans (10)

tox_AUC_multiple — for each target (Cmin and AUC) the dose in the tox species will be found to cover this multiple over the projected AUC in humans (10)

tox_sample_interval — interval to consider the AUC and Cmax for comparing the human prediction to the tox multiple

tox_sim_doses — optional list of doses into tox_bolus to simulate (see Details below)

tox_sim_samples — optional list of sample times in units specified by timescale above to label on plots of simulated doses (the default NULL will disable labels)

annotate_plots — Boolean switch to indicate if human_sim_samples and tox_sim_samples should be labeled on their respective plots (TRUE)

Details

Both `human_sim_doses` and `tox_sim_doses` are lists with names corresponding to the label of the dose. Each element has an AMT and TIME element which corresponds to the dosing times and amounts in the units specified with `<B:?>` in the system file.

For example if you wanted to simulate four weekly doses of 20 mg to a 70 kg person and the units of bolus doses were days and mg/kg for the times and amounts you would do the following:

```
human_sim_doses = list()
human_sim_doses[["20 mg QW"]]$TIME = c(     0,      7,     14,     21)
human_sim_doses[["20 mg QW"]]$AMT  = c(0.2857, 0.2857, 0.2857, 0.2857)
```

Value

cfg ubiquity system object with the scenario added if successful

---

system_load_data *Loading Datasets*

Description

Loads datasets at the scripting level from a variable if `data_file` is a data.frame or from the following formats (based on the file extension)

- csv - comma delimited
- tab - tab delimited
- xls or xlsx - excel spread sheet

Multiple datasets can be loaded as long as they are given different names. Datasets should be in a NONMEM-ish format with the first row containing the column header names.

Usage

```
system_load_data(cfg, dsname, data_file, data_sheet)
```

Arguments

`cfg`	ubiquity system object
`dsname`	short name of the dataset to be used to link this dataset to different operations
`data_file`	the file name of the dataset or a data frame containing the data
`data_sheet`	argument identifying the name of the sheet in an excel file

Value

Ubiquity system object with the dataset loaded

system_log_debug_save *Save variables to files*

Description

Triggered when debugging is enabled, this function will save the contents of values to the specified file name in the ubiquity temporary directory.

Usage

```
system_log_debug_save(cfg, file_name = "my_file", values = NULL)
```

Arguments

cfg	ubiquity system object
file_name	name of the save file without the ".RData" extension
values	named list of variables to save

Value

Boolean variable indicating success

Examples

```
# Creating a system file from the mab_pk example
fr = system_new(file_name        = "system.txt",
                system_file      = "mab_pk",
                overwrite        = TRUE,
                output_directory = tempdir())

# Building the system
cfg = build_system(system_file  = file.path(tempdir(), "system.txt"),
      output_directory          = file.path(tempdir(), "output"),
      temporary_directory       = tempdir())

# enable debugging:
cfg=system_set_option(cfg,group = "logging",
                          option = "debug",
                          value  = TRUE)

# Saving the cfg variable
system_log_debug_save(cfg,
   file_name = 'my_file',
   values = list(cfg=cfg))
```

system_log_init *Initialize System Log File*

Description

Initializes the currently specified system log file.

Usage

```
system_log_init(cfg)
```

Arguments

cfg	ubiquity system object

Value

ubiquity system object with logging enabled

Examples

```
# Creating a system file from the mab_pk example
fr = system_new(file_name        = "system.txt",
                system_file      = "mab_pk",
                overwrite        = TRUE,
                output_directory = tempdir())

# Building the system
cfg = build_system(system_file  = file.path(tempdir(), "system.txt"),
      output_directory          = file.path(tempdir(), "output"),
      temporary_directory       = tempdir())

# Initialzing the log file
cfg = system_log_init(cfg)
```

system_nca_parameters_meta *List NCA parameters, text names and descriptions*

Description

Provides a verbose information about NCA parameters

Usage

```
system_nca_parameters_meta(cfg)
```

Arguments

cfg	ubiquity system object

Value

List with the following elements:

- isgood Boolean value indicating the success of the function call.
- parameters List with element names for each standard column header for NCA output. Each element name is a list with the following elements:
  - label Textual descriptor of the parameter.
  - description Verbose description of the parameter.
  - from Text indicating the source of the parameter (either PKNCA or ubiquity).

See Also

Vignette on NCA (vignette("NCA", package = "ubiquity"))

---

system_nca_run *Automatic NCA*

Description

Performs NCA in an automated fashion

Usage

```
system_nca_run(
  cfg,
  dsname = "PKDS",
  dscale = 1,
  NCA_options = NULL,
  NCA_min = 4,
  analysis_name = "analysis",
  dsfilter = NULL,
  extrap_C0 = TRUE,
  extrap_N = 2,
  sparse = FALSE,
 dsmap = list(TIME = "TIME", NTIME = "NTIME", CONC = "CONC", DOSE = "DOSE", ID = "ID",
    ROUTE = "ROUTE", DOSENUM = NULL, BACKEXTRAP = NULL, SPARSEGROUP = NULL),
  dsinc = NULL
)
```

Arguments

cfg	ubiquity system object
dsname	name of dataset loaded with (system_load_data)
dscale	factor to multiply the dose to get it into the same units as concentration (default 1): if you are dosing in mg/kg and your concentrations is in ng/ml, then dscale = 1e6
NCA_options	specify a list of options for PKNCA to overwrite the defaults (default NULL will use defaults). For example if you want to set the maximum extrapolation of AUCinf to 10 half-life half-life of 0.8 you would use: list(max.aucinf.pext=10, min.hl.r.squared=.9)
NCA_min	minimum number of points required to perform NCA for a given subset (default 4)
analysis_name	string containing the name of the analysis (default 'analysis') to archive to files and reference results later
dsfilter	list of names corresponding to the column names in the dataset and values are a sequence indicating values to keep (default NULL. Multiple names are and-ed together. For example the following would keep all of the records where dose is 1, 2, or 5 and the dose_number is 1 `dsfilter = list(dose=c(1,2,5), dose_number = c(1))`
extrap_C0	Boolean variable to enable automatic determination of initial drug concentration if no value is specified; the rules used by WinNonlin will be used: • If the route is "iv infusion" or "extra-vascular" and the data is single dose data, then a concentration of zero will be used. If repeat dosing is used, the minimum value from the previous dosing interval will be used. • If the route is "iv bolus" then log-linear regression of the number of observations specified by extrap_N will be used. If the slope of these points is positive the first positive observation will be used as an estimate of C0
extrap_N	number of points to use for back extrapolation (default 2); this number can be overwritten for each subject using the BACKEXTRAP column in the dataset
sparse	Boolean variable used to indicate data used sparse sampling and the analysis should use the average at each time point (the SPARSEGROUP column must be specified in the dsmap below)
dsmap	list with names specifying the columns in the dataset (* required): • TIME* Time since the first dose; "TIME" (default) • NTIME* Nominal time since last dose; "NTIME" (default) • CONC* Concentration data; "CONC" (default) • DOSE* Dose given; ("DOSE" (default) • ID* Subject ID; ("ID" (default) • ROUTE* Route of administration; "ROUTE" (default), can be either "iv bolus", "iv infusion" or "extra-vascular". Variants such as "IV_bolus" and "extravascular" should work as well. • DOSENUM Numeric dose (starting at 1) used for grouping multiple dose data; optional, NULL (default) for single dose data)

- BACKEXTRAP Specifying the number of points to use to extrapolate the initial concentration for "iv bolus" dosing; optoinal f NULL (default) will use the value defined in extrap_N (note this value must be <= NCA_min)
- SPARSEGROUP Column containing a unique value grouping cohorts for pooling data. Needed when sparse is set to TRUE; optional, NULL (default)

dsinc (NOT CURRENTLY IMPLEMENTED) optional character vector of columns from the dataset to include in the output summary (default NULL)

Value

cfg ubiquity system object with the NCA results and if the analysis name is specified:

- output/analysis_name-nca_summary-pknca.csv NCA summary
- output/analysis_name-pknca_summary.csv Raw output from PKNCA with subject and dose number columns appended
- output/analysis_name-nca_data.RData objects containing the NCA summary and a list with the ggplot grobs

See Also

Vignette on NCA (vignette("NCA", package = "ubiquity"))

---

system_nca_summary *Summarize NCA Results in Tabular Format*

Description

Creates tabular summaries of NCA results

Usage

```
system_nca_summary(
  cfg,
  analysis_name = "analysis",
  treat_as_factor = c("ID", "Dose_Number", "Dose"),
  params_include = c("ID", "cmax", "tmax", "auclast"),
  params_header = NULL,
  rptname = "default",
  label_format = NULL,
  summary_stats = NULL,
  summary_labels = list(MEAN = "Mean", STD = "Std Dev", MEDIAN = "Median", N = "N obs",
    SE = "Std Err."),
  summary_location = NULL,
  ds_wrangle = NULL,
  digits = 3,
  table_theme = "theme_zebra"
)
```

Arguments

cfg	ubiquity system object
analysis_name	string containing the name of the analysis (default 'analysis') that was previously run
treat_as_factor	sequence of column names to be treated as factors (default c("ID", "Dose_Number", "Dose")). Use this to report values without added decimals.
params_include	vector with names of parameters to include (default c("ID", "cmax", "tmax", "auclast"))
params_header	list with names of parameters followed by a vector of headers. You can use the placeholder "<label>" to include the standard label (e.g. list(cmax=c("<label>", "(ng/ml)"))), with a default of NULL.
rptname	report name (either PowerPoint or Word) that this table will be used in ("default")
label_format	string containing the format in which headers and labels are being specified, either "text", or "md" (default NULL assumes "text" format)
summary_stats	list with strings as names containing placeholders for summary statistics and the values indicate the parameters to apply those statistics to. for example, if you want to calculate mean and standard deviation of AUClast you could use list("<MEAN> (<STD>)"=c("auclast"). This would create a row at the bottom of the table with this information for just the listed parameters. To split this up across two rows just do the following: list("<MEAN>"=c("auclast"), "<STD>"=c("auclast")). Any NA values will be ignored when calculating statistics. The allowed summary statistics are the mean (<MEAN>), median (<MEDIAN>), standard deviation (<STD>), standard error (<SE>), and the number of observations used to calculate statistics. (<N>). The default value of NULL prevents any summary statistics from being included.

summary_labels	list containing the mapping of summary statistics defined by summary_stats with their text labels in the output tables:
	`list(MEAN = "Mean",`
	`STD = "Std Dev",`
	`MEDIAN = "Median",`
	`N = "N obs",`
	`SE = "Std Err.")`

summary_location	column where to put the labels (e.g. Mean (Std)) for summary statistic. The default (NULL) will leave these labels off. If you set this to the "ID" column it will put them under the subject IDs.
ds_wrangle	ds_wrangle = list(Dose=c(30), Dose_Number = c(1))
digits	number of significant digits to report (3) or NULL to prevent rounding
table_theme	flextable theme see the flextable package for available themes, and set to NULL to prevent themes from being applied. (default="theme_zebra")

Value

list with the following elements

- isgood Boolean variable indicating success (TRUE) or failure (FALSE) if the call is successful the following will be defined (NULL
- nca_summary dataframe containing the summary table with headers and any summary statistics appended to the bottom
- nca_summary_ft same information in the nca_summary ouput as a flextable object
- components list with the elements of the summary table each as dataframes (header, data, and summary)

See Also

Vignette on NCA (vignette("NCA", package = "ubiquity"))

---

system_new *Create New* system.txt *File*

Description

Copy a blank template (system_file="template") file to the working directory or an example by specifying the following:

- "template" - Empty system file template
- "adapt" - Parent/metabolite model taken from the adapt manual used in estimation examples [ADAPT]
- "two_cmt_cl" - Two compartment model parameterized in terms of clearances
- "one_cmt_cl" - One compartment model parameterized in terms of clearances
- "two_cmt_micro" - Two compartment model parameterized in terms of rates (micro constants)
- "one_cmt_micro" - One compartment model parameterized in terms of rates (micro constants)
- "mab_pk" - General compartmental model of mAb PK from Davda 2014 [DG]
- "pbpk" - PBPK model of mAb disposition in mice from Shah 2012 [SB]
- "pbpk_template" - System parameters from Shah 2012 [SB] have been defined for all species along with the set notation to be used as a template for developing models with physiological parameters
- "pwc" - Example showing how to make if/then or piece-wise continuous variables
- "tmdd" - Model of antibody with target-mediated drug disposition
- "tumor" - Transit tumor growth model taken from Lobo 2002 [LB]

Usage

```
system_new(
  file_name = "system.txt",
  system_file = "template",
  overwrite = FALSE,
  output_directory = getwd()
)
```

Arguments

file_name	name of the new file to create
system_file	name of the system file to copy
overwrite	if TRUE the new system file will overwrite any existing files present
output_directory	getwd() directory where system file will be placed

Details

References

- [ADAPT] Adapt 5 Users Guide https://bmsr.usc.edu/files/2013/02/ADAPT5-User-Guide.pdf
- [DG] Davda et. al. mAbs (2014) 6(4):1094-1102 doi:10.4161/mabs.29095
- [LB] Lobo, E.D. & Balthasar, J.P. AAPS J (2002) 4, 212-222 doi:10.1208/ps040442
- [SB] Shah, D.K. & Betts, A.M. JPKPD (2012) 39 (1), 67-86 doi:10.1007/s1092801192322

Value

TRUE if the new file was created and FALSE otherwise

Examples

```
# To create an example system file named example_system.txt:
system_new(system_file      = "mab_pk",
           file_name        = "system_example.txt",
           overwrite        = TRUE,
           output_directory = tempdir())
```

system_new_list *Fetch List of Available System Templates*

Description

Returns a list of internal templates with descriptions of their contents and file locations

Usage

```
system_new_list()
```

Value

list with the template names as the keys

- file_path Full path to the system file
- description Description of what this system file provides

Examples

```
# To get a list of systems
systems = system_new_list()
```

system_new_tt_rule *Titration Rules*

Description

Defines a new titration rule and the times when that rule is evaluated

Usage

```
system_new_tt_rule(cfg, name, times, timescale)
```

Arguments

cfg	ubiquity system object
name	name for the titration rule
times	list of times when the rule will be evaluated
timescale	time scale associated with the titration times (as defined by <TS:?>)

Details

```
cfg = system_new_tt_rule(cfg,
                         name      = "rname",
                         times     = c(0, 2, 4),
                         timescale = "weeks")'
```

A titration rule identifies a set of times (`times`) and an associated time scale (`timescale`) in which titration events can potentially occur. Any times scale, as defined in the system file with <TS:?>, can be used in place of "weeks" above. The `name`, `"rname"` above, is used to link the titration rule to different conditions discussed below. The name should be a string beginning with a letter, and it can contain any combination of numbers, letters, and underscores. With the rule created we can then add conditions to that rule.'

Value

Ubiquity system object with the titration rule created

See Also

`system_set_tt_cond`, `run_simulation_titrate`

---

system_od_general *General Observation Details Function*

Description

Used to calculate observation details based on cohorts created with `system_define_cohort`

Usage

```
system_od_general(pest, cfg, estimation = TRUE, details = FALSE)
```

Arguments

`pest`	vector of parameters to be estimated
`cfg`	ubiquity system object
`estimation`	`TRUE` when called during an estimation and `FALSE` when called to test objective function or generate observation information for plotting
`details`	`TRUE` to display information about cohorts as they are simulated (useful for debugging when passed through `system_simulate_estimation_results`)

Value

If estimation is TRUE then the output is a matrix of observation details of the format:

```
od$pred  = [TIME, OBS, PRED, VAR, OUTPUT, COHORT]
```

The values are the observed (OBS) data, predicted values (PRED) and variance (VAR) at the given TIME. The columns OUTPUT and COHORT can be used for sorting. These should be unique numbers.

When estimation is FALSE we output od$pred is a data frame with the following headings:

```
od$pred  = [TIME, OBS, PRED, VAR, SMOOTH, OUTPUT, COHORT]
```

The TIME, OBS, PRED and VAR are the same as those listed above. The SMOOTH variable is FALSE for rows that correspond to records in the dataset and TRUE when the PRED represents the smooth predictions. The OUTPUT and COHORT columns here are text values used when defining the cohorts.

Also the od$all list item is created with all of the simulation information stored for each cohort:

```
od$all = [ts.time, ts.ts1, ... ts.tsn, pred, name, cohort]
```

- tstime - timescale of the system
- ts.ts1, ... ts.tsn - timescales defined in the system
- pred - smooth prediction
- name - state or output name corresponding to the prediction
- cohort - name of the cohort for these predictions

Lastly the field isgood will be set to FALSE if any problems are encountered, and TRUE if everything worked.

```
od$isgood = TRUE
```

See Also

system_define_cohort and system_simulate_estimation_results

---

system_plot_cohorts *Plot Estimation Results*

Description

Generates figures for each cohort/output for a given set of parameter estimates.

Usage

```
system_plot_cohorts(
  erp,
  plot_opts = c(),
  cfg,
  analysis_name = "analysis",
  archive_results = TRUE,
  prefix = NULL
)
```

Arguments

erp	output from system_simulate_estimation_results
plot_opts	list controling how predictions and data are overlaid
cfg	ubiquity system object
analysis_name	string containing the name of the analysis
archive_results	boolean variable to control whether results will be archived
prefix	depreciated input mapped to analysis_name

Details

The general format for a plot option for a given output (OUTPUT) is:

plot_opts$outputs$OUTPUTt$option = value

The following options are:

- yscale and xscale = "linear" or "log"
- ylabel and xlabel = "text"
- xlim and ylim = c(min, max)

It is also possible to control the height and width of the time course tc and observed vs predicted op file by specifying the following in the default units of ggsave.

- plot_optstcwidth = 10
- plot_optstcheight = 5.5
- plot_optsopwidth = 10
- plot_optsopheight = 8.0

To control the figures that are generated you can set the purpose to either "print", "present" (default) or "shiny".

plot_opts$purpose = "present"

Value

List of plot outputs containing two elements `timecourse` and `obs_pred`, for the time course of and observed vs predicted, respectively. Both of these fields contain three elements for a given output. For example, say there is an output named `PK` the both the `timecourse` and `obs_pred` elements will have a field named `PK` containing a ggplot object and two fields `PK_png` and `PK_pdf` containing the paths to the files containing that figure in the respective formats.

See Also

The estimation vignette (`vignette("Estimation", package = "ubiquity")`)

---

system_rpt_add_doc_content

Adds Content to a Word Report

Description

Appends content to an open ubiquity Word report.

Usage

```
system_rpt_add_doc_content(
  cfg,
  type = NULL,
  content = NULL,
  rptname = "default"
)
```

Arguments

`cfg`	ubiquity system object
`type`	Type of content to add. See the onbrand function `report_add_doc_content` for the allowed content types.
`content`	List with content to add to the report. See the onbrand function `report_add_doc_content` format of this list.
`rptname`	Report name

Value

ubiquity system object with the content added to the specified report

See Also

`report_add_doc_content` and Reporting vignette (`vignette("Reporting", package = "ubiquity")`)

system_rpt_add_slide *Add Slide to a Powerpoint Report*

Description

Adds a slide to a ubiquity report.

Usage

```
system_rpt_add_slide(
  cfg,
  template = NULL,
  elements = NULL,
  rptname = "default"
)
```

Arguments

cfg	ubiquity system object
template	Name of slide template to use
elements	List with content to populate placeholders in the slide. See the onbrand functions report_add_slide and add_pptx_ph_content for details on the expected format of this list.
rptname	Report name

Value

ubiquity system object with the slide added to the specified report

See Also

report_add_slide, add_pptx_ph_content, and Reporting vignette (vignette("Reporting", package = "ubiquity"))

system_rpt_estimation *Generate a Report from Parameter Estimation*

Description

This will take the output generated during a parameter estimation and append those results to a specified report.

Usage

```
system_rpt_estimation(cfg, rptname = "default", analysis_name = NULL)
```

Arguments

cfg	ubiquity system object
rptname	report name ("default")
analysis_name	string containing the name of the estimation analysis and used as a prefix to store the results

Value

ubiquity system object with estimation report appended

See Also

system_rpt_read_template, the reporting vignette (vignette("Reporting", package = "ubiquity")) and the estimation vignette (vignette("Estimation", package = "ubiquity"))

---

system_rpt_nca *Report NCA*

Description

Appends the results of NCA to a report

Usage

```
system_rpt_nca(
  cfg,
  rptname = "default",
  analysis_name = "analysis",
  rows_max = 10,
  table_headers = TRUE
)
```

Arguments

cfg	ubiquity system object
rptname	report name (either PowerPoint or Word)
analysis_name	string containing the name of the NCA analysis (default 'analysis')
rows_max	maximum number of rows per slide when generating tabular data
table_headers	Boolean variable to add descriptive headers to output tables (default TRUE)

Value

cfg ubiquity system object with the NCA results appended to the specified report and if the analysis name is specified:

See Also

Vignette on NCA (`vignette("NCA", package = "ubiquity")`)

`system_rpt_read_template` *Initialize a New Report*

Description

Creates a new officer report based either on the ubiquity template or one specified by the user. Once created, content can then be added.

Usage

```
system_rpt_read_template(
  cfg,
  template = "PowerPoint",
  mapping = NULL,
  rptname = "default"
)
```

Arguments

`cfg`	ubiquity system object
`template`	Type of internal template to use ("PowerPoint" or "Word") or path to template file.
`mapping`	Path to an onbrand yaml mapping file: If an internal ubiquity template has been supplied, this argument will be ignored and the yaml file from ubiquity will be used.
`rptname`	report name

Details

The ‘template‘ and ‘mapping‘ inputs can specify either the internal ubiquity templates or user-defined templates If you specify ‘template‘ values of ’PowerPoint‘ or ‘Word‘ then the internal ubiquity templates for PowerPoint or Word will be used and the mapping information will be ignored.

If templates other than the values above are specified you will need also supply a yaml mapping file for an ‘onbrand‘ reporting template. The vignette below highlights how to go about creating these files.

Value

ubiquity system object with and empty report initialized

See Also

Reporting vignette (`vignette("Reporting", package = "ubiquity")`)

Custom Office Template vignette (`vignette("Custom_Office_Templates", package="onbrand")`)

`system_rpt_save_report`

Save Report to a File

Description

Saves a ubiquity report to the specified file.

Usage

```
system_rpt_save_report(cfg, output_file = NULL, rptname = "default")
```

Arguments

`cfg`	ubiquity system object
`output_file`	File to save the report to (must be either .pptx or .docx depending on the type of report)
`rptname`	ubiquity report name

Value

list with the follwoing elements

- isgood Boolean variable indicating success or failure
- msgs Verbose description of the save results

See Also

Reporting vignette (`vignette("Reporting", package = "ubiquity")`)

system_rpt_template_details *Generate Details about Report Template*

Description

Wrapper for the onbrand::template_details function, see the help for that function for more information

Usage

```
system_rpt_template_details(cfg, rptname = "default")
```

Arguments

cfg	ubiquity system object
rptname	Report name

Value

list with template information, see template_details for information on the structure of this list.

See Also

template_details and Reporting vignette (vignette("Reporting", package = "ubiquity"))

system_select_set *Selecting Parameter Sets*

Description

The system file can contain multiple parameterizations using the <PSET:?:?>? notation. This function provides the means for switching between these parameterizations, and (optionally) specifying a subset of parameters estimated when performing parameter estimation.

Usage

```
system_select_set(cfg, set_name = "default", parameter_names = NULL)
```

Arguments

cfg	ubiquity system object
set_name	string containing the name of the parameter set
parameter_names	list of parameter names to be estimated

Value

Ubiquity system object with the specified parameter set active

Examples

```
# Creating a system file from the mab_pk example
fr = system_new(file_name        = "system.txt",
                system_file      = "mab_pk",
                overwrite        = TRUE,
                output_directory = tempdir())

# Building the system
cfg = build_system(system_file  = file.path(tempdir(), "system.txt"),
      output_directory          = file.path(tempdir(), "output"),
      temporary_directory       = tempdir())

# Selecting the default parameter set
cfg = system_select_set(cfg, "default")
```

---

system_set_bolus *Set Bolus Inputs*

Description

Defines infusion rates specified in the system file using <B:times> and <B:events>

Usage

```
system_set_bolus(cfg, state, times, values)
```

Arguments

cfg	ubiquity system object
state	name of the state to apply the bolus
times	list of injection times
values	corresponding list injection values

Value

Ubiquity system object with the bolus information set

See Also

system_zero_inputs

Examples

```
# Creating a system file from the mab_pk example
fr = system_new(file_name        = "system.txt",
                system_file      = "mab_pk",
                overwrite        = TRUE,
                output_directory = tempdir())

# Building the system
cfg = build_system(system_file  = file.path(tempdir(), "system.txt"),
      output_directory          = file.path(tempdir(), "output"),
      temporary_directory       = tempdir())

# Clearing all inputs
cfg = system_zero_inputs(cfg)

# SC dose of 200 mg
cfg = system_set_bolus(cfg, state   ="At",
                            times   = c(  0.0),  #  day
                            values  = c(200.0))  #  mg
```

`system_set_covariate` *Set Covariate Values*

Description

Covariates specified in the system file using <CV:?> and <CVSET:?:?> will have their default values for a given parameter set. This function is a means to overwrite those values.

Usage

```
system_set_covariate(cfg, covariate, times, values)
```

Arguments

`cfg`	ubiquity system object
`covariate`	name of the covariate
`times`	list of times (system time units)
`values`	corresponding list of values

Value

Ubiquity system object with the covariate set

Examples

```
# Creating a system file from the mab_pk example
fr = system_new(file_name        = "system.txt",
                system_file      = "mab_pk",
                overwrite        = TRUE,
                output_directory = tempdir())

# Building the system
cfg = build_system(system_file  = file.path(tempdir(), "system.txt"),
      output_directory          = file.path(tempdir(), "output"),
      temporary_directory       = tempdir())

# Setting the covariate WT to 50
cfg = system_set_covariate(cfg,
                           covariate = "WT",
                           times     = c(0),
                           values    = c(50))
```

`system_set_guess` *Alter Initial Guess and Parameter Bounds*

Description

Default values for parameters are taken from the `system.txt` file either when the parameter was defined (`<P>`) or when it was reassigned for a parameter set (`<PSET:?:?>?`). These can be altered at the scripting level using this function.

Usage

```
system_set_guess(cfg, pname, value, lb = NULL, ub = NULL)
```

Arguments

`cfg`	ubiquity system object
`pname`	name of parameter to set
`value`	value to assign
`lb`	optionally change the lower bound (`NULL`)
`ub`	optionally change the upper bound (`NULL`)

Details

When performing a parameter estimation, the initial guess will be the value specified in the `system.txt` file for the currently selected parameter set. The following command can be used after the parameter set has been selected to specify the value (`VALUE`) of the parameter `PNAME` and optionally the lower (`lb`) and upper (`ub`) bounds:

```
cfg = system_set_guess(cfg, pname="PNAME", value=VALUE, lb=NULL, ub=NULL)
```

To set the initial guess for the parameter Vc to a value of 3, the following would be used:

```
cfg = system_set_guess(cfg, "Vc", value=3)
```

To specify the guess and overwrite the upper bound on Vc and set it to 5

```
cfg = system_set_guess(cfg, "Vc", value=3, ub=5)
```

Value

cfg ubiquity system object with guess and bounds assigned

`system_set_iiv` *Set Variability Terms*

Description

Set elements of the current variance covariance matrix specified in the system file with <IIV:?:?> ?, <IIVCOR:?:?>?, <IIVSET:?:?> ?, <IIVCORSET:?:?>?

Usage

```
system_set_iiv(cfg, IIV1, IIV2, value)
```

Arguments

`cfg`	ubiquity system object
`IIV1`	row name of the variance/covariance matrix
`IIV2`	column name of the variance/covariance matrix element
`value`	value to assign to the variance/covariance matrix element

Value

Ubiquity system object with IIV information set

See Also

`system_fetch_iiv`

Examples

```
# Creating a system file from the mab_pk example
fr = system_new(file_name        = "system.txt",
                system_file      = "mab_pk",
                overwrite        = TRUE,
                output_directory = tempdir())

# Building the system
cfg = build_system(system_file  = file.path(tempdir(), "system.txt"),
      output_directory          = file.path(tempdir(), "output"),
      temporary_directory       = tempdir())

# Clearing all inputs
cfg = system_zero_inputs(cfg)

# Setting the covariance element for CL and Vc to 0.03
cfg = system_set_iiv(cfg,
                     IIV1 = "ETACL",
                     IIV2 = "ETAVc",
                     value=0.03)
```

`system_set_option` *Setting Analysis Options*

Description

Different options associated performing analyses (e.g running simulations, performing parameter estimation, logging, etc.) can be set with this function

Usage

```
system_set_option(cfg, group, option, value)
```

Arguments

`cfg`	ubiquity system object
`group`	options are grouped together by the underlying activity being performed: "estimation", "general", "logging", "simulation", "solver", "stochastic", or "titration"
`option`	for each group there are a set of options
`value`	corresponding value for the option

Details

`group="estimation"`

The default estimation in R is performed using either the `optim` or `optimx` libraries. This is selected by setting the `optimizer` option:

```
cfg = system_set_option(cfg,
                        group  = "estimation",
                        option = "optimizer",
                        value  = "optim")
```

The optimization routine then specified using the `method`. By default this `option` is set to `Nelder-Mead`.

```
cfg = system_set_option(cfg,
                        group  = "estimation",
                        option = "method",
                        value  = "Nelder-Mead")
```

And different attributes are then selected using the control.

```
cfg = system_set_option(cfg,
                        group  = "estimation",
                        option = "control",
                        value  = list(trace  = TRUE,
                                      maxit  = 500,
                                      REPORT = 10))
```

For the different methods and control options, see the documentation for the `optim` and `optimx` libraries.

To perform a global optimization you can install either the particle swarm (`pso`) genetic algorithm (`GA`) libraries. To use the particle swarm set the `optimizer` and `method`:

```
cfg = system_set_option(cfg,
                        group  = "estimation",
                        option = "optimizer",
                        value  = "pso")

cfg = system_set_option(cfg,
                        group  = "estimation",
                        option = "method",
                        value  = "psoptim")
```

The control option is a list described `pso` documentation.

To use the genetic algorithm set the optimizer and method:

```
cfg = system_set_option(cfg,
                        group  = "estimation",
                        option = "optimizer",
                        value  = "ga")

cfg = system_set_option(cfg,
                        group  = "estimation",
                        option = "method",
                        value  = "ga")
```

The control option is a list and the list elements are the named options in the GA documentation. Use the following as an example:

```
cfg = system_set_option(cfg,
                        group  = "estimation",
                        option = "control",
                        value  = list(maxiter  = 10000,
                                      optimArgs = list(
                                        method  = "Nelder-Mead",
                                        maxiter = 1000)))
```

To alter initial guesses see: system_set_guess

When performing parameter estimation, the internal function system_od_general is used. This is the function that simulates your system at the conditions defined for the different cohorts. This is pretty flexible but if you want to go beyond this you can set the observation_function option:

```
cfg = system_set_option(cfg,
                        group  = "estimation",
                        option = "observation_function",
                        value  = "my_od")
```

That will instruct the optimziation routines to use the user defined function my_od. You will need to construct that function to have the same input/output format as system_od_general.

group=general

- "output_directory" = String where analysis outputs will be placed. Generally you wont want to change this, but it can be useful in Shiny apps where you need to have each shiny user generate output in that users directory : file.path(".", "output")

group=logging

By default ubiquity prints different information to the console and logs this information to a log file. The following options can be used to control this behavior:

- "enabled" = Boolean variable to control logging: TRUE
- "file" = String containing the name of the log file: file.path("transient", "ubiquity_log.txt")
- "timestamp" = Boolean switch to control appending a time stamp to log entries: TRUE
- "ts_str" = String format of timestamp: "
- "debug" = Boolean switch to control debugging (see below): FALSE
- "verbose" = Boolean switch to control printing to the console FALSE

To enable debugging of different functions (like when performing esitmation), set the debug option to TRUE. Important function calls will be trapped and information will be logged and reported to the console.

```
cfg = system_set_option(cfg,
                        group  = "estimation",
                        option = "debug",
                        value  = FALSE)
```

group="simulation"

- "dynamic" - Set to TRUE (default) and simulations will behave normally. Set to FALSE and ODES will evaulate to zero. This is useful for steady-state anslysis.
- "include_important_output_times" - Automatically add bolus, infusion rate switching times, etc: "yes"(default), "no".
- "integrate_with" - Specify if the ODE solver should use the Rscript ("r-file") or compiled C ("c-file"), if the build process can compile and load the C version it will be the default otherwise it will switch over to the R script.
- "output_times" - Vector of times to evaulate the simulation (default seq(0,100,1)).
- "solver" - Selects the ODE solver: "lsoda" (default), "lsode", "vode", etc.; see the documentation for deSolve for an exhaustive list.
- "sample_bolus_delta" - Spacing used when sampling around bolus events (default 1e-6).
- "sample_forcing_delta" - Spacing used when sampling around forcing functions (infusion rates, covariates, etc) (default 1e-3).

group=solver

Depending on the solver, different options can be set. The documentation for deSolve lists the different solvers. For a full list of options, see the documentation for the specific solver (e.g. ?lsoda). Some common options to consider are:

- "atol" - Relative error tolerance
- "rtol" - Absolute error tolerance
- "hmin" - Minimum integration step size
- "hmax" - Maximum integration step size

To select the vode solver and set the maximum step size to 0.01, the following would be used:

```
cfg=system_set_option(cfg,
                      group  = "simulation",
                      option = "solver",
                      value  = "vode")

cfg=system_set_option(cfg,
                      group  = "solver",
                      option = "hmax",
                      value  = 0.01)
```

group="stochastic"

When running stochastic simulations (inter-individual variability applied to system parameters) it can be useful to specify the following:

- "ci" - Confidence interval (default 95)
- "nsub" - Number of subjects (default 100)
- "seed" - Seed for the random numebr generator (default 8675309)
- "ponly" - Only generate the subject parameters but do not run the simulations (default FALSE)

- "ssp" - A list of the calculated static secondary parameters to include (default all parameters defined by <As>)
- "outputs" - A list of the predicted outputs to include (default all outputs defined by <O>)
- "states" - A list of the predicted states to include(default all states)
- "sub_file" - Name of data set loaded with (system_load_data) containing subject level parameters and coviariates
- "sub_file_sample" - Controls how subjects are sampled from the dataset

If you wanted to generate 1000 subjects but only wanted the parameters, you would use the following:

```
cfg = system_set_option(cfg,
                        group  = "stochastic",
                        option = "nsub ",
                        value  = 1000)

cfg = system_set_option(cfg,
                        group  = "stochastic",
                        option = "ponly",
                        value  = TRUE )
```

If you wanted to exclude both states and secondary parameters, while only including the output Cp_nM, you would do the following:

```
cfg = system_set_option (cfg,
                         group  = "stochastic",
                         option = "ssp",
                         value  = list())

cfg = system_set_option (cfg,
                         group  = "stochastic",
                         option = "states",
                         value  = list())

cfg = system_set_option (cfg,
                         group  = "stochastic",
                         option = "outputs",
                         value  = c("Cp_nM"))
```

To pull subject information from a data file instead of generating the subject parameters from IIV information the sub_file option can be used. The value here SUBFILE_NAME is the name given to a dataset loaded with (system_load_data):

```
cfg=system_set_option(cfg,
                      group  = "stochastic",
                      option = "sub_file",
                      value  = "SUBFILE_NAME")
```

Sampling from the dataset can be controlled using the sub_file_sample option:

```
cfg=system_set_option(cfg,
                      group  = "stochastic",
                      option = "sub_file_sample",
                      value  = "with replacement")
```

Sampling can be done sequentially ("sequential"), with replacement ("with replacement"), or without replacement ("without replacement")

group="titration"

"titrate" - By default titration is disable (set to FALSE). If you are going to use titration, enable it here by setting this option to TRUE. This will force #' simulate_subjects to use run_simulation_titrate internally when running simulations.

Value

Ubiquity system object with the option set

---

system_set_parameter *Set Value for Parameter*

Description

Assigns a value for a named parameter in a parameter list.

Usage

```
system_set_parameter(cfg, parameters, pname, value)
```

Arguments

cfg	ubiquity system object
parameters	vector of parameters
pname	parameter name
value	value

Details

To set the parameter Vc to a value of 3, the following would be used:

```
parameters = system_fetch_parameters(cfg)
parameters = system_set_parameter(cfg, parameters, pname = 'Vc', value = 3)
```

Value

parameters vector with pname set to value

`system_set_rate` *Set Infusion Rate Inputs*

Description

Defines infusion rates specified in the system file using <R:?>

Usage

```
system_set_rate(cfg, rate, times, levels)
```

Arguments

`cfg`	ubiquity system object
`rate`	name of infusion rate
`times`	list of time values
`levels`	corresponding list of infusion values

Value

Ubiquity system object with the infusion rate set

See Also

`system_zero_inputs`

Examples

```
# Creating a system file from the mab_pk example
fr = system_new(file_name        = "system.txt",
                system_file      = "mab_pk",
                overwrite        = TRUE,
                output_directory = tempdir())

# Building the system
cfg = build_system(system_file  = file.path(tempdir(), "system.txt"),
      output_directory          = file.path(tempdir(), "output"),
      temporary_directory       = tempdir())

# Clearing all inputs
cfg = system_zero_inputs(cfg)

# 5 minute infusion at 10 mg/min
cfg = system_set_rate(cfg,
           rate   = "Dinf",
           times  = c(0,  5),
           levels = c(10, 0))
```

`system_set_rpt_officer_object`
Sets the officer Object for the Specified ubiquity Report

Description

This will replace the officer object in the ubiqiuty system object for the specified report name with the value supplied.

Usage

```
system_set_rpt_officer_object(cfg, rpt = NULL, rptname = "default")
```

Arguments

`cfg`	ubiquity system object
`rpt`	officer report object
`rptname`	ubiquity report name

Value

ubiquity system object with the replaced officer object

See Also

`system_fetch_rpt_officer_object`

`system_set_rpt_onbrand_object`
Sets the onbrand Object for the Specified ubiquity Report

Description

This will reset the onbrand object in the ubiqiuty system object for the specified report name.

Usage

```
system_set_rpt_onbrand_object(cfg, obnd = NULL, rptname = "default")
```

Arguments

`cfg`	ubiquity system object
`obnd`	onbrand report object
`rptname`	ubiquity report name

Value

ubiquity system object with onbrand report set

See Also

system_fetch_rpt_onbrand_object

---

system_set_tt_cond *Define Titration Triggers and Actions*

Description

Once a rule has been defined using system_new_tt_rule, it can then be used by specifying checks at each of the titration time points that, when true, will perform some actions.

Usage

```
system_set_tt_cond(cfg, name, cond, action, value = "-1")
```

Arguments

cfg	ubiquity system object
name	string containing the name for the titration rule to which this condition applies
cond	string that evaluates a boolean value that is TRUE when the action should be triggered
action	stringing that evaluates to what should be done when the condition is met (e.g. changing the dose, state change, etc)
value	code to be stored in the titration history to track when this condition has been triggered

Details

The general syntax for setting a new condition is:

```
cfg = system_new_tt_cond(cfg,
                         name   = "rname",
                         cond   = "BOOLEAN EXPRESSION",
                         action = "EXPRESSION",
                         value  = "VALUE")
```

The name input will associate this condition with a previously defined rule. For each time defined when the rule was created, the condition (cond) will be evaluated. If that condition evaluates as TRUE then the action will be evaluated. Lastly, when a condition action is evaluated, the value is stored in the titration history.

Multiple conditions can be associated with a rule. The internal titration history will track each one where a condition has been evaluated as true, but the simulation output will only show the **last** condition to be evaluated as true.

The cond field is a string that, when evaluated, will produce a boolean value (TRUE or FALSE). If you simply want to force an action at each of the times for a given rule you can use: cond = "TRUE". Alternatively you can provide mathematical expressions or even complicated user defined functions.

The action field is evaluated when cond is true. To modify how a simulation is going to be performed, you will want to modify the SIMINT_cfgtt variable using the different system commands. Certain common tasks have prototype functions created to make it easier for the user:

- SI_TT_BOLUS - Set bolus dosing
- SI_TT_RATE - Set infusion inputs
- SI_TT_STATE - Reset system states

Note: Protype functions are strings but sometimes it is necessary to specify strings within this string. For the main string use double quotes (") and for the internal strings use single quotes (')

SI_TT_BOLUS

The simplest way to apply a bolus when the condition is true is to use the following:

```
action = "SI_TT_BOLUS[state='At',
                      values=c(10, 10, 10),
                      times=c(0, 1, 2)]"
```

The values and times are vectors of numbers of equal length. The dosing and time units are those specified in the system.txt file for the <B:?> delimiter. The times are relative to the titration time. So 0 above means at the titration time.

It's possible to specify an interval and a number of times to repeat the last dose using the following:

```
action = "SI_TT_BOLUS[state    = 'At',
                      values   = c(5, 5, 10),
                      times    = c(0, 2, 4),
                      repdose  = 'last',
                      number   = 7,
                      interval = 4]"
```

This will give a dose of 5 at the titration point and 2 time units later. The dose of 10 at time 4 will be repeated 7 times every 4 time units. So a total of 8 (7 + 1) doses at 10 will be administered. Remember the time units were those defined in <B:?>. The input repdose can be either 'last' or 'none'.

Note: The main string is in double quotes " " but the strings in the protype argument (e.g. 'last') are in single quotes ' '.

SI_TT_RATE

If you created an infusion named Dinf using <R:?> and the infusion units are min (times) and mg/min (rates). To have a 60 minute infusion of 20 mg/min then we would do the following:

```
action = "SI_TT_RATE[rate='Dinf', times=c(0, 60), levels=c(20.0, 0)]"
```

If we wanted to do this every day for 9 more days (a total of 10 days) we can repeat the sequence:

```
action = "SI_TT_RATE[rate     = 'Dinf',
                     times    = c(0, 60),
                     levels   = c(20, 0),
                     repdose  = 'sequence',
                     number   = 9,
                     interval = 24*60]"
```

The input repdose can be either 'sequence' or 'none'.

Note: The time units and dosing rate are those specified using <R:?>.

SI_TT_STATE

To provide fine control over states at titration points the state reset prototype is provided. For example, if you are modeling an assay where there is a wash step and you want to drop a concentration to zero. If you have a state named Cc defined in your system.txt and you want to set it to 0.0 in a condition the following action would work.

```
action = "SI_TT_STATE[Cc][0.0]"
```

The value here is a number but you can use any mathematical combination of variables available in the titration environment. Also you can create your own user function and place the function call within the brackets above.

Titration Environment

The cond, action, and value statements can use any variables available in the titration environment. If you want to perform complicated actions, you can simply create a user defined functions and pass it the variables from the titration environment that you need. These include named variables from the model as well as internal variables used to control the titration.

States and Parameters

System parameters (<P>), static secondary parameters (<As>) and the initial value of covariates are available. Also the state values (at the current titration time) can be used. These are all available as the names specified in the system.txt file. Since system resets (SI_TT_STATE) are processed first, any changes made to states are the values that are active for other actions.

Internal Simulation Variables

Internal variables are used to control titration activities. These variables can also be used in the conditions and actions.

- SIMINT_p - list of system parameters
- SIMINT_cfg - system configuration sent into the titration routine
- SIMINT_cfgtt- system configuration at the current titration event time
- SIMINT_ttimes - vector of titration times (in simulation units)
- SIMINT_ttime - current titration time (in simulation units)
- SIMINT_tt_ts - list of time scales for the current titration

- SIMINT_history - data frame tracking the history of conditions that evaluated true with the following structure:
- – tname - name of titration rule
  – value - value indicating condition that was satisfied
  – simtime - simulation time when that rule/value were triggered
  – timescale - time at the rule timescale when that rule/value were triggered

Individual Simulations

To run an individual titration simulation use the following:

```
som = run_simulation_titrate(parameters, cfg)
```

This provides the same output as run_simulation_ubiquity with two extra fields. The first, som$titration, contains three columns for each titration rule. The columns will have a length equal and corresponding to the simulation times. If the rule name is rname, then the column headers will have the following names and meanings:

- tt.rname.value - Value of the rule for the active condition or -1 if not triggered
- tt.rname.simtime - Simulation time where the last condition became active
- tt.rname.timescale - Simulation time in the time scale the rule was specified in

The second field is som$titration_history which contains a summary list of all of the titration events that were triggered.

- tname - Titration rule name
- value - Value of the rule for the active condition or -1 if not triggered
- simtime - Simulation time where the last condition became active
- timescale - Simulation time in the time scale the rule was specified in

To convert this structured list into a data frame the som_to_df command can be used:

```
sdf = som_to_df(cfg, som)
```

To run stochastic titration simulations, the same function is used:

```
som = simulate_subjects(parameters, cfg)
```

This will add a data a list element called som$titration with three fields for each titration rule:

- tt.rname.value - Value of the rule for the active condition or -1 if not triggered
- tt.rname.simtime - Simulation time where the last condition became active
- tt.rname.timescale - Simulation time in the time scale the rule was specified in

Each of these fields is a matrix with an entry for each simulation time (column) and each subject (row). This data structure can also be converted to a data frame using som_to_df.

Value

Ubiquity system object with the titration condition defined

See Also

system_new_tt_rule, run_simulation_titrate, som_to_df, simulate_subjects

---

system_set_tt_rate *Actual Function Called by* SI_TT_RATE

Description

The prototype function SI_TT_RATE provides an abstract interface to this function. Based on the input from SI_TT_RATE infusion rate inputs will be updated for the current titration time.

Usage

```
system_set_tt_rate(
  cfg,
  rate,
  times,
  levels,
  tt_ts,
  tsinfo,
  repdose = "none",
  interval = 1,
  number = 0
)
```

Arguments

cfg	ubiquity system object
rate	name of the infusion rate to update(Defined in <R:?>)
times	vector of switching times relative to the current titration time (in time units defined by <R:?>)
levels	vector of infusion rates (in dosing units defined by <R:?>)
tt_ts	list of timescale values for the current titration time
tsinfo	list with timescale information for inputs (bolus, rates, etc)
repdose	"none" or "sequence"
interval	interval to repeat in the units defined in <R:?>
number	number of times to repeat

Value

ubiquity system object with the infusion rates updated.

system_simulate_estimation_results *Simulate Results at Estimates*

Description

Simulates the system at the parameter estimates `pest` for creating diagnostic plots

Usage

```
system_simulate_estimation_results(pest, cfg, details = FALSE)
```

Arguments

`pest`	vector of parameters
`cfg`	ubiquity system object
`details`	set `TRUE` to display information about cohorts as they are simulated (useful for debugging)

Value

observations in a list, see `system_od_general` when `estimation=FALSE`

See Also

`system_define_cohort`, `system_plot_cohorts` and the vignette on parameter estimation (`vignette("Estimation", package = "ubiquity")`)

system_view *View Information About the System*

Description

Displays information (dosing, simulation options, covariates, etc) about the system.

Usage

```
system_view(cfg, field = "all", verbose = FALSE)
```

Arguments

`cfg`	ubiquity system object
`field`	string indicating the aspect of the system to display
`verbose`	Boolean variable that when set to true will echo the information to the screen

Value

sequence of strings with system in formation (one line per element)

The `field`

- `"all"` will show all information about the system
- `"parameters"` summary of parameter information
- `"bolus"` currently set bolus dosing
- `"rate"` infusion rate dosing
- `"covariate"` covariates
- `"iiv"` variance/covariance information
- `"datasets"` loaded datasets
- `"simulation"` simulation options
- `"estimation"` estimation options
- `"nca"` non-compartmental analyses that have been performed

Examples

```
# To log and display the current system information:

# Creating a system file from the mab_pk example
fr = system_new(file_name        = "system.txt",
                system_file      = "mab_pk",
                overwrite        = TRUE,
                output_directory = tempdir())

# Building the system
cfg = build_system(system_file  = file.path(tempdir(), "system.txt"),
      output_directory          = file.path(tempdir(), "output"),
      temporary_directory       = tempdir())

  msgs = system_view(cfg, verbose=TRUE)
```

---

system_zero_inputs *Zero All Model Inputs*

Description

Multiple default inputs can be specified in the system file. At the scripting level this function can be used to set all inputs to zero. Then only the subsequently specified inputs will be applied.

Usage

```
system_zero_inputs(cfg, bolus = TRUE, rates = TRUE)
```

Arguments

cfg	ubiquity system object
bolus	Boolean value indicating weather bolus inputs should be set to zero
rates	Boolean value indicating weather infusion rate inputs should be set to zero

Value

Ubiquity system object with the specified inputs set to zero

See Also

system_set_rate, system_set_bolus

Examples

```
# Creating a system file from the mab_pk example
fr = system_new(file_name        = "system.txt",
                system_file      = "mab_pk",
                overwrite        = TRUE,
                output_directory = tempdir())

# Building the system
cfg = build_system(system_file  = file.path(tempdir(), "system.txt"),
      output_directory          = file.path(tempdir(), "output"),
      temporary_directory       = tempdir())

# Clear only infusion rates
cfg = system_zero_inputs(cfg, bolus=TRUE, rates=FALSE)

# Clear all inputs:
cfg = system_zero_inputs(cfg)
```

---

tic *Implementation of Matlab* tic() *command*

Description

Used in conjunction with toc() to find the elapsed time when code is executed.

Usage

```
tic(type = c("elapsed", "user.self", "sys.self"))
```

Arguments

type	can be either "elapsed" "user.self" or "sys.self"

Value

time tic was called

See Also

toc

Examples

```
tic()
Sys.sleep(3)
toc()
```

---

toc *Implementation of Matlab* `toc()` *command*

Description

Used in conjunction with `tic()` to find the elapsed time when code is executed.

Usage

```
toc()
```

Value

time in seconds since tic() was called

See Also

tic

Examples

```
tic()
Sys.sleep(3)
toc()
```

var2string *Converts Numeric Variables into Padded Strings*

Description

Mechanism for converting numeric variables into strings for reporting.

Usage

```
var2string(vars, maxlength = 0, nsig_e = 3, nsig_f = 4)
```

Arguments

vars	numeric variable or a vector of numeric variables
maxlength	if this value is greater than zero spaces will be added to the beginning of the string until the total length is equal to maxlength
nsig_e	number of significant figures for scientific notation
nsig_f	number of significant figures for numbers (2.123)

Value

Number as a string padded

Examples

```
var2string(pi, nsig_f=20)
var2string(.0001121, nsig_e=2, maxlength=10)
```

vp *Print and Log Messages*

Description

Used to print messages to the screen and the log file.

Usage

```
vp(cfg, str, fmt = "alert")
```

Arguments

cfg	ubiquity system object
str	sequence of strings to print
fmt	string format should be one of the following: "h1", "h2", "h3", "verbatim", "alert" (default), "warning", "danger".

Value

Boolean variable indicating success (TRUE) or failure (FALSE)

Examples

```
# Creating a system file from the mab_pk example
fr = system_new(file_name        = "system.txt",
                system_file      = "mab_pk",
                overwrite        = TRUE,
                output_directory = tempdir())

# Building the system
cfg = build_system(system_file  = file.path(tempdir(), "system.txt"),
      output_directory          = file.path(tempdir(), "output"),
      temporary_directory       = tempdir())

# Initialzing the log file
vp(cfg, "Message that will be logged")
```

---

workshop_fetch *Fetch Ubiquity Workshop Sections*

Description

With the ubiquity package this function can be used to fetch example files for different sections of the workshop.

Usage

```
workshop_fetch(
  section = "Simulation",
  overwrite = FALSE,
  copy_files = TRUE,
  output_directory = getwd()
)
```

Arguments

section	Name of the section of workshop to retrieve ("Simulation")
overwrite	if TRUE the new workshop files will overwrite any existing files present (FALSE)
copy_files	if TRUE the files will be written to the output_directory, if FALSE only the names and locations of the files will be returned (TRUE)
output_directory	directory where workshop files will be placed (getwd())

Details

Valid sections are "Simulation", "Estimation", "In Vitro", "Titration" "Reporting", and "NCA"

Value

list

Examples

```
workshop_fetch("Estimation", output_directory=tempdir(), overwrite=TRUE)
```

Index