Package 'mrgsolve'

November 1, 2017

Type Package

Version 0.8.10

Title Simulate from ODE-Based Population PK/PD and Systems Pharmacology Models

Maintainer Kyle T Baron <kyleb@metrumrg.com>

URL https://github.com/metrumresearchgroup/mrgsolve

BugReports https://github.com/metrumresearchgroup/mrgsolve/issues

Copyright Metrum Research Group, LLC 2017

Description Facilitates simulation from hierarchical, ordinary differential equation (ODE) based models typically employed in drug development. A model specification file is created consisting of R and C++ code that is parsed, compiled, and dynamically loaded into the R session. Input data are passed in and simulated data are returned as R objects. A dosing event engine allows interventions (bolus and infusion) to be managed separately from the model code. Differential equations are solved with the 'DLSODA' routine in 'ODEPACK' (<https://computation.llnl.gov/casc/odepack/>).

License GPL (>= 2)

Depends R (>= 3.1.2), methods

Imports Rcpp (>= 0.12.12), dplyr (>= 0.7.4), magrittr (>= 1.5), lazyeval (>= 0.1.10), RcppArmadillo (>= 0.7.900.2.0), tibble (>= 1.2)

LinkingTo Rcpp (>= 0.12.12), RcppArmadillo (>= 0.7.900.2.0), BH (>= 1.62.0-1)

Suggests lattice, testthat, XML, rmarkdown

LazyLoad yes

NeedsCompilation yes

Collate 'RcppExports.R' 'utils.R' 'package.R' 'generics.R' 'class_modlist.R' 'class_tgrid.R' 'class_numericlist.R' 'class_matlist.R' 'class_ev.R' 'class_derived.R' 'class_mrgmod.R' 'class_mrgsims.R' 'Aaaa.R' 'altname.R' 'annot.R' 'chain.R' 'class_build.R' 'compile.R' 'complog.R'

'covset.R' 'data_set.R' 'datasets.R' 'env.R' 'events.R'
'example.R' 'funset.R' 'idata_set.R' 'init.R' 'inven.R'
'knobs.R' 'library.R' 'matlist.R' 'matrix.R' 'mcache.R'
'model_include.R' 'modlib.R' 'modspec.R' 'mread.R'
'mrgindata.R' 'mrgsims.R' 'mrgsolve.R' 'nmxml.R' 'param.R'
'print.R' 'qsim.R' 'render.R' 'simtime.R' 'update.R'

RoxygenNote 6.0.1

Author Kyle T Baron [aut, cre],
Alan C Hindmarsh [ctb],
Linda R Petzold [ctb],
Bill Gillespie [ctb],
Charles Margossian [ctb],
Devin Pastoor [ctb],
Metrum Research Group LLC [cph]

Repository CRAN

Date/Publication 2017-11-01 20:09:42 UTC

R topics documented:

`aboutsolver` *About the ODEPACK differential equation solver used by mrgsolve.*

Description

About the ODEPACK differential equation solver used by mrgsolve.

DLSODA

```
C-----------------------------------------------------------------------
C This is the 12 November 2003 version of
C DLSODA: Livermore Solver for Ordinary Differential Equations, with
C         Automatic method switching for stiff and nonstiff problems.
C
C This version is in double precision.
C
C DLSODA solves the initial value problem for stiff or nonstiff
C systems of first order ODEs,
C     dy/dt = f(t,y) ,  or, in component form,
C     dy(i)/dt = f(i) = f(i,t,y(1),y(2),...,y(NEQ)) (i = 1,...,NEQ).
C
C This a variant version of the DLSODE package.
C It switches automatically between stiff and nonstiff methods.
C This means that the user does not have to determine whether the
C problem is stiff or not, and the solver will automatically choose the
C appropriate method.  It always starts with the nonstiff method.
C
C Authors:       Alan C. Hindmarsh
C                Center for Applied Scientific Computing, L-561
C                Lawrence Livermore National Laboratory
C                Livermore, CA 94551
C and
C                Linda R. Petzold
C                Univ. of California at Santa Barbara
C                Dept. of Computer Science
C                Santa Barbara, CA 93106
C
C References:
C 1.  Alan C. Hindmarsh,  ODEPACK, A Systematized Collection of ODE
C     Solvers, in Scientific Computing, R. S. Stepleman et al. (Eds.),
C     North-Holland, Amsterdam, 1983, pp. 55-64.
C 2.  Linda R. Petzold, Automatic Selection of Methods for Solving
C     Stiff and Nonstiff Systems of Ordinary Differential Equations,
C     Siam J. Sci. Stat. Comput. 4 (1983), pp. 136-148.
C-----------------------------------------------------------------------
```

`as.list,mrgmod-method` *Coerce a model object to list.*

Description

Coerce a model object to list.

Usage

```
## S4 method for signature 'mrgmod'
as.list(x, ...)
```

Arguments

x	mrgmod object
...	passed to other methods

---

as_bmat *Coerce R objects to block or diagonal matrices.*

Description

Coerce R objects to block or diagonal matrices.

Usage

```
as_bmat(x, ...)

## S4 method for signature 'list'
as_bmat(x, ...)

## S4 method for signature 'numeric'
as_bmat(x, pat = "*", ...)

## S4 method for signature 'data.frame'
as_bmat(x, pat = "*", cols = NULL, ...)

## S4 method for signature 'ANY'
as_bmat(x, ...)

as_dmat(x, ...)

## S4 method for signature 'list'
as_dmat(x, ...)

## S4 method for signature 'ANY'
as_dmat(x, ...)

## S4 method for signature 'numeric'
as_dmat(x, pat = "*", ...)

## S4 method for signature 'data.frame'
as_dmat(x, pat = "*", cols = NULL, ...)
```

Arguments

x	an R object
...	passed along
pat	regular expression, character
cols	column names to use instead of pat

Value

A numeric matrix for list and numeric methods. For data.frames, a list of matrices are returned.

See Also

bmat, dmat

Examples

```
df <- data.frame(OMEGA1.1 = c(1,2),
                 OMEGA2.1 = c(11,22),
                 OMEGA2.2 = c(3,4),
                 SIGMA1.1 = 1,
                 FOO=-1)

as_bmat(df, "OMEGA")
as_dmat(df,"SIGMA")
as_dmat(df[1,],"OMEGA")
```

---

as_data_set *Create a simulatinon data set from ev objects.*

Description

Create a simulatinon data set from ev objects.

Usage

```
as_data_set(x, ...)

## S4 method for signature 'ev'
as_data_set(x, ...)

## S4 method for signature 'data.frame'
as_data_set(x, ...)
```

Arguments

x	ev objects
...	more ev objects

Details

The goal is to take a series of event objects and combine them into a single data set that can be passed to data_set. Each event object is added to the data frame as an ID or set of IDs that are distinct from the IDs in the other event objects. Note that including ID argument to the ev call where length(ID) is greater than one will render that set of events for all of IDs that are requested.

To get a data frame with one row (event) per ID look at expand.ev.

Value

a data frame suitable for passing into data_set

Examples

```
as_data_set(ev(amt=c(100,200), cmt=1, ID=1:3),
            ev(amt=300, time=24, ID=1:2),
            ev(amt=1000, ii=8, addl=10, ID=1:3))

# Instead of this, use expand.ev
as_data_set(ev(amt=100), ev(amt=200),ev(amt=300))
```

---

as_deslist *Create a list of designs from a data frame.*

Description

Create a list of designs from a data frame.

Usage

```
as_deslist(data, descol = "ID")
```

Arguments

data	input data set; see details
descol	character column name to be used for design groups

Details

The input data set must have a column with the same name as the value of descol. Other column names should be start (the time of the first observation), end (the time of the last observation), delta (the time steps to take between start and end), and add (other, ad-hoc times). Note that add might be a list-column to get a vector of times for each time grid object.

Value

The function returns a list of tgrid objects, one for each unique value found in descol.

Examples

```
idata <- dplyr::data_frame(ID=1:4, end=seq(24,96,24), delta=6,
add=list(c(122,124,135),c(111), c(99),c(88)))

idata <- dplyr::mutate(idata, GRP = ID %%2)

idata

l <- as_deslist(idata,"GRP")

l

lapply(l,stime)

lapply(as_deslist(idata, "ID"),stime)
```

---

blocks *Return the code blocks from a model specification file.*

Description

Return the code blocks from a model specification file.

Usage

```
blocks(x, ...)

## S4 method for signature 'mrgmod'
blocks(x, ...)

## S4 method for signature 'character'
blocks(x, ...)
```

Arguments

x	model object or path to model specification file
...	passed along

Examples

```
mod <- mrgsolve:::house()
mod %>% blocks
mod %>% blocks(PARAM,TABLE)
```

BLOCK_PARSE *Functions to parse code blocks.*

Description

Most of the basic blocks are listed in this help topic. But see also PKMODEL which has more-involved options and is documented separately.

Usage

```
PARAM(x, env, annotated = FALSE, pos = 1, ...)

FIXED(x, env, annotated = FALSE, pos = 1, ...)

THETA(x, env, annotated = FALSE, pos = 1, name = "THETA", ...)

INIT(x, env, annotated = FALSE, pos = 1, ...)

CMT(x, env, annotated = FALSE, pos = 1, ...)

CAPTURE(x, env, annotated = FALSE, pos = 1, ...)
```

Arguments

x	data
env	parse environment
annotated	logical
pos	block position
...	passed
name	block name

See Also

PKMODEL

bmat *Create matrices from vector input.*

Description

Create matrices from vector input.

Usage

```
bmat(..., correlation = FALSE, digits = -1)

cmat(..., digits = -1)

dmat(...)
```

Arguments

...	matrix data
correlation	logical; if TRUE, off diagonal elements are assumed to be correlations and converted to covariances
digits	if greater than zero, matrix is passed to signif (along with digits) prior to returning

Details

bmat makes a block matrix. cmat makes a correlation matrix. dmat makes a diagonal matrix.

See Also

as_bmat

as_dmat

Examples

```
dmat(1,2,3)/10

bmat(0.5,0.01,0.2)

cmat(0.5, 0.87,0.2)
```

---

c,matlist-method *Operations with matlist objects.*

Description

Operations with matlist objects.

Usage

```
## S4 method for signature 'matlist'
c(x, ..., recursive = FALSE)
```

Arguments

x	a matlist object
...	other matlist objects
recursive	not used

---

c,tgrid-method *Operations with tgrid objects.*

Description

Operations with tgrid objects.

Usage

```
## S4 method for signature 'tgrid'
c(x, ..., recursive = FALSE)

## S4 method for signature 'tgrids'
c(x, ..., recursive = FALSE)

## S4 method for signature 'tgrid,numeric'
e1 + e2

## S4 method for signature 'tgrid,numeric'
e1 * e2

## S4 method for signature 'tgrids,numeric'
e1 + e2

## S4 method for signature 'tgrids,numeric'
e1 * e2
```

Arguments

x	mrgmod object
...	passed along to other methods
recursive	not used
e1	tgrid or tgrids object
e2	numeric value

cama *Run the model cama function.*

Description

Run the model cama function.

Usage

```
cama(mod, fn = "cama", ...)
```

Arguments

mod	model object
fn	function name
...	passed to update

Details

sah-mah

carry_out *Select items to carry into simulated output.*

Description

When items named in this function are found in the input data set (either data_set or idata_set), they are copied into the simulated output. Special items like evid or amt or the like are not copied from the data set per se, but they are copied from datarecord objects that are created during the simulation.

Usage

```
carry_out(x, ...)

carry.out(x, ...)
```

Arguments

x	model object
...	passed along

Details

There is also a carry.out argument to mrgsim that can be set to accomplish the same thing as a call to carry_out in the pipeline.

carry.out and carry_out. Using the underscore version is now preferred.

chain *Functions for chaining commands together.*

Description

Use these functions with chaining commands togehter with the operator.

Details

Other functions that may be used in the chain of commands include: param, init, update, ev. or any other function that will take the output of the preceeding command as it's first argument.

Examples

```
mod <- mrgsolve:::house()

data(exidata)
data(exTheoph)

out <- mod %>% data_set(exTheoph) %>% mrgsim()
out <- mod %>% carry_out(evid) %>% ev(amt=100, cmt=1) %>% mrgsim()
out <- mod %>% Req(CP,RESP) %>% mrgsim()
```

cmtn *Get the compartment number from a compartment name.*

Description

Get the compartment number from a compartment name.

Usage

```
cmtn(x, ...)

## S4 method for signature 'mrgmod'
cmtn(x, tag, ...)
```

Arguments

x	model object
...	passed along
tag	compartment name

Examples

```
mod <- mrgsolve:::house()
mod %>% cmtn("CENT")
```

cmt_list-class *S4 cmt_list class*

Description

S4 cmt_list class

Details

cmt_list is a numericlist-class

code *Extract the code from a model.*

Description

Extract the code from a model.

Usage

```
code(x)
```

Arguments

x	an mrgsolve model object

Value

a character vector of model code

cvec *Create create character vectors.*

Description

Create create character vectors.

Usage

```
cvec(x, ...)

## S4 method for signature 'character'
cvec(x)

ch(...)

s(...)
```

Arguments

x	comma-separated quoted string (for cvec)
...	unquoted strings (for ch)

Examples

```
cvec("A,B,C")
ch(A,B,C)
s(A,B,C)
```

---

data_set *Select and modify a data set for simulation.*

Description

Select and modify a data set for simulation.

Usage

```
data_set(x, data, ...)

## S4 method for signature 'mrgmod,data.frame'
data_set(x, data, subset = TRUE,
  select = TRUE, object = NULL, need = NULL, ...)

## S4 method for signature 'mrgmod,ANY'
data_set(x, data, ...)

## S4 method for signature 'mrgmod,missing'
data_set(x, object, ...)
```

Arguments

x	model object
data	data set
...	passed along
subset	passed to dplyr::filter_; retain only certain rows in the data set
select	passed to dplyr::select_; retain only certain columns in the data set
object	character name of an object existing in $ENV to use for the data set
need	passed to inventory

Details

Input data sets are R data frames that can include columns with any valid name, however columns with selected names are treated specially by mrgsolve and incorporated into the simulation.

ID specifies the subject ID and is required for every input data set.

When columns have the same name as parameters ($PARAM in the model specification file), the values in those columns will be used to update the corresponding parameter as the simulation progresses.

Input data set may include the following columns related to PK dosing events: time, cmt, amt, rate, ii, addl, ss. time and cmt (and ID) are required columns in the input data set. time is the observation or event time, cmt is the compartment number (see init), amt is the dosing amount, rate is the infusion rate, ii is the dosing interval, addl specifies additional doses to administer, and ss is a flag for steady state dosing. These column names operate similarly to other non-linear mixed effects modeling software. Upper case PK dosing column names including TIME, CMT, AMT, RATE, II, ADDL, SS are also recognized. However, an error will be generated if a mix of upper case and lower case columns are found.

Only numeric data can be brought in to the problem. Any non-numeric data columns will be dropped with warning.

See exdatasets for different example data sets.

See Also

idata_set, ev, valid_data_set, valid_idata_set

Examples

```
mod <- mrgsolve:::house()

data <- expand.ev(ID=1:3, amt=c(10,20))

mod %>% data_set(data, ID > 1) %>% mrgsim

data(extran1)
head(extran1)

mod %>% data_set(extran1) %>% mrgsim
mod %>% mrgsim(data=extran1)
```

---

design *Set observation designs for the simulation.*

Description

This function also allows you to assign different designs to different groups or individuals in a population.

Usage

```
design(x, deslist = list(), descol = character(0), ...)
```

Arguments

x	model object
deslist	a list of tgrid or tgrids objects or numeric vector to be used in place of ...
descol	the idata column name (character) for design assignment
...	not used

Details

This setup requires the use of an idata_set, with individual-level data passed in one ID per row. For each ID, specify a grouping variable in idata (descol). For each unique value of the grouping variable, make one tgrid object and pass them in order as ... or form them into a list and pass as deslist.

You must assign the idata_set before assigning the designs in the command chain (see the example below).

Examples

```
peak <- tgrid(0,6,0.1)
sparse <- tgrid(0,24,6)

des1 <- c(peak,sparse)
des2 <- tgrid(0,72,4)


data <- expand.ev(ID = 1:10, amt=c(100,300))
data$GRP <- data$amt/100

idata <- data[,c("ID", "amt")]

mod <- mrgsolve:::house()

mod %>%
  omat(dmat(1,1,1,1)) %>%
  carry_out(GRP) %>%
  idata_set(idata) %>%
  design(list(des1, des2),"amt") %>%
  data_set(data) %>%
  mrgsim %>%
  plot(RESP~time|GRP)
```

details *Extract model details.*

Description

Extract model details.

Usage

```
details(x, complete = FALSE, values = FALSE, ...)
```

Arguments

x	a model object
complete	logical; if TRUE, un-annotated parameters and compartments will be added to the output
values	logical; if TRUE, a values column will be added to the output
...	not used

env_eval *Re-evaluate the code in the ENV block.*

Description

The $ENV block is a block of R code that can realize any sort of R object that might be used in running a model.

Usage

```
env_eval(x, seed = NULL)
```

Arguments

x	model object
seed	passed to set.seed if a numeric value is supplied

See Also

env_get, env_ls

env_get *Return model environment.*

Description

Return model environment.

Usage

```
env_get(x, tolist = TRUE)
```

Arguments

x	model object
tolist	should the environment be coreced to list?

env_ls *List objects in the model environment.*

Description

Each model keeps an internal environment that allows the user to carry any R object along. Objects are coded in $ENV.

Usage

```
env_ls(x, ...)
```

Arguments

x	model object
...	passed to ls

env_update *Update objects in model environment.*

Description

Update objects in model environment.

Usage

```
env_update(.x, ..., .dots = list())
```

Arguments

.x	model object
...	objects to update
.dots	list of objects to updated

ev-class *S4 events class*

Description

S4 events class

Slots

data a data frame of events

events *Event objects for simulating PK and other interventions.*

Description

Events can either be specified when the model object is created (with mrgmod) or by updating an existing model object (with update).

Usage

```
events(x, ...)

ev(x, ...)

as.ev(x, ...)

## S4 method for signature 'mrgmod'
events(x, ...)

## S4 method for signature 'mrgmod'
ev(x, object = NULL, ...)

## S4 method for signature 'missing'
ev(time = 0, evid = 1, ID = numeric(0), cmt = 1,
  replicate = TRUE, until = NULL, realize_addl = FALSE, ...)

## S4 method for signature 'ev'
ev(x, realize_addl = FALSE, ...)

## S4 method for signature 'data.frame'
as.ev(x, nid = 1, keep_id = TRUE, ...)

## S4 method for signature 'ev'
as.ev(x, ...)

## S4 method for signature 'ev'
as.matrix(x, ...)

## S4 method for signature 'ev'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S4 method for signature 'ev'
show(object)

## S4 method for signature 'mrgsims'
events(x, ...)
```

Arguments

x	mrgmodel object
...	passed on
object	passed to show
time	event time
evid	event ID
ID	subject ID
cmt	compartment

replicate	logical; if TRUE, events will be replicated for each individual in ID
until	the expected maximum **observation** time for this regimen
realize_addl	if FALSE (default), no change to addl doses. If TRUE, addl doses are made explicit with realize_addl.
nid	if greater than 1, will expand to the appropriate number of individuals
keep_id	if TRUE, ID column is retained if it exists
row.names	passed to as.data.frame
optional	passed to as.data.frame

Details

- Required input for creating events objects include time and cmt
- If not supplied, evid is assumed to be 1.
- If not supplied, cmt is assumed to be 1.
- If not supplied, time is assumed to be 0.
- ID may be specified as a vector.
- If replicate is TRUE (default), thenthe events regimen is replicated for each ID; otherwise, the number of event rows must match the number of IDs entered

Value

Returns a user-defined data frame of events that should be suitable for passing into lsoda. If events are stored as a data frame, events returns the data frame. If events are stored as a function that generates the data frame, events calls the function and passes return back to the user.

events object

Examples

```
mod <- mrgsolve:::house()
mod <- mod %>% ev(amt=1000, time=0, cmt=1)
events(mod)

loading <- ev(time=0, cmt=1, amt=1000)
maint <- ev(time=12, cmt=1, amt=500, ii=12, addl=10)
loading + maint


ev(ID=1:10, cmt=1, time=0, amt=100)
```

ev_assign *Replicate a list of events into a data set.*

Description

Replicate a list of events into a data set.

Usage

```
ev_assign(l, idata, evgroup, join = FALSE)

assign_ev(...)
```

Arguments

l	list of event objects
idata	an idata set (one ID per row)
evgroup	the character name of the column in idata that specifies event object to implement
join	if TRUE, join idata to the data set before returning.
...	used to pass arguments from assign_ev to ev_assign

Details

ev_assign connects events in a list passed in as the l argument to values in the data set identified in the evgroup argument. For making assignments, the unique values in the evgroup column are first sorted so that the first sorted unique value in evgroup is assigned to the first event in l, the second sorted value in evgroup column is assigned to the second event in l, and so on. This is a change from previous behavior, which did not sort the unique values in evgroup prior to making the assignments.

Examples

```
ev1 <- ev(amt=100)
ev2 <- ev(amt=300, rate=100, ii=12, addl=10)

idata <- data.frame(ID=1:10)
idata$arm <- 1+(idata$ID %%2)

ev_assign(list(ev1,ev2), idata, "arm", join=TRUE)
```

`ev_days` *Schedule dosing events on days of the week.*

Description

This function lets you schedule doses on specific days of the week, allowing you to create dosing regimens on Monday/Wednesday/Friday, or Tuesday/Thursday, or every other day (however you want to define that) etc.

Usage

```
ev_days(ev = NULL, days = "", addl = 0, ii = 168, unit = c("hours",
  "days"), ...)
```

Arguments

`ev`	an event object
`days`	comma- or space-separated character string of valid days of the the week (see details)
`addl`	additional doses to administer
`ii`	inter-dose interval; intended use is to keep this at the default value
`unit`	time unit; the function can only currently handle hours or days
`...`	event objects named by one the valid days of the week (see details)

Details

Valid names of the week are:

- `m` for Monday
- `t` for Tuesday
- `w` for Wednesday
- `th` for Thursday
- `f` for Friday
- `sa` for Saturday
- `s` for Sunday

The whole purpose of this function is to schedule doses on specific days of the week, in a repeating weekly schedule. Please do use caution when changing `ii` from it's default value.

Examples

```
# Monday, Wednesday, Friday x 4 weeks
ev_days(ev(amt=100), days="m,w,f", addl=3)

# 50 mg Tuesdays, 100 mg Thursdays x 6 months
ev_days(t=ev(amt=50), th=ev(amt=100), addl=23)
```

---

ev_ops *Operations for ev objects.*

Description

Operations for ev objects.

Usage

```
## S4 method for signature 'ev,ev'
e1 + e2

e1 %then% e2

## S4 method for signature 'ev,ev'
e1 %then% e2

## S4 method for signature 'ev,numeric'
e1 + e2

## S4 method for signature 'ev'
c(x, ..., recursive = TRUE)
```

Arguments

e1	object on left hand side of operator (lhs)
e2	object on right hand side of operator (rhs)
x	an ev object
...	other ev objects to collect
recursive	not used

Details

All operations involving mrgmod objects have been deprecated.

ev_rep *Replicate an event object*

Description

An event sequence can be replicated a certain number of times in a certain number of IDs.

Usage

```
ev_rep(x, id = 1, n = NULL, wait = 0, as.ev = FALSE)
```

Arguments

x	event object
id	numeric vector if IDs
n	passed to ev_repeat
wait	passed to ev_repeat
as.ev	if TRUE an event object is returned

Value

A single event object or event object as determined by the value of as.ev.

See Also

ev_repeat

Examples

```
e1 <- c(ev(amt=100), ev(amt=200, ii=24, addl=2, time=72))

ev_rep(e1, 1:5)
```

ev_repeat *Repeat a block of dosing events*

Description

Repeat a block of dosing events

Usage

```
ev_repeat(x, n, wait = 0)
```

Arguments

x	event object or dosing data frame
n	number of times to repeat
wait	time to wait between repeats

Value

A dosing data.frame.

---

ev_seq *Schedule a series of event objects*

Description

Schedule a series of event objects

Usage

```
ev_seq(..., id = NULL, .dots = NULL)

## S3 method for class 'ev'
seq(...)
```

Arguments

...	event objects or numeric arguments named wait
id	numeric vector of subject ids
.dots	a list of event objects that replaces ...

Details

The doses for the next event line start after all of the doses from the previous event line plus one dosing interval from the previous event line (see examples).

When numerics named wait are mixed in with the event objects, a period with no dosing activity is incorporated into the sequence, between the adjacent dosing event objects. Values for wait can be negative.

Values for time in any event object act like a prefix time spacer wherever that event occurs in the event sequence (see examples).

Use the generic seq when the first argument is an event object. If a waiting period is the first event, you will need to use ev_seq.

Value

A single event object.

Examples

```
e1 <- ev(amt=100, ii=12, addl=1)

e2 <- ev(amt=200)

seq(e1, e2)

seq(e1, wait = 8, e2)

seq(e1, wait = 8, e2, id = 1:10)

ev_seq(wait = 12, e1, wait = 120, e2, wait = 120, e1)

seq(ev(amt=100, ii=12), ev(time=8, amt=200))
```

`exdatasets` *Example input data sets.*

Description

Example input data sets.

Usage

```
data(exidata)

data(extran1)

data(extran2)

data(extran3)

data(exTheoph)

data(exBoot)
```

Details

- `exidata` holds individual-level parameters and other data items, one per row
- `extran1` is a "condensed" data set
- `extran2` is a full dataset
- `extran3` is a full dataset with parameters
- `exTheoph` is the theophylline data set, ready for input into `mrgsolve`
- `exBoot` a set of bootstrap parameter estimates

Examples

```
mod <- mrgsolve:::house() %>% update(end=240) %>% Req(CP)

## Full data set
data(exTheoph)
out <- mod %>% data_set(exTheoph) %>% mrgsim
out
plot(out)

## Condensed: mrgsolve fills in the observations
data(extran1)
out <- mod %>% data_set(extran1) %>% mrgsim
out
plot(out)

## Add a parameter to the data set
stopifnot(require(dplyr))
data <- extran1 %>% distinct(ID) %>% select(ID) %>%
  mutate(CL=exp(log(1.5) + rnorm(nrow(.), 0,sqrt(0.1)))) %>%
  left_join(extran1,.)

data

out <- mod %>% data_set(data) %>% carry.out(CL) %>%  mrgsim
out
plot(out)

## idata
data(exidata)
out <- mod %>% idata_set(exidata) %>% ev(amt=100,ii=24,addl=10) %>% mrgsim
plot(out, CP~time|ID)
```

`expand.idata` *Create template data sets for simulation.*

Description

Create template data sets for simulation.

Usage

```
expand.idata(...)

expand.ev(...)
```

Arguments

`...`	passed to `expand.grid`

Details

An ID column is added as `1:nrow(ans)` if not supplied by the user. For `expand.ev`, defaults also added: `cmt = 1`, `time = 0`, `evid = 1`.

Examples

```
idata <- expand.idata(CL=c(1,2,3), VC=c(10,20,30))

doses <- expand.ev(amt=c(300,100), ii=c(12,24), cmt=1)
```

---

file_show *Show model specification and C++ files.*

Description

Show model specification and C++ files.

Usage

```
file_show(x, spec = TRUE, source = TRUE, ...)
```

Arguments

`x`	model object
`spec`	logical; show the model specification file
`source`	logical; show the C++ file that is actually compiled
`...`	not used

---

house *Return a pre-compiled, PK/PD model.*

Description

Return a pre-compiled, PK/PD model.

Usage

```
house(...)
```

Arguments

`...`	passed to update

Value

A `packmod` object, ready to simulate.

Examples

```
mod <- mrgsolve:::house()

see(mod)

mod %>% ev(amt=100) %>% mrgsim %>% plot
```

`idata_set` *Select and modify a idata set for simulation.*

Description

Select and modify a idata set for simulation.

Usage

```
idata_set(x, data, ...)

## S4 method for signature 'mrgmod,data.frame'
idata_set(x, data, subset = TRUE,
  select = TRUE, object = NULL, need = NULL, ...)

## S4 method for signature 'mrgmod,ANY'
idata_set(x, data, ...)

## S4 method for signature 'mrgmod,missing'
idata_set(x, object, ...)
```

Arguments

`x`	model object
`data`	a data set coercable to data.frame
`...`	passed along
`subset`	passed to `dplyr::filter_`
`select`	passed to `dplyr::select_`
`object`	character name of an object existing in `$ENV` to use for the data set
`need`	passed to `inventory`

Details

The idata_set is a data.frame that specifies individual-level data for the problem. An ID column is required and there can be no more than one row in the data frame for each individual.

In most cases, the columns in the 'idata_set' have the same names as parameters in the param list. When this is the case, the parameter set is updated as the simulation proceeds once at the start of each individual. The 'idata_set' can also be used to set initial conditions for each individual: for a compartment called CMT, make a column in idata_set called CMT_0 and make the value the desired initial value for that compartment. Note that this initial condition will be over-ridden if you also set the CMT_0 in $MAIN.

The most common application of idata_set is to specify a population or bach of simulations to do. We commonly use idata_set with an event object (see ev). In that case, the event gets applied to each individual in the idata_set.

It is also possible to provide both a data_set and a idata_set. In this case, the idata_set is used as a parameter lookup for IDs found in the data_set. Remember in this case, it is the data_set (not the idata_set) that determines the number of individuals in the simulation.

See Also

data_set, ev

Examples

```
mod <- mrgsolve:::house()

data(exidata)

exidata

mod %>% idata_set(exidata, ID <= 2) %>% mrgsim %>% plot

mod %>% idata_set(exidata) %>% mrgsim

mod %>% mrgsim(idata=exidata)
```

---

init *Methods for working with the model compartment list.*

Description

Calling init with the model object as the first argument will return the model initial conditions as a numericlist object. See numericlist for methods to deal with cmt_list objects.

Usage

```
init(.x, ...)

## S4 method for signature 'mrgmod'
init(.x, .y = list(), ..., .pat = "*")

## S4 method for signature 'mrgsims'
init(.x, ...)

## S4 method for signature 'missing'
init(.x, ...)

## S4 method for signature 'list'
init(.x, ...)

## S4 method for signature 'ANY'
init(.x, ...)

as.init(.x, ...)

## S4 method for signature 'list'
as.init(.x, ...)

## S4 method for signature 'numeric'
as.init(.x, ...)

## S4 method for signature 'cmt_list'
as.init(.x, ...)

## S4 method for signature 'missing'
as.init(.x, ...)

## S4 method for signature '`NULL`'
as.init(.x, ...)

## S4 method for signature 'cmt_list'
show(object)
```

Arguments

.x	the model object
...	passsed along
.y	list to be merged into parameter list
.pat	a regular expression (character) to be applied as a filter when printing compartments to the screen
object	to show

Details

Can be used to either get a compartment list object from a mrgmod model object or to update the compartment initial conditions in a model object. For both uses, the return value is a cmt_list object. For the former use, init is usually called to print the compartment initial conditions to the screen, but the cmt_list object can also be coreced to a list or numeric R object.

Value

an object of class cmt_list (see numericlist)

Examples

```
## example("init")
mod <- mrgsolve:::house()

init(mod)
init(mod, .pat="^C") ## may be useful for large models

class(init(mod))

init(mod)$CENT

as.list(init(mod))
as.data.frame(init(mod))
```

---

inventory *Check whether all required parameters needed in a model are present in an object*

Description

Check whether all required parameters needed in a model are present in an object

Usage

```
inventory(x, obj, ..., .strict = FALSE)
```

Arguments

x	model object
obj	data.frame to pass to idata_set or data_set
...	capture dplyr-style parameter requirements
.strict	whether to stop execution if all requirements are present (TRUE) or just warn (FALSE); see details

Details

If parameter requirements are not explicitly stated, the requirement defaults to all parameter names in x. Note that, by default, the inventory is not .strict unless the user explicitly states the parameter requirement. That is, if parameter requirements are explicitly stated, .strict will be set to TRUE if a value .strict was not passed in the call.

Value

original mrgmod

Examples

```
## Not run:
inventory(mod, idata, CL:V) # parameters defined, inclusively, CL through Volume
inventory(mod, idata, everything()) # all parameters
inventory(mod, idata, contains("OCC")) # all parameters containing OCC
inventory(mod, idata, -F) # all parameters except F

## End(Not run)
```

---

is.mrgmod *Check if an object is a model object.*

Description

The function checks to see if the object is either mrgmod or packmod.

Usage

```
is.mrgmod(x)
```

Arguments

x	any object

Value

TRUE if x inherits mrgsims.

is.mrgsims *Check if an object is mrgsim output.*

Description

Check if an object is mrgsim output.

Usage

```
is.mrgsims(x)
```

Arguments

x	any object

Value

TRUE if x inherits mrgsims.

knobs *Run sensitivity analysis on model settings.*

Description

Knobs can be parameter values or PK dosing items (e.g. amt). By design, all combinations of specified knob/values are simulated.

Usage

```
knobs(x, y, ...)

## S4 method for signature 'mrgmod,missing'
knobs(x, y, ...)

## S4 method for signature 'mrgmod,batch_mrgsims'
knobs(x, y, ...)

## S4 method for signature 'batch_mrgsims'
as.data.frame(x, row.names = NULL,
  optional = FALSE, ...)

## S4 method for signature 'batch_mrgsims,ANY'
knobs(x, y, ...)

## S4 method for signature 'batch_mrgsims'
show(object)
```

Arguments

x	the model object
y	a batch_mrgsims object
...	knobs: named numeric vectors that identify knob names and knob values for a batch run. See details.
row.names	passed to as.data.frame
optional	passed to as.data.frame
object	passed to show

Details

Valid knob names include: any parameter name (in param(mod)), time variables (start, end, delta), PK dosing items (amt, ii, rate, and others ...), and solver settings (atol, hmax, etc...).

Value

An object of class batch_mrgsims. Most methods for mrgsims objects also work on batch_mrgsims object.

Examples

```
## example("knobs")

mod <- mrgsolve:::house(end=72)

events <- ev(amt=1000, cmt=1, addl=3, ii=12)

out <- mod %>% ev(events) %>% knobs(CL=c(1,2,3))
plot(out)

out

out <- mod %>% ev(events) %>% knobs(CL=c(1,2,3), VC=c(5,20,50))
plot(out)
plot(out,CP~.)
plot(out, CP~time|VC, groups=CL, lty=2)

out <- knobs(mod, amt=c(100,300,500), cmt=1)
plot(out)

out <- mod %>% knobs(amt=c(100,300), CL=c(1,3), VC=c(5,20), cmt=1)
plot(out)
plot(out, CP~.)

out <- knobs(mod, CL=c(1,2,3))
out

out <- knobs(mod, CL=c(1,2,3))
out
```

lctran — *Convert select upper case column names to lower case to conform to mrgsolve data expectations.*

Description

Convert select upper case column names to lower case to conform to mrgsolve data expectations.

Usage

```
lctran(data)
```

Arguments

data	an nmtran-like data frame

Details

Columns that will be renamed with lower case versions: AMT, II, SS, CMT, ADDL, RATE, EVID, TIME. If a lower case version of these names exist in the data set, the column will not be renamed.

Value

A data.frame with renamed columns.

loadso — *Load the model shared object.*

Description

Load the model shared object.

Usage

```
loadso(x, ...)

## S4 method for signature 'mrgmod'
loadso(x, ...)
```

Arguments

x	the model object
...	passed along

lower2matrix *Create a square numeric matrix from the lower-triangular elements.*

Description

Create a square numeric matrix from the lower-triangular elements.

Usage

```
lower2matrix(x, context = NULL)
```

Arguments

x	numeric data
context	the working context

Value

a square symmetric numeric matrix with column names

matlist *Methods for working with matrix-list objects.*

Description

Methods for working with matrix-list objects.

Usage

```
zero.re(.x, ...)

## S4 method for signature 'mrgmod'
zero.re(.x, ...)

zero_re(...)

drop.re(.x, ...)

## S4 method for signature 'mrgmod'
drop.re(.x, ...)

drop_re(...)

## S4 method for signature 'matlist'
as.list(x, ...)
```

```
## S4 method for signature 'matlist'
as.matrix(x, ...)

## S4 method for signature 'matlist'
names(x)

## S4 method for signature 'matlist'
length(x)

## S4 method for signature 'matlist'
labels(object, ...)

## S4 method for signature 'matlist'
dim(x)

## S4 method for signature 'matlist'
nrow(x)

## S4 method for signature 'matlist'
show(object)
```

Arguments

.x	a matlist object
...	passed along
x	a matlist object
object	passed to showmatlist

`matlist-class` *S4 class matlist.*

Description

S4 class matlist.

`mcode_cache` *Write, compile, and load model code.*

Description

This is a convenience function that ultimately calls `mread`.

Usage

```
mcode_cache(model, code, project = tempdir(), ...)

mcode(model, code, project = tempdir(), ...)
```

Arguments

model	model name
code	character string specifying a mrgsolve model
project	project name
...	passed to mread

Details

Note that the arguments are in slightly different order than mread. The default project is tempdir().

Examples

```
## Not run:
code <- '
$CMT DEPOT CENT
$PKMODEL ncmt=1, depot=TRUE
$MAIN
double CL = 1;
double V = 20;
double KA = 1;
'

mod <- mcode("example",code)

## End(Not run)
```

---

mcRNG *Set RNG to use L'Ecuyer-CMRG.*

Description

Set RNG to use L'Ecuyer-CMRG.

Usage

```
mcRNG()
```

merge.list *Merge two lists.*

Description

Merge two lists.

Usage

```
## S3 method for class 'list'
merge(x, y, ..., open = FALSE, warn = TRUE,
  context = "object", wild = "...")
```

Arguments

x	the original list
y	the new list for merging
...	not used
open	logical indicating whether or not new items should be allowed in the list upon merging.
warn	issue warning if nothing found to update
context	description of usage context
wild	wild-card name; see details

Details

Wild-card names (wild) are always retained in x and are brought along from y only when open.

mod *Return the model object.*

Description

Return the model object.

Usage

```
mod(x, ...)

## S4 method for signature 'mrgsims'
mod(x, ...)
```

Arguments

x	mrgsims object
...	passed along

modelparse *Parse model specification text.*

Description

Parse model specification text.

Usage

```
modelparse(txt, split = FALSE, drop_blank = TRUE, comment_re = c("//",
  "##"), ...)
```

Arguments

txt	model specification text
split	logical
drop_blank	logical; TRUE if blank lines are to be dropped
comment_re	regular expression for comments
...	arguments passed along

modlib *Internal model library.*

Description

Internal model library.

Usage

```
modlib(list = FALSE)
```

Arguments

list	list available models

Details

See modlib_details, modlib_pk, modlib_pkpd, modlib_tmdd, modlib_viral for details.

Call modlib(list=TRUE) to list available models. Once the model is loaded (see examples below), call mrgsolve:::code(mod) to see model code and equations.

Examples

```
## Not run:
mod <- mread("pk1cmt", modlib())
mod <- mread("pk2cmt", modlib())
mod <- mread("pk3cmt", modlib())
mod <- mread("irm1",   modlib())
mod <- mread("irm2",   modlib())
mod <- mread("irm3",   modlib())
mod <- mread("irm4",   modlib())
mod <- mread("emax",   modlib())
mod <- mread("effect", modlib())
mod <- mread("tmdd",   modlib())
mod <- mread("viral1", modlib())
mod <- mread("viral2", modlib())

mrgsolve:::code(mod)

## End(Not run)
```

`modlib_details` *modlib: PK/PD Model parameters, compartments, and output variables.*

Description

modlib: PK/PD Model parameters, compartments, and output variables.

Compartments

- `EV1`, `EV2`: extravasular dosing compartments
- `CENT`: central PK compartment
- `PERIPH`: peripheral PK compartment
- `PERIPH2`: peripheral PK compartment 2
- `RESP`: response PD compartment (irm models)

Output variables

- `CP`: concentration in the central compartment (`CENT/VC`)
- `RESP`: response (emax model)

PK parameters

- `KA1`, `KA2`: first order absorption rate constants from first and second extravascular compartment (1/time)
- `CL`: clearance (volume/time)

- VC: volume of distribution, central compartment (volume)
- VP: volume of distribution, peripheral compartment (volume)
- VP2: volume of distribution, peripheral compartment 2 (volume)
- Q: intercompartmental clearance (volume/time)
- Q2: intercompartmental clearance 2 (volume/time)
- VMAX: maximum rate, nonlinear process (mass/time)
- KM: Michaelis constant (mass/volume)
- K10: elimination rate constant (1/time); CL/VC
- K12: rate constant for transfer to peripheral compartment from central (1/time); Q/VC
- K21: rate constant for transfer to central compartment from peripheral (1/time); Q/VP

PD parameters

- E0: baseline effect (emax model)
- EMAX, IMAX: maximum effect (response)
- EC50, IC50: concentration producing 50 percent of effect (mass/volume)
- KIN: zero-order response production rate (irm models) (response/time)
- KOUT: first-order response elimination rate (irm models) (1/time)
- n: sigmoidicity factor
- KEO: rate constant for transfer to effect compartment (1/time)

---

modlib_pk *modlib: Pharmacokinetic models.*

Description

modlib: Pharmacokinetic models.

Arguments

...	passed to update

Details

See modlib_details for more detailed descriptions of parameters and compartments.

The pk1cmt model is parameterized in terms of CL, VC, KA1 and KA2 and uses compartments EV1, EV2, and CENT. The pk2cmt model adds a PERIPH compartment and parameters Q and VP to that of the one-compartment model. Likewise, the three-compartment model (pk3cmt) adds PERIPH2 and parameters Q2 and VP2 to that of the two-compartment models. All pk models also have parameters VMAX (defaulting to zero, no non-linear clearance) and KM.

Value

an object of class packmod

Model description

All pk models have two extravascular dosing compartments and potential for linear and nonlinear clearance.

- pk1cmt: one compartment pk model
- pk2cmt: two compartment pk model
- pk3cmt: three compartment pk model

---

modlib_pkpd *modlib: Pharmacokinetic / pharmacodynamic models.*

Description

modlib: Pharmacokinetic / pharmacodynamic models.

Details

See modlib_details for more detailed descriptions of parameters and compartments.

All PK/PD models include 2-compartment PK model with absorption from 2 extravasular compartments and linear + nonlinear clearance. The PK models are parameterized with CL, VC, Q, VMAX, KM, KA1 and KA2 and implement compartments EV1, EV2, CENT, PERIPH . The indirect response models have compartment RESP and the emax model has output variable RESP. PD parameters include KIN, KOUT, IC50, EC50, IMAX, EMAX, E0, and n.

Also, once the model is loaded, use see method for mrgmod to view the model code.

Model description

- irm1 inhibition of response production
- irm2 inhibition of response loss
- irm3 stimulation of response production
- irm4 stimulation of response loss
- pd_effect effect compartment model
- emax sigmoid emax model

`modlib_tmdd` *modlib: Target mediated disposition model.*

Description

modlib: Target mediated disposition model.

Arguments

...	passed to update

Parameters

- `KEL`: elimination rate constant
- `KTP`: tissue to plasma rate constant
- `KPT`: plasma to tissue rate constant
- `VC`: volume of distribution
- `KA1`, `KA2`: absorption rate constants
- `KINT`: internalization rate constant
- `KON`: association rate constant
- `KOFF`: dissociation rate constant
- `KSYN`: target systhesis rate
- `KDEG`: target degredation rate constant

Compartments

- `CENT`: unbound drug in central compartment
- `TISS`: unbound drug in tissue compartment
- `REC`: concentration of target
- `RC`: concentration of drug-target complex
- `EV1`, `EV2`: extravascular dosing compartments

Output variables

- `CP`: unbound drug in the central compartment
- `TOTAL`: total concentration of target (complexed and uncomplexed)

`modlib_viral` *modlib: HCV viral dynamics models.*

Description

modlib: HCV viral dynamics models.

Models

- `viral1`: viral dynamics model with single HCV species
- `viral2`: viral dynamics model with wild-type and mutant HCV species

Parameters

- `s`: new hepatocyte synthesis rate (cells/ml/day)
- `d`: hepatocyte death rate constant (1/day)
- `p`: viral production rate constant (copies/cell/day)
- `beta`: new infection rate constant (ml/copy/day)
- `delta`: infected cell death rate constant (1/day)
- `c`: viral clearance rate constant (1/day)
- `fit`: mutant virus fitness
- `N`: non-target hepatocytes
- `mu`: forward mutation rate
- `Tmax`: maximum number of target hepatocytes (cells/ml)
- `rho`: maximum hepatocyte regeneration rate (1/day)

Compartments

- `T`: uninfected target hepatocytes (cells/ml)
- `I`: productively infected hepatocytes (cells/ml)
- `V`: hepatitis C virus (copies/ml)
- `IM`: mutant infected hepatocytes (cells/ml)
- `VM`: mutant hepatitis C virus (copies/ml)
- `expos`: exposure metric to drive pharmacodynamic model

`modlist` *Create a modlist object.*

Description

Create a modlist object.

Usage

```
modlist(project = ".", soloc = tempdir(), prefix = "",
  pattern = paste0(prefix, "*\\.cpp$"), index_file = "MODLIST")
```

Arguments

`project`	file path to models
`soloc`	directory where the models will be built
`prefix`	leading tag for models to process
`pattern`	a regular expression for models to get
`index_file`	name of file to look for registered models

`modlist-class` *S4 class matlist.*

Description

S4 class matlist.

Usage

```
## S4 method for signature 'modlist'
x$name
```

Arguments

`x`	modlist object
`name`	model to take; used with $

modMATRIX *Create a matrix.*

Description

Create a matrix.

Usage

```
modMATRIX(x, use = TRUE, block = FALSE, correlation = FALSE,
  digits = -1, context = "matlist", ...)
```

Arguments

x	data for building the matrix. Data in x are assumed to be on-diagonal elements if block is FALSE and lower-triangular elements if block is TRUE
use	logical; if FALSE, all matrix elements are set to 0
block	logical; if TRUE, try to make a block matrix; diagonal otherwise
correlation	logical; if TRUE, off diagonal elements are assumed to be correlations and converted to covariances; if correlation is TRUE, then block is set to TRUE
digits	if value of this argument is greater than zero, the matrix is passed to signif (along with digits) prior to returning
context	the working context
...	passed along

Examples

```
modMATRIX("1 2.2 333")
modMATRIX("1 1.1 2.2", block=TRUE)
modMATRIX("23 234 234 5234", use=FALSE)

ans <- modMATRIX("1.1 0.657 2.2", correlation=TRUE, block=TRUE)
ans
cov2cor(ans)
```

mread_cache *Read a model specification file.*

Description

mread reads and parses a mrgsolve model specification file, builds the model, and returns a model object for simulation.

Usage

```
mread_cache(model = NULL, project = getwd(), file = paste0(model, ".cpp"),
  code = NULL, soloc = tempdir(), quiet = FALSE, preclean = FALSE, ...)

mread(model = NULL, project = getwd(), code = NULL, file = paste0(model,
  ".cpp"), udll = TRUE, ignore.stdout = TRUE, raw = FALSE,
  compile = TRUE, audit = TRUE, quiet = getOption("mrgsolve_mread_quiet",
  FALSE), check.bounds = FALSE, warn = TRUE, soloc = tempdir(),
  preclean = FALSE, ...)

mread_file(file, ...)
```

Arguments

model	model name
project	location of the model specification file an any headers to be included
file	the full file name (with extension, but without path) where the model is specified
code	a character string with model specification code to be used instead of a model file
soloc	directory where model shared object is stored
quiet	don't print messages when compiling
preclean	logical; if TRUE, compilation artifacts are cleaned up first
...	passed along
udll	use unique name for shared object
ignore.stdout	passed to system call for compiling model
raw	if TRUE, return a list of raw output
compile	logical; if TRUE, the model will be built
audit	check the model specification file for errors
check.bounds	check boundaries of parameter list
warn	logical; if TRUE, print warning messages that may arise

Details

When the model argument is used, mrgsolve assumes the model is written in the file with model as the stem and .cpp as the extension. Use the file argument to mread or use mread_file to read a model from a file without the .cpp extension.

Model Library

mrgsolve comes bundled with several precoded PK, PK/PD, and other systems models that are accessible via the mread interface.

Models available in the library include:

- PK models: pk1cmt, pk2cmt, pk3cmt, tmdd

- PKPD models: `irm1`, `irm2`, `irm3`, `irm4`, `emax`, `effect`
- Other models: `viral1`, `viral2`

When the library model is accessed, `mrgsolve` will compile and load the model as you would for any other model. It is only necessary to reference the correct model name and point the `project` argument to the `mrgsolve` model library location via `modlib`.

For more details, see `modlib_pk`, `modlib_pkpd`, `modlib_tmdd`, `modlib_viral`, and `modlib_details` for more information about the state variables and parameters in each model.

Examples

```
## Not run:
code <- '
$PARAM CL = 1, VC = 5
$CMT CENT
$ODE dxdt_CENT = -(CL/VC)*CENT;
'

mod <- mcode("ex_mread", code)

mod

mod %>% init(CENT=1000) %>% mrgsim %>% plot


mod <- mread("irm3", modlib())

mod

# if the model is in the file mymodel.cpp
mod <- mread("mymodel")

# if the model is in the file mymodel.txt
mod <- mread(file = "mymodel.txt")

or

mod <- mread_file("mymodel.txt")



## End(Not run)
```

`mrgmod-class` *S4 class for mrgsolve model object.*

Description

S4 class for mrgsolve model object.

Slots

`model` model name <character>
`project` working directory; must be writeable with no spaces <character>
`start` simulation start time <numeric>
`end` simulation end time <numeric>
`delta` simulation time interval <numeric>
`add` additional simulation times <numeric-vector>
`param` parameter_list
`fixed` a parameter_list of fixed value parameters; these are not updatable from R
`init` cmt_list
`events` events object
`digits` significant digits in simulated output; negative integer means ignore <numeric>
`hmin` passed to dlsoda <numeric>
`hmax` passed to dlsoda <numeric>
`mxhnil` passed to dlsoda <numeric>
`ixpr` passed to dlsoda <numeric>
`atol` passed to dlsoda <numeric>
`rtol` passed to dlsoda <numeric>
`maxsteps` passed to dlsoda <numeric>
`preclean` passed to R CMD SHLIB during compilation <logical>
`verbose` print run information to screen <logical>
`tscale` used to scale time in simulated output <numeric>
`omega` matlist for simulating individual-level random effects
`sigma` matlist for simulating residual error variates
`args` <list> of arguments to be passed to mrgsim
`advan` either 2, 4, or 13 <numeric>
`trans` either 1, 2, 4, or 11
`request` vector of compartments to request <character>
`soloc` directory path for storing the model shared object <character>
`code` a character vector of the model code
`mindt` minimum time between simulation records <numeric>
`envir` internal model environment <environment>
`annot` model annotations <list>
`plugin` model plugins <character>

Notes

- Spaces in paths (`project` and `soloc`) are prohibited.

mrgsim *Simulate from a model object.*

Description

This function sets up the simulation run from data stored in the model object as well as arguments passed in. Note that there are many non-formal arguments to this function that can be used to customize the simulation run and it's output.

Usage

```
mrgsim(x, data = NULL, idata = NULL, nid = 1, ...)
```

Arguments

x	the model objects
data	NMTRAN-like data set
idata	a matrix or data frame of model parameters, one parameter per row
nid	integer number of individuals to simulate; only used if idata and data are missing
...	passed to update

Details

- Both data and idata will be coreced to numeric matrix
- carry.out can be used to insert data columns into the output data set. This is partially dependent on the nature of the data brought into the problem.
- When using data and idata together, an error is generated if an ID occurs in data but not idata. Also, when looking up data in idata, ID in idata is assumed to be uniquely keyed to ID in data. No error is generated if ID is duplicated in data; parameters will be used from the first occurrence found in idata.
- carry.out: idata is assumed to be individual-level and variables that are carried from idata are repeated throughout the invidivual's simulated data. Variables carried from data are carried via last-observation carry forward. NA is returned from observations that are inserted into simulated output that occur prior to the first record in data.

Value

an object of class mrgsims

Additional arguments

- `mtime` numeric vector of times where the model is evaluated (with solver reset), but results are not included in simulated output
- `Request` a vector of compartment or table names to take in simulated output; if this is specified, `request` is ignored
- `obsonly` omit records with `evid` != 0 from simulated output
- `obsaug` logical; when `TRUE` and a full data set is used, the simulated output is augmented with an observation at each time in `stime`(). When using `obsaug`, a flag indicating augmented observations can be requested by including `a.u.g` in `carry.out`
- `recsort` Default value is 1. Possible values are 1,2,3,4: 1 and 2 put doses in a data set after padded observations at the same time; 3 and 4 put those doses before padded observations at the same time. 2 and 4 will put doses scheduled through `addl` after observations at the same time; 1 and 3 put doses scheduled through `addl` before observations at the same time. `recsort` will not change the order of your input data set if both doses and observations are given.
- `filbak` For each `ID`, carry the first record `data` backward to start of the simulation
- `tad` logical; when `TRUE` a column is added to simulated output is added showing the time since the last dose. Only data records with `evid == 1` will be considered doses for the purposes of `tad` calculation.
- `nocb` if `TRUE` (default), time-varying items in a data set will be implemented as next observation carried back; if `FALSE` time-varying items in a data set will be implemented as last observation carried forward.

Examples

```
## example("mrgsim")

mod <- mrgsolve:::house() %>%  ev(amt=1000, cmt=1)
out <- mod %>% mrgsim()
plot(out)

out <- mod %>% mrgsim(end=22)
out


data(exTheoph)

out <- mod %>% data_set(exTheoph) %>% mrgsim()
out

out <- mod %>% mrgsim(data=exTheoph)

out <- mrgsim(mod, data=exTheoph, obsonly=TRUE)
out

out <- mod %>% mrgsim(data=exTheoph, obsaug=TRUE, carry.out="a.u.g")
out
```

```
out <- mod %>% mrgsim(req="CENT")
out

out <- mrgsim(mod, Req="CP,RESP")
out
```

---

mrgsims *Methods for working with* mrgsims *objects.*

Description

These methods help the user view simulation output and extract simulated data to work with further. The methods listed here for the most part have generics defined by R or other R packages. See the seealso section for other methods defined by mrgsolve that have their own documentation pages.

Usage

```
## S4 method for signature 'mrgsims'
x$name

## S4 method for signature 'mrgsims'
tail(x, ...)

## S4 method for signature 'mrgsims'
head(x, ...)

## S4 method for signature 'mrgsims'
dim(x)

## S4 method for signature 'mrgsims'
names(x)

## S4 method for signature 'mrgsims'
as.data.frame(x, row.names = NULL, optional = FALSE,
  ...)

## S4 method for signature 'mrgsims'
as.matrix(x, ...)

## S4 method for signature 'mrgsims'
subset(x, ...)

## S4 method for signature 'mrgsims'
summary(object, ...)
```

```
## S4 method for signature 'mrgsims'
show(object)
```

Arguments

x	mrgsims object
name	name of column of simulated output to retain
...	passed to other functions
row.names	passed to as.data.frame
optional	passed to as.data.frame
object	passed to show

Details

Most methods should behave as expected according to other method commonly used in R (e.g. head, tail, as.data.frame, etc ...)

- subset coreces simulated output to data.frame and passes to subset.data.frame
- $ selects a column in the simulated data and returns numeric
- head see head.matrix; returns simulated data
- tail see tail.matrix; returns simulated data
- dim, nrow, ncol returns dimensions, number of rows, and number of columns in simulated data
- as.data.frame coreces simulated data to data.frame and returns the data.frame
- as.matrix returns matrix of simulated data
- as.tbl coreces simulated to tbl_df; requires dplyr
- summary coreces simulated data to data.frame and passes to summary.data.frame
- plot plots simulated data; see plot_mrgsims

See Also

stime

Examples

```
## example("mrgsims")

mod <- mrgsolve:::house() %>% init(GUT=100)

out <- mrgsim(mod)
class(out)

out
head(out)
tail(out)
```

```
mrgsolve:::mod(out)

dim(out)
names(out)

mat <- as.matrix(out)
df <- as.data.frame(out)

df <- subset(out, time < 12) ## a data frame
out$CP

plot(out)
plot(out, CP~.)
plot(out, CP+RESP~time, scales="same", xlab="Time", main="Model sims")
```

`mrgsims-class` *S4 class for mrgsolve simulation output.*

Description

S4 class for mrgsolve simulation output.

Slots

`request` character vector of compartments requested in simulated output

`outnames` character vector of column names in simulated output coming from table step

`data` matrix of simulated data

`mod` the mrgmod model object

`mrgsims_dplyr` *Methods for handling output with dplyr verbs.*

Description

Methods for handling output with dplyr verbs.

Usage

```
## S3 method for class 'mrgsims'
as.tbl(x, ...)

## S3 method for class 'mrgsims'
filter_(.data, ..., .dots)

## S3 method for class 'mrgsims'
group_by_(.data, ..., .dots, add = FALSE)

## S3 method for class 'mrgsims'
distinct_(.data, ..., .dots, .keep_all = FALSE)

## S3 method for class 'mrgsims'
mutate_(.data, ..., .dots)

summarise.each(.data, funs, ...)

## S3 method for class 'mrgsims'
summarise_(.data, ..., .dots)

## S3 method for class 'mrgsims'
do_(.data, ..., .dots)

## S3 method for class 'mrgsims'
select_(.data, ..., .dots)

## S3 method for class 'mrgsims'
slice_(.data, ...)

## S3 method for class 'mrgsims'
as_data_frame(.data_, ...)
```

Arguments

x	mrgsims object
...	passed to other methods
.data	passed to various dplyr functions
.dots	passed to various dplyr functions
add	passed to dplyr::group_by_
.keep_all	passed to dplyr::distinct_
funs	passed to dplyr::summarise_each
.data_	mrgsims object

mrgsolve *mrgsolve*

Description

mrgsolve is an R package maintained under the auspices of Metrum Research Group, LLC, that facilitates simulation from models based on systems of ordinary differential equations (ODE) that are typically employed for understanding pharmacokinetics, pharmacodynamics, and systems biology and pharmacology. mrgsovle consists of computer code written in the R and C++ languages, providing an interface to the DLSODA differential equation solver (written in FORTRAN) provided through ODEPACK - A Systematized Collection of ODE Solvers.

Example models

See `mrgsolve_example` to export example models into your own, writeable project directory.

Input data sets

See `data_set` for help creating input data sets. See `exdatasets` for example input data sets.

Package help

- Package index, including a listing of all functions
- Reserved words in `mrgsolve`: `reserved`

About the model object

The model object has class `mrgmod`.

Handling simulated output

See `mrgsims` for methods to use with simulated output.

About the solver used by `mrgsolve`

See: `aboutsolver`

Examples

```
## example("mrgsolve")

mod <- mrgsolve:::house(delta=0.1)  %>% param(CL=0.5)

events <-  ev(amt=1000, cmt=1, addl=5, ii=24)

events
```

```
mod

see(mod)

stime(mod)

param(mod)
init(mod)

out <- mod %>% ev(events) %>% mrgsim(end=168)

out

head(out)
tail(out)
dim(out)

plot(out, GUT+CP~.)

sims <- as.data.frame(out)

t72 <- subset(sims, time==72)
str(t72)

idata <- data.frame(ID=c(1,2,3), CL=c(0.5,1,2),VC=12)
out <- mod %>% ev(events) %>% mrgsim(end=168, idata=idata, req="")
plot(out)

out <- mod %>% ev(events) %>% mrgsim(carry.out="amt,evid,cmt,CL")
head(out)

out <-
  mod %>%
  ev() %>%
  knobs(CL=c(0.5, 1,2), amt=c(100,300,1000), cmt=1,end=48)

plot(out, CP~., scales="same")
plot(out, RESP+CP~time|amt,groups=CL)


ev1 <- ev(amt=500, cmt=2,rate=10)
ev2 <- ev(amt=100, cmt=1, time=54, ii=8, addl=10)
events <- ev1+ev2
events

out <- mod %>% ev(ev1+ev2) %>% mrgsim(end=180, req="")
plot(out)



## "Condensed" data set
data(extran1)
extran1
```

```
out <- mod %>% data_set(extran1) %>% mrgsim(end=200)

plot(out,CP~time|factor(ID))


## idata
data(exidata)
exidata

out <-
  mod %>%
  ev(amt=1000, cmt=1) %>%
  idata_set(exidata) %>%
  mrgsim(end=72)

plot(out, CP~., as="log10")


# Internal model library
## Not run:
mod <- mread("irm1", modlib())

mod

mod %>% ev(amt=300, ii=12, addl=3) %>% mrgsim

## End(Not run)
```

`mrgsolve_example` *Extract example model from system library*

Description

Extract example model from system library

Usage

```
mrgsolve_example(model = c("pkExample", "pkpdExample", "firstmodeExample",
  "viralExample", "popExample"), project = getwd(), overwrite = FALSE,
  quiet = FALSE, ...)
```

Arguments

`model`	name of model
`project`	working directory
`overwrite`	passed to file.copy
`quiet`	don't print any status messages to the screen
`...`	additional arguments

mrgsolve_template *Create model specification file from template*

Description

Create model specification file from template

Usage

```
mrgsolve_template(model = "template", project = getwd(),
  writeable = FALSE, overwrite = FALSE)
```

Arguments

model	name of the model to create
project	working directory
writeable	logical; if TRUE, parameters may be overwritten in the main block
overwrite	logical; if TRUE, an existing file with same stem will be overwritten

mvgauss *Simulate from a multivariate normal distribution with mean zero.*

Description

Simulate from a multivariate normal distribution with mean zero.

Usage

```
mvgauss(mat, n = 10, seed = NULL)
```

Arguments

mat	a positive-definite matrix
n	number of variates to simulate
seed	if not null, passed to set.seed

nmxml *Get THETA, OMEGA and SIGMA from a completed NONMEM run*

Description

Get THETA, OMEGA and SIGMA from a completed NONMEM run

Usage

```
nmxml(run = numeric(0), project = character(0), file = character(0),
  theta = TRUE, omega = FALSE, sigma = FALSE, olabels = NULL,
  slabels = NULL, oprefix = "", sprefix = "", tname = "THETA",
  oname = "...", sname = "...", ...)
```

Arguments

run	run number
project	project directory
file	the complete path to the run.xml file
theta	logical; if TRUE, the $THETA vector is returned
omega	logical; if TRUE, the $OMEGA matrix is returned
sigma	logical; if TRUE, the $SIGMA matrix is returned
olabels	labels for $OMEGA
slabels	labels for $SIGMA
oprefix	prefix for $OMEGA labels
sprefix	prefix for $SIGMA labels
tname	name for $THETA
oname	name for $OMEGA
sname	name for $SIGMA
...	passed along

Details

If run and project are supplied, the .xml file is assumed to be located in run.xml, in directory run off the project directory. If file is supplied, run and project arguments are ignored.

Value

a list with theta, omega and sigma elements, depending on what was requested

numeric2diag *Create a diagonal numeric matrix from diagonal elements.*

Description

Create a diagonal numeric matrix from diagonal elements.

Usage

```
numeric2diag(x, context = NULL)
```

Arguments

x	numeric data
context	used to generate column names

Value

a numeric diagonal matrix

numericlist *Methods for numericlist.*

Description

These methods can be used to corece param and init objects into common R data structures, extract elements from numericlists, or get attributes from numericlists.

Usage

```
## S4 method for signature 'numericlist'
as.list(x, ...)

## S4 method for signature 'numericlist'
as.numeric(x)

## S4 method for signature 'numericlist'
as.data.frame(x, row.names = NULL, optional = FALSE,
  ...)

## S4 method for signature 'numericlist'
length(x)

## S4 method for signature 'numericlist'
names(x)
```

```
## S4 method for signature 'numericlist'
x$name

## S4 method for signature 'numericlist'
x[i, j, ..., drop = TRUE]
```

Arguments

x	object
...	passed along to other methods
row.names	passed to as.data.frame
optional	passed to as.data.frame
name	column to take
i	elements to keep
j	not used
drop	not used

---

numericlist-class *S4 class numeric list.*

Description

S4 class numeric list.

Arguments

data	list of data
pattern	character of length 1 containing regular expression to be used as a filter when printing data to the console

---

obsaug *Augment observations in the simulated output.*

Description

Augment observations in the simulated output.

Usage

```
obsaug(x, value = TRUE, ...)
```

Arguments

x	model object
value	the value for obsaug
...	passed along There is also a obsaug argument to mrgsim that can be set to accomplish the same thing as a call to obsaug in the pipeline.

---

obsonly *Collect only observations in the simulated output.*

Description

Collect only observations in the simulated output.

Usage

```
obsonly(x, value = TRUE, ...)
```

Arguments

x	model object
value	the value for obsonly
...	passed along

Details

There is also a obsonly argument to mrgsim that can be set to accomplish the same thing as a call to obsonly in the pipeline.

---

omega *Manipulate OMEGA matrices.*

Description

The primary function is omat that can be used to both get the $OMEGA matrices out of a model object and to update $OMEGA matrices in a model object.

Usage

```
omat(.x, ...)

## S4 method for signature 'missing'
omat(.x, ...)

## S4 method for signature 'matrix'
omat(.x, ..., labels = list())

## S4 method for signature '`NULL`'
omat(.x, ...)

## S4 method for signature 'list'
omat(.x, ...)

## S4 method for signature 'omegalist'
omat(.x, ...)

## S4 method for signature 'mrgmod'
omat(.x, ..., make = FALSE, open = FALSE)

## S4 method for signature 'mrgsims'
omat(.x, make = FALSE, ...)
```

Arguments

.x	a matrix, list of matrices or matlist object
...	passed to other functions, including modMATRIX
labels	character vector of names for $OMEGA elements; must be equal to number of rows/columns in the matrix
make	logical; if TRUE, matrix list is rendered into a single matrix
open	passed to merge.list
x	matlist object

Examples

```
## example("omega")
mat1 <- matrix(1)
mat2 <- diag(c(1,2,3))
mat3 <- matrix(c(0.1, 0.002, 0.002, 0.5), 2,2)
mat4 <- dmat(0.1, 0.2, 0.3, 0.4)

omat(mat1)
omat(mat1, mat2, mat3)
omat(A=mat1, B=mat2, C=mat3)

mod <- mrgsolve:::house() %>% omat(mat4)
```

```
omat(mod)
omat(mod, make=TRUE)


## Not run:

$OMEGA
1 2 3

$OMEGA @block
1 0.1 2

$OMEGA \@cor
\@ prefix ETA_
\@ labels CL VC KA
0.1
0.67 0.2
0 0 0.3


## End(Not run)
```

`parameter_list-class` *S4 parameter_list class*

Description

S4 parameter_list class

Details

parameter_list is a `numericlist-class`

`PKMODEL` *Parse PKMODEL BLOCK data.*

Description

Parse PKMODEL BLOCK data.

Usage

```
PKMODEL(ncmt = 1, depot = FALSE, cmt = NULL, trans = pick_trans(ncmt,
  depot), env = list(), pos = 1, ...)
```

Arguments

ncmt	number of compartments; must be 1 (one-compartment, not including a depot dosing compartment) or 2 (two-compartment model, not including a depot dosing compartment)
depot	logical indicating whether to add depot compartment
cmt	compartment names as comma-delimited character
trans	the parameterization for the PK model; must be 1, 2, 4, or 11
env	parse environment
pos	block position number
...	not used

Details

When using $PKMODEL, certain symbols must be defined in the model specification depending on the value of ncmt, depot and trans.

- ncmt 1, depot FALSE, trans 2: CL, V
- ncmt 1, depot TRUE , trans 2: CL, V, KA
- ncmt 2, depot FALSE, trans 4: CL, V1, Q, V2
- ncmt 2, depot TRUE , trans 4: CL, V2, Q, V3, KA

If trans=11 is specfied, use the symbols listed above for the ncmt / depot combination, but append i at the end (e.g. CLi or Qi or KAi).

If trans=1, the user must utilize the following symbols:

- pred_CL for clearance
- pred_V or pred_V2 for central compartment volume of distribution
- pred_Q for intercompartmental clearance
- pred_V3 for for peripheral compartment volume of distribution
- pred_KA for absorption rate constant

See Also

BLOCK_PARSE

pkmodel *Simulate from 1- or 2-compartment PK model.*

Description

This is an R function that returns model objects based on $PKMODEL.

Usage

```
pkmodel(ncmt = 1, depot = FALSE, ...)
```

Arguments

ncmt	passed to PKMODEL
depot	passed to PKMODEL
...	passed to update

Details

Once the model object is generated, use param to check names of the parameters in the model and init to check the names of the compartments in the model. Calculations for the amounts in each compartment are done via analytical solutions, not differential equations. A subject-level random effect is also provided for each PK parameter; use omat to see the names of those random effects. All random effect variances have initial value of zero and may be updated via omat.

Value

An object of class mrgmod-class

Examples

```
## Not run:
mod <- pkmodel(1)

mod %>% ev(amt=1000, ii=24, addl=3) %>% mrgsim(end=120)
mod <- pkmodel(1,TRUE)
mod <- pkmodel(2)
mod <- pkmodel(2,TRUE)


## End(Not run)
```

plot,batch_mrgsims,missing-method *Plot method for mrgsims objects.*

Description

Plot method for mrgsims objects.

Usage

```
## S4 method for signature 'batch_mrgsims,missing'
plot(x, yval = variables(x),
  auto.key = list(), mincol = 3, ...)

## S4 method for signature 'batch_mrgsims,formula'
plot(x, y, show.grid = TRUE, lwd = 2,
  type = "l", yval = variables(x), auto.key = list(columns = 1),
  scales = list(y = list(relation = "free")), ...)
```

Arguments

x	mrsims object
yval	y varialbes to plot
auto.key	passed to xyplot
mincol	minimum number of columns in key
...	arguments passed to xyplot
y	a formula passed to xyplot
show.grid	print grid in the plot
lwd	passed to xyplot
type	passed to xyplot
scales	passed to xyplot

plot_mrgsims *Generate a quick plot of simulated data.*

Description

Generate a quick plot of simulated data.

Usage

```
## S4 method for signature 'mrgsims,missing'
plot(x, limit = 16, ...)

## S4 method for signature 'mrgsims,formula'
plot(x, y, limit = 16, show.grid = TRUE,
  outer = TRUE, type = "l", lwd = 2, ylab = "value", groups = ID,
  scales = list(y = list(relation = "free")), ...)
```

Arguments

x	mrgsims object
limit	limit the the number of panels to create
...	other arguments passed to xyplot
y	formula used for plotting
show.grid	logical indicating whether or not to draw panel.grid
outer	passed to xyplot
type	passed to xyplot
lwd	passed to xyplot
ylab	passed to xyplot
groups	passed to xyplot
scales	passed to xyplot

Details

Values for as argument: ; raw: raw simulated output;

Examples

```
mod <- mrgsolve:::house(end=48, delta=0.2) %>% init(GUT=1000)

out <- mrgsim(mod)

plot(out)

plot(out, subset=time <=24)

plot(out, GUT+CP~.)

plot(out, CP+RESP~time, col="black", scales="same", lty=2)
```

qsim *A quick simulation function.*

Description

A quick simulation function.

Usage

```
qsim(x, e, idata, req = NULL, tgrid = NULL)
```

Arguments

x	model object
e	event object
idata	individual data set
req	compartments to request
tgrid	tgrid object; used if e is an ev object

Examples

```
mod <- mrgsolve:::house()

des <- tgrid(0,2400,1)

data <- recmatrix(ev(amt=1000, ii=24, addl=100),des)

out <- mod %>% qsim(data)
```

realize_addl *Make addl doses explicit in an event object or data set.*

Description

Make addl doses explicit in an event object or data set.

Usage

```
realize_addl(x, ...)

## S3 method for class 'data.frame'
realize_addl(x, ...)

## S3 method for class 'ev'
realize_addl(x, ...)
```

Arguments

x	a data_set data frame or an ev object (see details)
...	not used

Details

Required data elements: addl and ii.

---

recmatrix *Create a matrix of events for simulation.*

Description

This function is for use with qsim only.

Usage

```
recmatrix(x, times, c_indexing = TRUE)
```

Arguments

x	an events object
times	object that can be coerced to numeric with stime
c_indexing	if TRUE, compartment numbers will be decremented by 1

---

relocate *Update model or project in an model object.*

Description

Update model or project in an model object.

Usage

```
relocate(x, ...)

## S4 method for signature 'mrgmod'
relocate(x, model = NULL, project = NULL)
```

Arguments

x	mrgmod object
...	passed along
model	model name
project	project directory

Value

updated model object

`rename_cols` *rename columns from vector for new names*

Description

rename columns from vector for new names

Usage

```
rename_cols(.df, new_names)
```

Arguments

`.df`	dataframe to rename
`new_names`	vector of names using syntax "<newname>" = "<oldname>"

Examples

```
rename_cols(Theoph, c("dv" = "conc", "ID" = "Subject"))
```

`render` *Render a model to a document.*

Description

Render a model to a document.

Usage

```
render(x, ...)

## S4 method for signature 'character'
render(x, project, ...)

## S4 method for signature 'mrgmod'
render(x, ...)

dorender(model, modfile, project, template = NULL, compile = TRUE, ...)
```

Arguments

x	model object or the model name
...	passed to rmarkdown::render
project	the directory containing the .cpp model file
model	model name
modfile	the model specification file name (with extension)
template	template document
compile	logical; if true, the model will be compiled to run

Examples

```
## Not run:
mod <- mrgsolve:::house()
mrgsolve:::render(mod)
mrgsolve:::render("irm2", modlib())

## End(Not run)
```

Req *Request simulated output.*

Description

Use this function to select, by name, either compartments or derived variables that have been captured (see CAPTURE).

Usage

```
Req(x, ...)

## S4 method for signature 'mrgmod'
Req(x, ...)

req(x, ...)

## S4 method for signature 'mrgmod'
req(x, ...)
```

Arguments

x	model object

`...` unquoted names of compartments or tabled items

There is also a `Req` argument to `mrgsim` that can be set to accomplish the same thing as a call to `Req` in the pipeline.

Note the difference between `req` and `Req`: the former only selects compartments to appear in output while the latter selects both compartments and captured items. Also, when there are items are explicitly listed in `Req`, all other compartments or captured items not listed there are ignored. But when compartments are selected with `req` all of the captured items are returned. Remember that `req` is strictly for compartments.

Examples

```
mod <- mrgsolve:::house()

mod %>% Req(CP,RESP) %>% ev(amt=1000) %>%  mrgsim
```

---

reserved *Reserved words.*

Description

Reserved words.

Usage

```
reserved()
```

Details

Note: this function is not exported; you must go into the `mrgsolve` namespace by using the `mrgsolve:::` prefix.

Examples

```
mrgsolve:::reserved()
```

revar *Get model random effect variances and covariances.*

Description

Get model random effect variances and covariances.

Usage

```
revar(x, ...)

## S4 method for signature 'mrgmod'
revar(x, ...)
```

Arguments

x	model object
...	passed along

scrape_and_call *Scrape options and pass to function.*

Description

Scrape options and pass to function.

Usage

```
scrape_and_call(x, env, pass, ...)
```

Arguments

x	data
env	parse environment
pass	function to call
...	dots

Details

Attributes of x are also scraped and merged with options.

scrape_opts *Scrape options from a code block.*

Description

Scrape options from a code block.

Usage

```
scrape_opts(x, envir = list(), def = list(), all = TRUE, marker = "=",
  narrow = TRUE)
```

Arguments

x	data
envir	environment from $ENV
def	default values
all	return all options, even those that are not in def
marker	assignment operator; used to locate lines with options
narrow	logical; if TRUE, only get options on lines starting with >>

Value

list with elements x (the data without options) and named options as specified in the block.

see *Print model code to the console.*

Description

Print model code to the console.

Usage

```
see(x, ...)

## S4 method for signature 'mrgmod'
see(x, raw = FALSE, ...)
```

Arguments

x	model object
...	passed along
raw	return the raw code

Value

invisible NULL

---

show,modlist-method *Show a modlist object.*

Description

Show a modlist object.

Usage

```
## S4 method for signature 'modlist'
show(object)
```

Arguments

object	modlist object

---

show,mrgmod-method *Print model details.*

Description

Print model details.

Usage

```
## S4 method for signature 'mrgmod'
show(object)
```

Arguments

object	the model object

sigma *Manipulate SIGMA matrices.*

Description

The primary function is smat that can be used to both get the $SIGMA matrices out of a model object and to update $SIGMA matrices in a model object.

Usage

```
smat(.x, ...)

## S4 method for signature 'missing'
smat(.x, ...)

## S4 method for signature 'matrix'
smat(.x, ..., labels = list())

## S4 method for signature 'list'
smat(.x, ...)

## S4 method for signature 'sigmalist'
smat(.x, ...)

## S4 method for signature 'mrgmod'
smat(.x, ..., make = FALSE, open = FALSE)

## S4 method for signature '`NULL`'
smat(.x, ...)

## S4 method for signature 'mrgsims'
smat(.x, make = FALSE, ...)
```

Arguments

.x	a matrix, list of matrices or matlist object
...	passed to other functions, including modMATRIX
labels	character vector of names for $SIGMA elements; must be equal to number of rows/columns in the matrix
make	logical; if TRUE, matrix list is rendered into a single matrix
open	passed to merge.list
x	matlist object

Examples

```
## example("sigma")
mat1 <- matrix(1)
mat2 <- diag(c(1,2))
mat3 <- matrix(c(0.1, 0.002, 0.002, 0.5), 2,2)
mat4 <- dmat(0.1, 0.2, 0.3, 0.4)

smat(mat1)
smat(mat1, mat2, mat3)
smat(A=mat1, B=mat2, C=mat3)

mod <- mrgsolve:::house() %>% smat(mat1)

smat(mod)
smat(mod, make=TRUE)
```

---

simargs *Access or clear arguments for calls to mrgsim.*

Description

Access or clear arguments for calls to mrgsim.

Usage

```
simargs(x, ...)

## S3 method for class 'mrgmod'
simargs(x, clear = FALSE, ...)
```

Arguments

x	model object
...	passed along
clear	logical indicating whether or not clear args from the model object

Value

If clear is TRUE, the argument list is cleared and the model object is returned. Otherwise, the argument list is returned.

Examples

```
mod <- mrgsolve:::house()
mod %>% Req(CP,RESP) %>% carry_out(evid,WT,FLAG) %>% simargs
```

soloc *Return the location of the model shared object.*

Description

Return the location of the model shared object.

Usage

```
soloc(x, short = FALSE)
```

Arguments

x	model object
short	logical; if TRUE, soloc will be rendered with a short path name

Examples

```
mod <- mrgsolve:::house()
soloc(mod)
```

stime *Get the times at which the model will be evaluated.*

Description

Get the times at which the model will be evaluated.

Usage

```
stime(x, ...)
```

Arguments

x	object of class mrgmod
...	passed on

Details

Simulation times include the sequence of times created from start, end, and delta and the vector of times found in add. Making end negative will omit any start / end / delta sequence. Negative values are discarded from the result.

Value

a sorted vector of unique times

Examples

```
## example("stime", package="mrgsolve")

mod <- mrgsolve:::house(end=12, delta=2, add=c(11,13,15))

stime(mod)
```

`stime,mrgmod-method` *Create a simtime object.*

Description

simtime objects allow the user to specify simulation start and end times, along with the simulation time step.

Usage

```
## S4 method for signature 'mrgmod'
stime(x, ...)

tgrid(start = 0, end = 24, delta = 1, add = numeric(0), .offset = 0,
  .scale = 1, ...)

## S4 method for signature 'tgrid'
stime(x, ...)

## S4 method for signature 'tgrids'
stime(x, ...)

## S4 method for signature 'numeric'
stime(x, ...)

## S4 method for signature 'tgrid'
show(object)

## S4 method for signature 'tgrids'
show(object)
```

Arguments

x	tgrid object
...	passed on to other methods
start	simulation start time
end	simulation end time
delta	simulation time step
add	addition simulation times
.offset	the resulting set of times will be adjusted by this amount
.scale	the resulting set of times will be scaled by this factor
object	passed to show

Examples

```
peak <- tgrid(0,6,0.2)
sparse <- tgrid(0,24,4)

day1 <- c(peak,sparse)

design <- c(day1, day1+72, day1+240)

## Not run:
mod <- mrgsolve:::house()

out <- mod %>% ev(amt=1000, ii=24, addl=10) %>% mrgsim(tgrid=design)

plot(out,CP~., type='b')

## End(Not run)
```

---

touch_funs *Get inits from compiled function.*

Description

Get inits from compiled function.

Usage

```
touch_funs(x, keep_pointers = TRUE)
```

Arguments

x	mrgmod model object
keep_pointers	should function pointers be returned?

tscale *Rescale time in the simulated output.*

Description

Rescale time in the simulated output.

Usage

```
tscale(x, value = 1, ...)
```

Arguments

x	model object
value	value by which time will be scaled
...	passed along

Details

There is also a `tscale` argument to `mrgsim` that can be set to accomplish the same thing as a call to `tscale` in the pipeline.

Examples

```
# The model is in hours:
mod <- mrgsolve:::house()

# The output is in days:
mod %>% tscale(1/24) %>% mrgsim
```

update *Get all names from a model object.*

Description

Get all names from a model object.

After the model object is created, update various attributes.

Usage

```
## S4 method for signature 'mrgmod'
names(x)

## S4 method for signature 'mrgmod'
update(object, ..., merge = TRUE, open = FALSE,
  data = list())

## S4 method for signature 'omegalist'
update(object, y, ...)

## S4 method for signature 'sigmalist'
update(object, y, ...)

## S4 method for signature 'parameter_list'
update(object, y, ...)

## S4 method for signature 'ev'
update(object, y, ...)
```

Arguments

x	the model object
object	a model object
...	passed to other functions
merge	logical indicating to merge (rather than replace) new and existing attributes.
open	logical; used only when merge is TRUE and parameter list or initial conditions list is being updated; if FALSE, no new items will be added; if TRUE, the parameter list may expand.
data	a list of items to update; not used for now
y	another object involved in update

Value

The updated model object is returned.

Examples

```
mod <- mrgsolve:::house()
names(mod)

## Not run:
 mod <- mrgsolve:::house()

 mod <- update(mod, end=120, delta=4, param=list(CL=19.1))

## End(Not run)
```

valid_data *Validate and prepare data sets for simulation.*

Description

Validate and prepare data sets for simulation.

Validate and prepare idata data sets for simulation.

Usage

```
valid_data_set(x, ...)

## Default S3 method:
valid_data_set(x, ...)

## S3 method for class 'data.frame'
valid_data_set(x, m = NULL, verbose = FALSE,
  quiet = FALSE, ...)

valid_idata_set(x, verbose = FALSE, quiet = FALSE, ...)

## S3 method for class 'matrix'
valid_data_set(x, verbose = FALSE, ...)
```

Arguments

x	data.frame or matrix
...	additional arguments
m	a model object
verbose	logical
quiet	if TRUE, messages will be suppressed

Value

a matrix with non-numeric columns dropped; if x is a data.frame with character cmt column comprised of valid compartment names and m is a model object, the cmt column will be converted to the corresponding compartment number.

A numeric matrix with class valid_idata_set.

See Also

idata_set, data_set, valid_data_set

`$,mrgmod-method` *Create and work with parameter objects.*

Description

See `numericlist` for methods to deal with `parameter_list` objects.

Usage

```
## S4 method for signature 'mrgmod'
x$name

param(.x, ...)

## S4 method for signature 'mrgmod'
param(.x, .y = list(), ..., .pat = "*",
  .strict = FALSE)

## S4 method for signature 'mrgsims'
param(.x, ...)

## S4 method for signature 'missing'
param(..., .strict = TRUE)

## S4 method for signature 'list'
param(.x, ...)

## S4 method for signature 'ANY'
param(.x, ...)

as.param(.x, ...)

## S4 method for signature 'list'
as.param(.x, ...)

## S4 method for signature 'numeric'
as.param(.x, ...)

## S4 method for signature 'parameter_list'
as.param(.x, ...)

## S4 method for signature 'missing'
as.param(.x, ...)

## S4 method for signature 'parameter_list'
show(object)
```

```
allparam(.x)
```

Arguments

x	mrgmod object
name	parameter to take
.x	the model object
...	passed along or name/value pairs to update the parameters in a model object
.y	list to be merged into parameter list
.pat	a regular expression (character) to be applied as a filter for which parameters to show when printing
.strict	if TRUE, all names to be updated must be found in the parameter list
object	passed to show

Details

Can be used to either get a parameter list object from a mrgmod model object or to update the parameters in a model object. For both uses, the return value is a parameter_list object. For the former use, param is usually called to print the parameters to the screen, but the parameter_list object can also be coreced to a list or numeric R object.

Value

An object of class parameter_list (see numericlist).

Examples

```
## example("param")
mod <- mrgsolve:::house()

param(mod)
param(mod, .pat="^(C|F)") ## may be useful when large number of parameters

class(param(mod))

param(mod)$KA

as.list(param(mod))
as.data.frame(param(mod))
```

%>% *Forward pipe.*

Description

Forward pipe.

Tee.

Index