

# Package ‘RxODE’

May 11, 2021

**Version** 1.0.9

**Title** Facilities for Simulating from ODE-Based Models

**Maintainer** Wenping Wang <wwang8198@gmail.com>

**Depends** R (>= 4.0.0)

**Suggests** spelling, Matrix, DT, covr, crayon, curl, data.table (>= 1.12.4), digest, dplyr (>= 0.8.0), ggrepel, gridExtra, htmltools, knitr, learnr, microbenchmark, nlme, remotes, rlang, rmarkdown, scales, shiny, stringi, symengine, testthat, tidyr, usethis, vdiff, xgxr, pillar, tibble, units (>= 0.6-0), rsconnect, devtools, patchwork

**Imports** PreciseSums (>= 0.3), Rcpp (>= 0.12.3), assertthat, backports, checkmate, cli (>= 2.0.0), dparser (>= 0.1.8), ggplot2, inline, lotri (>= 0.3.0), magrittr, memoise, methods, rex, qs, sys, tools, utils

**Description** Facilities for running simulations from ordinary differential equation ('ODE') models, such as pharmacometrics and other compartmental models. A compilation manager translates the ODE model into C, compiles it, and dynamically loads the object code into R for improved computational efficiency. An event table object facilitates the specification of complex dosing regimens (optional) and sampling schedules. NB: The use of this package requires both C and Fortran compilers, for details on their use with R please see Section 6.3, Appendix A, and Appendix D in the "R Administration and Installation" manual. Also the code is mostly released under GPL. The 'VODE' and 'LSODA' are in the public domain. The information is available in the inst/COPYRIGHTS.

**BugReports** <https://github.com/nlmixrdevelopment/RxODE/issues/>

**NeedsCompilation** yes

**VignetteBuilder** knitr

**License** GPL (>= 3)

**URL** <https://nlmixrdevelopment.github.io/RxODE/>,  
<https://github.com/nlmixrdevelopment/RxODE/>

**RoxygenNote** 7.1.1

**Biarch** true

**LinkingTo** dparser (>= 1.3.1-0), PreciseSums (>= 0.3), RcppEigen (>= 0.3.3.3.0), RcppArmadillo (>= 0.9.300.2.0), sitmo, StanHeaders (>= 2.21.0), BH

**Encoding** UTF-8

**LazyData** true

**Language** en-US

**Author** Matthew L. Fidler [aut] (<<https://orcid.org/0000-0001-8538-6691>>),  
 Melissa Hallow [aut],  
 Wenping Wang [aut, cre],  
 Zufar Mulyukov [ctb],  
 Alan Hindmarsh [ctb],  
 Awad H. Al-Mohy [ctb],  
 Matt Dowle [ctb],  
 Cleve Moler [ctb],  
 David Cooley [ctb],  
 Drew Schmidt [ctb],  
 Arun Srinivasan [ctb],  
 Ernst Hairer [ctb],  
 Gerhard Wanner [ctb],  
 Goro Fuji [ctb],  
 Hadley Wickham [ctb],  
 Jack Dongarra [ctb],  
 Linda Petzold [ctb],  
 Martin Maechler [ctb],  
 Matteo Fasiolo [ctb],  
 Morwenn [ctb],  
 Nicholas J. Higham [ctb],  
 Roger B. Sidje [ctb],  
 Simon Frost [ctb],  
 Kevin Ushey [ctb],  
 Yu Feng [ctb]

**Repository** CRAN

**Date/Publication** 2021-05-11 05:12:17 UTC

## R topics documented:

.rxFinalizeInner . . . . .	5
.rxFinalizePred . . . . .	6
.rxGenEBE . . . . .	6
.rxGenEtaS . . . . .	7
.rxGenFoce . . . . .	8
.rxGenFocei . . . . .	9
.rxGenFunState . . . . .	10

.rxGenHdEta	11
.rxGenPred	12
.rxRmPrint	13
.rxWinRtoolsPath	13
.rxWithOptions	14
.rxWithWd	14
.setWarnIdSort	15
add.dosing	16
add.sampling	19
as.data.table.rxEt	21
as.et	22
as_tibble.rxEt	22
coef.RxODE	23
cvPost	23
et	26
etExpand	31
etRbind	32
etRep	35
etSeq	38
eventTable	41
forderForceBase	43
gammap	43
gammapDer	44
gammapInv	45
gammaq	46
gammaqInv	47
genShinyApp.template	48
getRxThreads	49
guide_none	50
invWR1d	51
is.rxEt	51
is.rxSolve	52
logit	52
lowergamma	54
phi	55
print.RxODE	55
probit	56
rinvchisq	57
rLKJ1	57
rxAddReturn	58
rxAllowUnload	58
rxAssignPtr	59
rxbeta	60
rxbinom	61
rxcauchy	62
rxChain	64
rxchisq	64
rxClean	66

rxCompile . . . . .	66
rxCreateCache . . . . .	68
rxD . . . . .	69
rxDelete . . . . .	69
rxDerived . . . . .	70
rxDfdy . . . . .	72
rxEvid . . . . .	72
rxexp . . . . .	74
rxf . . . . .	75
rxFun . . . . .	76
rxgamma . . . . .	78
rxGenSaem . . . . .	79
rxgeom . . . . .	80
rxGetLin . . . . .	81
rxGetRxODE . . . . .	82
rxHtml . . . . .	83
rxIndLinState . . . . .	83
rxIndLinStrategy . . . . .	84
rxInv . . . . .	84
rxIsCurrent . . . . .	85
rxLhs . . . . .	85
rxLock . . . . .	86
rxNorm . . . . .	86
rxnorm . . . . .	87
RxODE . . . . .	88
rxOptExpr . . . . .	95
rxParams . . . . .	96
rxPkg . . . . .	98
rxpois . . . . .	99
rxPp . . . . .	100
rxProgress . . . . .	102
rxRandNV . . . . .	103
rxRateDur . . . . .	104
rxReservedKeywords . . . . .	104
rxRmvn . . . . .	105
rxS . . . . .	107
rxSetIni0 . . . . .	108
rxSetProd . . . . .	109
rxSetProgressBar . . . . .	109
rxSetSum . . . . .	110
rxShiny . . . . .	110
rxSimThetaOmega . . . . .	111
rxSolve . . . . .	115
rxStack . . . . .	127
rxState . . . . .	127
rxSumProdModel . . . . .	128
rxSupportedFuns . . . . .	129
rxSuppressMsg . . . . .	129

rxSymInvChol . . . . .	130
rxSyncOptions . . . . .	131
rxSyntaxFunctions . . . . .	132
rxT . . . . .	132
rxTempDir . . . . .	133
rxTheme . . . . .	134
rxToSE . . . . .	134
rxTrans . . . . .	135
rxunif . . . . .	137
rxUnloadAll . . . . .	138
rxUse . . . . .	139
rxValidate . . . . .	139
rxweibull . . . . .	140
rxWinSetup . . . . .	141
stat_amt . . . . .	142
stat_cens . . . . .	143
summary.RxODE . . . . .	145
uppergamma . . . . .	145

**Index** **147**

.rxFinalizeInner      *Finalize inner RxODE based on symengine saved info*

**Description**

Finalize inner RxODE based on symengine saved info

**Usage**

```
.rxFinalizeInner(.s, sum.prod = FALSE, optExpression = TRUE)
```

**Arguments**

- .s                      Symengine/RxODE object
- sum.prod              A boolean determining if RxODE should use more numerically stable sums/products.
- optExpression        Optimize the model text for computer evaluation.

**Value**

Nothing

**Author(s)**

Matthew L Fidler

---

.rxFinalizePred            *Finalize RxODE pred based on symengine saved info*

---

**Description**

Finalize RxODE pred based on symengine saved info

**Usage**

```
.rxFinalizePred(.s, sum.prod = FALSE, optExpression = TRUE)
```

**Arguments**

.s                            Symengine/RxODE object  
sum.prod                    A boolean determining if RxODE should use more numerically stable sums/products.  
optExpression            Optimize the model text for computer evaluation.

**Value**

Nothing

**Author(s)**

Matthew L Fidler

---

.rxGenEBE                    *Generate pieces for EBE only problem*

---

**Description**

Generate pieces for EBE only problem

**Usage**

```
.rxGenEBE(  
  obj,  
  predfn,  
  pkpars = NULL,  
  errfn = NULL,  
  init = NULL,  
  pred.minus.dv = TRUE,  
  sum.prod = FALSE,  
  optExpression = TRUE,  
  theta = FALSE,  
  addProp = c("combined2", "combined1")  
)
```

**Arguments**

obj	RxODE object
predfn	Prediction function
pkpars	Pk Pars function
errfn	Error function
init	Initialization parameters for scaling.
pred.minus.dv	Boolean stating if the FOCEi objective function is based on PRED-DV (like NONMEM). Default TRUE.
sum.prod	A boolean determining if RxODE should use more numerically stable sums/products.
optExpression	Optimize the model text for computer evaluation.
theta	Calculate THETA derivatives instead of ETA derivatives. By default FALSE
addProp	one of "combined1" and "combined2"; These are the two forms of additive+proportional errors supported by monolix/nonmem: combined1: $\text{transform}(y)=\text{transform}(f)+(a+b*f^c)*\text{eps}$ combined2: $\text{transform}(y)=\text{transform}(f)+(a^2+b^2*f^{(2c)})*\text{eps}$

**Value**

RxODE/symengine environment

**Author(s)**

Matthew L. Fidler

---

.rxGenEtaS

*Generate the ETA sensitivities for FO related methods*

---

**Description**

Generate the ETA sensitivities for FO related methods

**Usage**

```
.rxGenEtaS(  
  obj,  
  predfn,  
  pkpars = NULL,  
  errfn = NULL,  
  init = NULL,  
  promoteLinSens = TRUE,  
  theta = FALSE,  
  addProp = c("combined2", "combined1")  
)
```

**Arguments**

obj	RxODE object
predfn	Prediction function
pkpars	Pk Pars function
errfn	Error function
init	Initialization parameters for scaling.
promoteLinSens	Promote solved linear compartment systems to sensitivity-based solutions.
theta	Calculate THETA derivatives instead of ETA derivatives. By default FALSE
addProp	one of "combined1" and "combined2"; These are the two forms of additive+proportional errors supported by monolix/nonmem: combined1: $\text{transform}(y)=\text{transform}(f)+(a+b*f^c)*\text{eps}$ combined2: $\text{transform}(y)=\text{transform}(f)+(a^2+b^2*f^{(2c)})*\text{eps}$

**Value**

RxODE/symengine environment

**Author(s)**

Matthew L. Fidler

---

.rxGenFoce

*Generate FOCE without interaction*

---

**Description**

Generate FOCE without interaction

**Usage**

```
.rxGenFoce(
  obj,
  predfn,
  pkpars = NULL,
  errfn = NULL,
  init = NULL,
  pred.minus.dv = TRUE,
  sum.prod = FALSE,
  optExpression = TRUE,
  promoteLinSens = TRUE,
  theta = FALSE,
  addProp = c("combined2", "combined1")
)
```



**Arguments**

obj	RxODE object
predfn	Prediction function
pkpars	Pk Pars function
errfn	Error function
init	Initialization parameters for scaling.
pred.minus.dv	Boolean stating if the FOCEi objective function is based on PRED-DV (like NONMEM). Default TRUE.
sum.prod	A boolean determining if RxODE should use more numerically stable sums/products.
optExpression	Optimize the model text for computer evaluation.
promoteLinSens	Promote solved linear compartment systems to sensitivity-based solutions.
theta	Calculate THETA derivatives instead of ETA derivatives. By default FALSE
addProp	one of "combined1" and "combined2"; These are the two forms of additive+proportional errors supported by monolix/nonmem: combined1: $\text{transform}(y)=\text{transform}(f)+(a+b*f^c)*\text{eps}$ combined2: $\text{transform}(y)=\text{transform}(f)+(a^2+b^2*f^{(2c)})*\text{eps}$

**Value**

RxODE/symengine environment

**Author(s)**

Matthew Fidler

---

.rxGenFocei                      *Generate pieces for FOCEi inner problem*

---

**Description**

Generate pieces for FOCEi inner problem

**Usage**

```
.rxGenFocei(  
  obj,  
  predfn,  
  pkpars = NULL,  
  errfn = NULL,  
  init = NULL,  
  pred.minus.dv = TRUE,  
  sum.prod = FALSE,  
  optExpression = TRUE,
```

```

    promoteLinSens = TRUE,
    theta = FALSE,
    addProp = c("combined2", "combined1")
  )

```

### Arguments

obj	RxODE object
predfn	Prediction function
pkpars	Pk Pars function
errfn	Error function
init	Initialization parameters for scaling.
pred.minus.dv	Boolean stating if the FOCEi objective function is based on PRED-DV (like NONMEM). Default TRUE.
sum.prod	A boolean determining if RxODE should use more numerically stable sums/products.
optExpression	Optimize the model text for computer evaluation.
promoteLinSens	Promote solved linear compartment systems to sensitivity-based solutions.
theta	Calculate THETA derivatives instead of ETA derivatives. By default FALSE
addProp	one of "combined1" and "combined2"; These are the two forms of additive+proportional errors supported by monolix/nonmem: combined1: $\text{transform}(y)=\text{transform}(f)+(a+b*f^c)*\text{eps}$ combined2: $\text{transform}(y)=\text{transform}(f)+(a^2+b^2*f^{(2c)})*\text{eps}$

### Value

RxODE/symengine environment

### Author(s)

Matthew L. Fidler

---

.rxGenFunState

*Get the state information for the current model*

---

### Description

Get the state information for the current model

### Usage

```
.rxGenFunState(obj)
```

### Arguments

obj	RxODE model that can take rxModelVars
-----	---------------------------------------

**Value**

character vector of initial preserved states (state), extra states (statef) and dvid translation information (dvid). This is used in generating the final RxODE model.

**Author(s)**

Matthew Fidler

---

.rxGenHdEta	<i>Generate the d(err)/d(eta) values for FO related methods</i>
-------------	---

---

**Description**

Generate the d(err)/d(eta) values for FO related methods

**Usage**

```
.rxGenHdEta(
  obj,
  predfn,
  pkpars = NULL,
  errfn = NULL,
  init = NULL,
  pred.minus.dv = TRUE,
  promoteLinSens = TRUE,
  theta = FALSE,
  addProp = c("combined2", "combined1")
)
```

**Arguments**

obj	RxODE object
predfn	Prediction function
pkpars	Pk Pars function
errfn	Error function
init	Initialization parameters for scaling.
pred.minus.dv	Boolean stating if the FOCEi objective function is based on PRED-DV (like NONMEM). Default TRUE.
promoteLinSens	Promote solved linear compartment systems to sensitivity-based solutions.
theta	Calculate THETA derivatives instead of ETA derivatives. By default FALSE
addProp	one of "combined1" and "combined2"; These are the two forms of additive+proportional errors supported by monolix/nonmem: combined1: $transform(y)=transform(f)+(a+b*f^c)*eps$ combined2: $transform(y)=transform(f)+(a^2+b^2*f^(2c))*eps$

**Value**

RxODE/symengine environment

**Author(s)**

Matthew L. Fidler

---

.rxGenPred

*Generate Augmented pred/err RxODE model*

---

**Description**

Generate Augmented pred/err RxODE model

**Usage**

```
.rxGenPred(obj, predfn, errfn, init, addProp = c("combined2", "combined1"))
```

**Arguments**

obj	RxODE object
predfn	Prediction function
errfn	Error function
init	Initialization parameters for scaling.
addProp	one of "combined1" and "combined2"; These are the two forms of additive+proportional errors supported by monolix/nonmem: combined1: $\text{transform}(y)=\text{transform}(f)+(a+b*f^c)*\text{eps}$ combined2: $\text{transform}(y)=\text{transform}(f)+(a^2+b^2*f^{(2c)})*\text{eps}$

**Value**

A list of (1) RxODE model variables augmented with pred/error information and (2) extra error variables created.

**Author(s)**

Matthew L Fidler

---

.rxRmPrint                    *Remove print statements*

---

**Description**

Remove print statements

**Usage**

```
.rxRmPrint(x)
```

**Arguments**

x                    RxODE lines to remove

**Value**

RxODE with print lines removed.

**Author(s)**

Matthew L. Fidler

---

.rxWinRtoolsPath            *Setup Rtools path*

---

**Description**

Setup Rtools path

**Usage**

```
.rxWinRtoolsPath(rm.rtools = TRUE, retry = FALSE)
```

**Arguments**

rm.rtools            Remove the Rtools from the current path specs.  
retry                Should you retry to find Rtools? If NA, don't throw an error if it isn't found.

**Author(s)**

Matthew L. Fidler

---

`.rxWithOptions`      *Temporarily set options then restore them while running code*

---

**Description**

Temporarily set options then restore them while running code

**Usage**

```
.rxWithOptions(ops, code)
```

**Arguments**

ops	list of options that will be temporarily set for the code
code	The code to run during the sink

**Value**

value of code

**Examples**

```
.rxWithOptions(list(digits=21), {  
  print(pi)  
})  
  
print(pi)
```

---

`.rxWithWd`      *Temporarily set options then restore them while running code*

---

**Description**

Temporarily set options then restore them while running code

**Usage**

```
.rxWithWd(wd, code)
```

**Arguments**

wd	working directory to temporarily set the system to while evaluating the code
code	The code to run during the sink

**Value**

value of code

**Examples**

```
.rxWithWd(tempdir(), {  
  getwd()  
})  
  
getwd()
```

---

`.setWarnIdSort`      *Turn on/off warnings for ID sorting.*

---

**Description**

Turn on/off warnings for ID sorting.

**Usage**

```
.setWarnIdSort(warnIdSort = TRUE)
```

**Arguments**

`warnIdSort`      Boolean for if the sorting warning is turned on or off.

**Value**

Nothing

**Author(s)**

Matthew Fidler

---

add.dosing	<i>Add dosing to eventTable</i>
------------	---------------------------------

---

### Description

This adds a dosing event to the event table. This is provided for piping syntax through magrittr. It can also be accessed by `eventTable$add.dosing(...)`

### Usage

```
add.dosing(
  eventTable,
  dose,
  nbr.doses = 1L,
  dosing.interval = 24,
  dosing.to = 1L,
  rate = NULL,
  amount.units = NA_character_,
  start.time = 0,
  do.sampling = FALSE,
  time.units = NA_character_,
  ...
)
```

### Arguments

<code>eventTable</code>	eventTable object; When accessed from object it would be <code>eventTable\$</code>
<code>dose</code>	numeric scalar, dose amount in <code>amount.units</code> ;
<code>nbr.doses</code>	integer, number of doses;
<code>dosing.interval</code>	required numeric scalar, time between doses in <code>time.units</code> , defaults to 24 of <code>time.units="hours"</code> ;
<code>dosing.to</code>	integer, compartment the dose goes into (first compartment by default);
<code>rate</code>	for infusions, the rate of infusion (default is NULL, for bolus dosing);
<code>amount.units</code>	optional string indicating the dosing units. Defaults to NA to indicate as per the original EventTable definition.
<code>start.time</code>	required dosing start time;
<code>do.sampling</code>	logical, should observation sampling records be added at the dosing times? Defaults to FALSE.
<code>time.units</code>	optional string indicating the time units. Defaults to "hours" to indicate as per the original EventTable definition.
<code>...</code>	Other parameters passed to <code>et()</code> .



**Value**

eventTable with updated dosing (note the event table will be updated anyway)

**Author(s)**

Matthew L. Fidler

Matthew L Fidler, Wenping Wang

**References**

Wang W, Hallow K, James D (2015). "A Tutorial on RxODE: Simulating Differential Equation Pharmacometric Models in R." *CPT: Pharmacometrics & Systems Pharmacology*, 5(1), 3-10. ISSN 2163-8306, <URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4728294/>>.

**See Also**

[eventTable](#), [add.sampling](#), [add.dosing](#), [et](#), [etRep](#), [etRbind](#), [RxODE](#)

**Examples**

```
library(RxODE)
library(units)

## Model from RxODE tutorial
mod1 <- RxODE({
  KA=2.94E-01;
  CL=1.86E+01;
  V2=4.02E+01;
  Q=1.05E+01;
  V3=2.97E+02;
  Kin=1;
  Kout=1;
  EC50=200;
  C2 = centr/V2;
  C3 = peri/V3;
  d/dt(depot) = -KA*depot;
  d/dt(centr) = KA*depot - CL*C2 - Q*C2 + Q*C3;
  d/dt(peri) = Q*C2 - Q*C3;
  d/dt(eff) = Kin - Kout*(1-C2/(EC50+C2))*eff;
});

## These are making the more complex regimens of the RxODE tutorial

## bid for 5 days
bid <- et(timeUnits="hr") %>%
  et(amt=10000,ii=12,until=set_units(5, "days"))

## qd for 5 days
```

```
qd <- et(timeUnits="hr") %>%
  et(amt=20000,ii=24,until=set_units(5, "days"))

## bid for 5 days followed by qd for 5 days

et <- seq(bid,qd) %>% et(seq(0,11*24,length.out=100));

bidQd <- rxSolve(mod1, et)

plot(bidQd, C2)

## Now Infusion for 5 days followed by oral for 5 days

## note you can dose to a named compartment instead of using the compartment number
infusion <- et(timeUnits = "hr") %>%
  et(amt=10000, rate=5000, ii=24, until=set_units(5, "days"), cmt="centr")

qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(5, "days"), cmt="depot")

et <- seq(infusion,qd)

infusionQd <- rxSolve(mod1, et)

plot(infusionQd, C2)

## 2wk-on, 1wk-off

qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")

et <- seq(qd, set_units(1,"weeks"), qd) %>%
  add.sampling(set_units(seq(0, 5.5,by=0.005),weeks))

wkOnOff <- rxSolve(mod1, et)

plot(wkOnOff, C2)

## You can also repeat the cycle easily with the rep function

qd <-et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")

et <- etRep(qd, times=4, wait=set_units(1,"weeks")) %>%
  add.sampling(set_units(seq(0, 12.5,by=0.005),weeks))

repCycle4 <- rxSolve(mod1, et)

plot(repCycle4, C2)
```

---

add.sampling	<i>Add sampling to eventTable</i>
--------------	-----------------------------------

---

### Description

This adds a dosing event to the event table. This is provided for piping syntax through magrittr. It can also be accessed by `eventTable$add.sampling()`

### Usage

```
add.sampling(eventTable, time, time.units = NA)
```

### Arguments

<code>eventTable</code>	An eventTable object. When accessed from object it would be <code>eventTable\$</code>
<code>time</code>	a vector of time values (in <code>time.units</code> ).
<code>time.units</code>	an optional string specifying the time units. Defaults to the units specified when the EventTable was initialized.

### Value

`eventTable` with updated sampling. (Note the event table will be updated even if you don't reassign the `eventTable`)

### Author(s)

Matthew L Fidler, Wenping Wang

### References

Wang W, Hallow K, James D (2015). "A Tutorial on RxODE: Simulating Differential Equation Pharmacometric Models in R." *CPT: Pharmacometrics & Systems Pharmacology*, 5(1), 3-10. ISSN 2163-8306, <URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4728294/>>.

### See Also

[eventTable](#), [add.sampling](#), [add.dosing](#), [et](#), [etRep](#), [etRbind](#), [RxODE](#)

### Examples

```
library(RxODE)
library(units)

## Model from RxODE tutorial
mod1 <-RxODE{{
```

```

KA=2.94E-01;
CL=1.86E+01;
V2=4.02E+01;
Q=1.05E+01;
V3=2.97E+02;
Kin=1;
Kout=1;
EC50=200;
C2 = centr/V2;
C3 = peri/V3;
d/dt(depot) =-KA*depot;
d/dt(centr) = KA*depot - CL*C2 - Q*C2 + Q*C3;
d/dt(peri) = Q*C2 - Q*C3;
d/dt(eff) = Kin - Kout*(1-C2/(EC50+C2))*eff;
});

## These are making the more complex regimens of the RxODE tutorial

## bid for 5 days
bid <- et(timeUnits="hr") %>%
  et(amt=10000,ii=12,until=set_units(5, "days"))

## qd for 5 days
qd <- et(timeUnits="hr") %>%
  et(amt=20000,ii=24,until=set_units(5, "days"))

## bid for 5 days followed by qd for 5 days

et <- seq(bid,qd) %>% et(seq(0,11*24,length.out=100));

bidQd <- rxSolve(mod1, et)

plot(bidQd, C2)

## Now Infusion for 5 days followed by oral for 5 days

## note you can dose to a named compartment instead of using the compartment number
infusion <- et(timeUnits = "hr") %>%
  et(amt=10000, rate=5000, ii=24, until=set_units(5, "days"), cmt="centr")

qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(5, "days"), cmt="depot")

et <- seq(infusion,qd)

infusionQd <- rxSolve(mod1, et)

plot(infusionQd, C2)

## 2wk-on, 1wk-off

qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")

```

```

et <- seq(qd, set_units(1,"weeks"), qd) %>%
  add.sampling(set_units(seq(0, 5.5,by=0.005),weeks))

wkOnOff <- rxSolve(mod1, et)

plot(wkOnOff, C2)

## You can also repeat the cycle easily with the rep function

qd <-et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")

et <- etRep(qd, times=4, wait=set_units(1,"weeks")) %>%
  add.sampling(set_units(seq(0, 12.5,by=0.005),weeks))

repCycle4 <- rxSolve(mod1, et)

plot(repCycle4, C2)

```

---

as.data.table.rxEt      *Convert an event table to a data.table*

---

### Description

Convert an event table to a data.table

### Usage

```
as.data.table.rxEt(x, keep.rownames = FALSE, ...)
```

### Arguments

x	An R object.
keep.rownames	Default is FALSE. If TRUE, adds the input object's names as a separate column named "rn". keep.rownames = "id" names the column "id" instead.
...	Additional arguments to be passed to or from other methods.

### Value

data.table of event table

---

as.et	<i>Coerce object to data.frame</i>
-------	------------------------------------

---

**Description**

Coerce object to data.frame

**Usage**

```
as.et(x, ...)
```

```
## Default S3 method:
```

```
as.et(x, ...)
```

**Arguments**

x                    Object to coerce to et.

...                  Other parameters

**Value**

An event table

---

as_tibble.rxEt	<i>Convert to tbl</i>
----------------	-----------------------

---

**Description**

Convert to tbl

**Usage**

```
as_tibble.rxEt(x, ...)
```

**Arguments**

x                    RxODE event table

...                  Other arguments to as\_tibble

**Value**

tibble of event table

---

coef.RxODE	<i>Return the RxODE coefficients</i>
------------	--------------------------------------

---

**Description**

This returns the parameters , state variables

**Usage**

```
## S3 method for class 'RxODE'  
coef(object, ...)
```

**Arguments**

object	is an RxODE object
...	ignored arguments

**Value**

a rxCoef object with the following

- params is a list of strings for parameters for the RxODE object
- state is a list of strings for the names of each state in the RxODE object.
- ini is the model specified default values for the parameters.
- RxODE is the referring RxODE object

**Author(s)**

Matthew L.Fidler

---

cvPost	<i>Sample a covariance Matrix from the Posterior Inverse Wishart distribution.</i>
--------	--

---

**Description**

Note this Inverse wishart rescaled to match the original scale of the covariance matrix.

**Usage**

```

cvPost(
  nu,
  omega,
  n = 1L,
  omegaIsChol = FALSE,
  returnChol = FALSE,
  type = c("invWishart", "lkj", "separation"),
  diagXformType = c("log", "identity", "variance", "nlmixrSqrt", "nlmixrLog",
    "nlmixrIdentity")
)

```

**Arguments**

nu	Degrees of Freedom (Number of Observations) for covariance matrix simulation.
omega	Either the estimate of covariance matrix or the estimated standard deviations in matrix form each row forming the standard deviation simulated values
n	Number of Matrices to sample. By default this is 1. This is only useful when omega is a matrix. Otherwise it is determined by the number of rows in the input omega matrix of standard deviations
omegaIsChol	is an indicator of if the omega matrix is in the Cholesky decomposition. This is only used when codetype="invWishart"
returnChol	Return the Cholesky decomposition of the covariance matrix sample. This is only used when codetype="invWishart"
type	The type of covariance posterior that is being simulated. This can be: <ul style="list-style-type: none"> <li>• <code>invWishart</code> The posterior is an inverse wishart; This allows for correlations between parameters to be modeled. All the uncertainty in the parameter is captured in the degrees of freedom parameter.</li> <li>• <code>lkj</code> The posterior separates the standard deviation estimates (modeled outside and provided in the omega argument) and the correlation estimates. The correlation estimate is simulated with the <code>rLKJ1()</code>. This simulation uses the relationship <math>\eta = (\nu - 1) / 2</math>. This is relationship based on the proof of the relationship between the restricted LKJ-distribution and inverse wishart distribution (XXXXXX). Once the correlation posterior is calculated, the estimated standard deviations are then combined with the simulated correlation matrix to create the covariance matrix.</li> <li>• <code>separation</code> Like the <code>lkj</code> option, this separates out the estimation of the correlation and standard deviation. Instead of using the LKJ distribution to simulate the correlation, it simulates the inverse wishart of the identity matrix and converts the result to a correlation matrix. This correlation matrix is then used with the standard deviation to calculate the simulated covariance matrix.</li> </ul>
diagXformType	Diagonal transformation type. These could be: <ul style="list-style-type: none"> <li>• <code>log</code> The standard deviations are log transformed, so the actual standard deviations are <math>\exp(\omega)</math></li> </ul>



- `identity` The standard deviations are not transformed. The standard deviations are not transformed; They should be positive.
- `variance` The variances are specified in the omega matrix; They are transformed into standard deviations.
- `nlmixrSqrt` These standard deviations come from an nlmixr omega matrix where  $\text{diag}(\text{chol}(\text{inv}(\text{omega}))) = x^2$
- `nlmixrLog` These standard deviations come from a nlmixr omega matrix where  $\text{diag}(\text{chol}(\text{solve}(\text{omega}))) = \exp(x)$
- `nlmixrIdentity` These standard deviations come from a nlmixr omega matrix where  $\text{diag}(\text{chol}(\text{solve}(\text{omega}))) = x$

The nlmixr transformations only make sense when there is no off-diagonal correlations modeled.

## Details

If your covariance matrix is a 1x1 matrix, this uses an scaled inverse chi-squared which is equivalent to the Inverse Wishart distribution in the uni-directional case.

In general, the separation strategy is preferred for diagonal matrices. If the dimension of the matrix is below 10, lkj is numerically faster than separation method. However, the lkj method has densities too close to zero (XXXX) when the dimension is above 10. In that case, though computationally more expensive separation method performs better.

For matrices with modeled covariances, the easiest method to use is the inverse Wishart which allows the simulation of correlation matrices (XXXX). This method is more well suited for well behaved matrices, that is the variance components are not too low or too high. When modeling non-linear mixed effects modeling matrices with too high or low variances are considered sub-optimal in describing a system. With these rules in mind, it is reasonable to use the inverse Wishart.

## Value

a matrix (n=1) or a list of matrices (n > 1)

## Author(s)

Matthew L.Fidler & Wenping Wang

## References

Alvarez I, Niemi J and Simpson M. (2014) *Bayesian Inference for a Covariance Matrix*. Conference on Applied Statistics in Agriculture. <https://newprairiepress.org/cgi/viewcontent.cgi?article=1004&context=agstatconference>

Wangl Z, Wu Y, and Chu H. (2018) *On Equivalence of the LKJ distribution and the restricted Wishart distribution*. arXiv:1809.04746

## Examples

```
## Sample a single covariance.
draw1 <- cvPost(3, matrix(c(1,.3,.3,1),2,2))
```

```

## Sample 3 covariances
set.seed(42)
draw3 <- cvPost(3, matrix(c(1,.3,.3,1),2,2), n=3)

## Sample 3 covariances, but return the cholesky decomposition
set.seed(42)
draw3c <- cvPost(3, matrix(c(1,.3,.3,1),2,2), n=3, returnChol=TRUE)

## Sample 3 covariances with lognormal standard deviations via LKJ
## correlation sample
cvPost(3,sapply(1:3,function(...){rnorm(10)}), type="lkj")

## or return cholesky decomposition
cvPost(3,sapply(1:3,function(...){rnorm(10)}), type="lkj",
      returnChol=TRUE)

## Sample 3 covariances with lognormal standard deviations via separation
## strategy using inverse Wishart correlation sample
cvPost(3,sapply(1:3,function(...){rnorm(10)}), type="separation")

## or returning the cholesky decomposition
cvPost(3,sapply(1:3,function(...){rnorm(10)}), type="separation",
      returnChol=TRUE)

```

---

et

*Event Table Function*


---

## Description

Event Table Function

## Usage

```

et(x, ..., envir = parent.frame())

## S3 method for class 'RxODE'
et(x, ..., envir = parent.frame())

## S3 method for class 'rxSolve'
et(x, ..., envir = parent.frame())

## S3 method for class 'rxParams'
et(x, ..., envir = parent.frame())

## Default S3 method:
et(
  x,

```

```

    ...,
    time,
    amt,
    evid,
    cmt,
    ii,
    addl,
    ss,
    rate,
    dur,
    until,
    id,
    amountUnits,
    timeUnits,
    addSampling,
    envir = parent.frame(),
    by = NULL,
    length.out = NULL
  )

```

### Arguments

x	This is the first argument supplied to the event table. This is named to allow et to be used in a pipe-line with arbitrary objects.
...	Times or event tables. They can also be one of the named arguments below.
envir	the <a href="#">environment</a> in which expr is to be evaluated. May also be NULL, a list, a data frame, a pairlist or an integer as specified to <a href="#">sys.call</a> .
time	Time is the time of the dose or the sampling times. This can also be unspecified and is determined by the object type (list or numeric/integer).
amt	Amount of the dose. If specified, this assumes a dosing record, instead of a sampling record.
evid	Event ID; This can be:

Numeric Value	Description
0	An observation. This can also be specified as evid=obs
1	A dose observation. This can also be specified as evid=dose
2	A non-dose event. This can also be specified as evid=other
3	A reset event. This can also be specified as evid=reset.
4	Dose and reset event. This can also be specified as evid=doseReset or evid=resetDose

Note a reset event resets all the compartment values to zero and turns off all infusions.

cmt	Compartment name or number. If a number, this is an integer starting at 1. Negative compartments turn off a compartment. If the compartment is a name, the compartment name is changed to the correct state/compartment number before running the simulation. For a compartment named "-cmt" the compartment is
-----	---

turned off.

Can also specify `cmt` as `dosing.to`,  
`dose.to`, `doseTo`, `dosingTo`, and  
`state`.

ii	When specifying a dose, this is the inter-dose interval for ss, addl and until options (described below).
addl	The number of additional doses at a inter-dose interval after one dose.
ss	Steady state flag; It can be one of:

Value	Description
0	This dose is not a steady state dose
1	This dose is a steady state dose with the between/inter-dose interval of ii
2	Superposition steady state

When ss=2 the steady state dose that uses the super-position principle to allow more complex steady states, like 10 mg in the morning and 20 mg at night, or dosing at 8 am 12 pm and 8 pm instead of every 12 hours. Since it uses the super positioning principle, it only makes sense when you know the kinetics are linear.

All other values of SS are currently invalid.

rate	When positive, this is the rate of infusion. Otherwise:
------	---

Value	Description
0	No infusion is on this record
-1	Modeled rate (in RxODE:rate(cmt) =); Can be et(rate=model).
-2	Modeled duration (in RxODE: dur(cmt) =); Can be et(dur=model) or et(rate=dur).

When a modeled bioavailability is applied to positive rates ( $rate > 0$ ), the duration of infusion is changed. This is because the data specify the rate and amount, the only think that modeled bioavailability can affect is duration.

If instead you want the modeled bioavailability to increase the rate of infusion instead of the duration of infusion, specify the dur instead or model the duration with rate=2.

dur	Duration of infusion. When amt and dur are specified the rate is calculated from the two data items. When dur is specified instead of rate, the bioavailability changes will increase rate instead of duration.
until	This is the time until the dosing should end. It can be an easier way to figure out how many additional doses are needed over your sampling period.
id	A integer vector of IDs to add or remove from the event table. If the event table is identical for each ID, then you may expand it to include all the IDs in this vector. All the negative IDs in this vector will be removed.
amountUnits	The units for the dosing records (amt)
timeUnits	The units for the time records (time)

addSampling	This is a boolean indicating if a sampling time should be added at the same time as a dosing time. By default this is FALSE.
by	When there are no observations in the event table, this is the amount to increment for the observations between from and to.
length.out	The number of observations to create if there isn't any observations in the event table. By default this is 200.

**Value**

A new event table

**Author(s)**

Matthew L Fidler, Wenping Wang

**References**

Wang W, Hallow K, James D (2015). "A Tutorial on RxODE: Simulating Differential Equation Pharmacometric Models in R." *CPT: Pharmacometrics & Systems Pharmacology*, 5(1), 3-10. ISSN 2163-8306, <URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4728294/>>.

**See Also**

[eventTable](#), [add.sampling](#), [add.dosing](#), [et](#), [etRep](#), [etRbind](#), [RxODE](#)

**Examples**

```
library(RxODE)
library(units)

## Model from RxODE tutorial
mod1 <-RxODE({
  KA=2.94E-01;
  CL=1.86E+01;
  V2=4.02E+01;
  Q=1.05E+01;
  V3=2.97E+02;
  Kin=1;
  Kout=1;
  EC50=200;
  C2 = centr/V2;
  C3 = peri/V3;
  d/dt(depot) =-KA*depot;
  d/dt(centr) = KA*depot - CL*C2 - Q*C2 + Q*C3;
  d/dt(peri) = Q*C2 - Q*C3;
  d/dt(eff) = Kin - Kout*(1-C2/(EC50+C2))*eff;
});
```

```

## These are making the more complex regimens of the RxODE tutorial

## bid for 5 days
bid <- et(timeUnits="hr") %>%
  et(amt=10000,ii=12,until=set_units(5, "days"))

## qd for 5 days
qd <- et(timeUnits="hr") %>%
  et(amt=20000,ii=24,until=set_units(5, "days"))

## bid for 5 days followed by qd for 5 days

et <- seq(bid,qd) %>% et(seq(0,11*24,length.out=100));

bidQd <- rxSolve(mod1, et)

plot(bidQd, C2)

## Now Infusion for 5 days followed by oral for 5 days

## note you can dose to a named compartment instead of using the compartment number
infusion <- et(timeUnits = "hr") %>%
  et(amt=10000, rate=5000, ii=24, until=set_units(5, "days"), cmt="centr")

qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(5, "days"), cmt="depot")

et <- seq(infusion,qd)

infusionQd <- rxSolve(mod1, et)

plot(infusionQd, C2)

## 2wk-on, 1wk-off

qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")

et <- seq(qd, set_units(1,"weeks"), qd) %>%
  add.sampling(set_units(seq(0, 5.5,by=0.005),weeks))

wkOnOff <- rxSolve(mod1, et)

plot(wkOnOff, C2)

## You can also repeat the cycle easily with the rep function

qd <-et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")

et <- etRep(qd, times=4, wait=set_units(1,"weeks")) %>%
  add.sampling(set_units(seq(0, 12.5,by=0.005),weeks))

repCycle4 <- rxSolve(mod1, et)

```

```
plot(repCycle4, C2)
```

---

etExpand	<i>Expand additional doses</i>
----------	--------------------------------

---

**Description**

Expand additional doses

**Usage**

```
etExpand(et)
```

**Arguments**

et                    Event table to expand additional doses for.

**Value**

New event table with addl doses expanded

**Author(s)**

Matthew Fidler

**Examples**

```
ev <- et(amt=3,ii=24,until=240);
print(ev)
etExpand(ev) # expands event table, but doesn't modify it

print(ev)

ev$expand() ## Expands the current event table and saves it in ev
```

---

 etRbind

*Combining event tables*


---

**Description**

Combining event tables

**Usage**

```
etRbind(
  ...,
  samples = c("use", "clear"),
  waitII = c("smart", "+ii"),
  id = c("merge", "unique")
)

## S3 method for class 'rxEt'
rbind(..., deparse.level = 1)
```

**Arguments**

...	The event tables and optionally time between event tables, called waiting times in this help document.
samples	How to handle samples when repeating an event table. The options are: <ul style="list-style-type: none"> <li>• "clear" Clear sampling records before combining the datasets</li> <li>• "use" Use the sampling records when combining the datasets</li> </ul>
waitII	This determines how waiting times between events are handled. The options are: <ul style="list-style-type: none"> <li>• "smart" This "smart" handling of waiting times is the default option. In this case, if the waiting time is above the last observed inter-dose interval in the first combined event table, then the actual time between doses is given by the wait time. If it is smaller than the last observed inter-dose interval, the time between event tables is given by the inter-dose interval + the waiting time between event tables.</li> <li>• "+ii" In this case, the wait time is added to the inter-dose interval no matter the length of the wait time or inter-dose interval</li> </ul>
id	This is how rbind will handle IDs. There are two different types of options: <ul style="list-style-type: none"> <li>• merge with id="merge", the IDs are merged together, overlapping IDs would be merged into a single event table.</li> <li>• unique with id="unique", the IDs will be renumbered so that the IDs in all the event tables are not overlapping.</li> </ul>
deparse.level	The deparse.level of a traditional rbind is ignored.



**Value**

An event table

**Author(s)**

Matthew L Fidler

Matthew L Fidler, Wenping Wang

**References**

Wang W, Hallow K, James D (2015). "A Tutorial on RxODE: Simulating Differential Equation Pharmacometric Models in R." *CPT: Pharmacometrics & Systems Pharmacology*, 5(1), 3-10. ISSN 2163-8306, <URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4728294/>>.

**See Also**

[eventTable](#), [add.sampling](#), [add.dosing](#), [et](#), [etRep](#), [etRbind](#), [RxODE](#)

**Examples**

```
library(RxODE)
library(units)

## Model from RxODE tutorial
mod1 <- RxODE({
  KA=2.94E-01;
  CL=1.86E+01;
  V2=4.02E+01;
  Q=1.05E+01;
  V3=2.97E+02;
  Kin=1;
  Kout=1;
  EC50=200;
  C2 = centr/V2;
  C3 = peri/V3;
  d/dt(depot) = -KA*depot;
  d/dt(centr) = KA*depot - CL*C2 - Q*C2 + Q*C3;
  d/dt(peri) = Q*C2 - Q*C3;
  d/dt(eff) = Kin - Kout*(1-C2/(EC50+C2))*eff;
});

## These are making the more complex regimens of the RxODE tutorial

## bid for 5 days
bid <- et(timeUnits="hr") %>%
  et(amt=10000,ii=12,until=set_units(5, "days"))

## qd for 5 days
```

```
qd <- et(timeUnits="hr") %>%
  et(amt=20000,ii=24,until=set_units(5, "days"))

## bid for 5 days followed by qd for 5 days

et <- seq(bid,qd) %>% et(seq(0,11*24,length.out=100));

bidQd <- rxSolve(mod1, et)

plot(bidQd, C2)

## Now Infusion for 5 days followed by oral for 5 days

## note you can dose to a named compartment instead of using the compartment number
infusion <- et(timeUnits = "hr") %>%
  et(amt=10000, rate=5000, ii=24, until=set_units(5, "days"), cmt="centr")

qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(5, "days"), cmt="depot")

et <- seq(infusion,qd)

infusionQd <- rxSolve(mod1, et)

plot(infusionQd, C2)

## 2wk-on, 1wk-off

qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")

et <- seq(qd, set_units(1,"weeks"), qd) %>%
  add.sampling(set_units(seq(0, 5.5,by=0.005),weeks))

wkOnOff <- rxSolve(mod1, et)

plot(wkOnOff, C2)

## You can also repeat the cycle easily with the rep function

qd <-et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")

et <- etRep(qd, times=4, wait=set_units(1,"weeks")) %>%
  add.sampling(set_units(seq(0, 12.5,by=0.005),weeks))

repCycle4 <- rxSolve(mod1, et)

plot(repCycle4, C2)
```

---

etRep

*Repeat an RxODE event table*


---

**Description**

Repeat an RxODE event table

**Usage**

```
etRep(
  x,
  times = 1,
  length.out = NA,
  each = NA,
  n = NULL,
  wait = 0,
  id = integer(0),
  samples = c("clear", "use"),
  waitII = c("smart", "+ii"),
  ii = 24
)

## S3 method for class 'rxEt'
rep(x, ...)
```

**Arguments**

x	An RxODE event table
times	Number of times to repeat the event table
length.out	Invalid with RxODE event tables, will throw an error if used.
each	Invalid with RxODE event tables, will throw an error if used.
n	The number of times to repeat the event table. Overrides times.
wait	Waiting time between each repeated event table. By default there is no waiting, or wait=0
id	A integer vector of IDs to add or remove from the event table. If the event table is identical for each ID, then you may expand it to include all the IDs in this vector. All the negative IDs in this vector will be removed.
samples	How to handle samples when repeating an event table. The options are: <ul style="list-style-type: none"> <li>• "clear" Clear sampling records before combining the datasets</li> <li>• "use" Use the sampling records when combining the datasets</li> </ul>
waitII	This determines how waiting times between events are handled. The options are:

- "smart" This "smart" handling of waiting times is the default option. In this case, if the waiting time is above the last observed inter-dose interval in the first combined event table, then the actual time between doses is given by the wait time. If it is smaller than the last observed inter-dose interval, the time between event tables is given by the inter-dose interval + the waiting time between event tables.
  - "+i i" In this case, the wait time is added to the inter-dose interval no matter the length of the wait time or inter-dose interval
- ii            When specifying a dose, this is the inter-dose interval for `ss`, `add1` and `until` options (described below).
- ...           Times or event tables. They can also be one of the named arguments below.

**Value**

An event table

**Author(s)**

Matthew L Fidler, Wenping Wang

**References**

Wang W, Hallow K, James D (2015). "A Tutorial on RxODE: Simulating Differential Equation Pharmacometric Models in R." *CPT: Pharmacometrics & Systems Pharmacology*, 5(1), 3-10. ISSN 2163-8306, <URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4728294/>>.

**See Also**

[eventTable](#), [add.sampling](#), [add.dosing](#), [et](#), [etRep](#), [etRbind](#), [RxODE](#)

**Examples**

```
library(RxODE)
library(units)

## Model from RxODE tutorial
mod1 <-RxODE({
  KA=2.94E-01;
  CL=1.86E+01;
  V2=4.02E+01;
  Q=1.05E+01;
  V3=2.97E+02;
  Kin=1;
  Kout=1;
  EC50=200;
  C2 = centr/V2;
  C3 = peri/V3;
```

```

    d/dt(depot) = -KA*depot;
    d/dt(centr) = KA*depot - CL*C2 - Q*C2 + Q*C3;
    d/dt(peri) = Q*C2 - Q*C3;
    d/dt(eff) = Kin - Kout*(1-C2/(EC50+C2))*eff;
  });

## These are making the more complex regimens of the RxODE tutorial

## bid for 5 days
bid <- et(timeUnits="hr") %>%
  et(amt=10000,ii=12,until=set_units(5, "days"))

## qd for 5 days
qd <- et(timeUnits="hr") %>%
  et(amt=20000,ii=24,until=set_units(5, "days"))

## bid for 5 days followed by qd for 5 days

et <- seq(bid,qd) %>% et(seq(0,11*24,length.out=100));

bidQd <- rxSolve(mod1, et)

plot(bidQd, C2)

## Now Infusion for 5 days followed by oral for 5 days

## note you can dose to a named compartment instead of using the compartment number
infusion <- et(timeUnits = "hr") %>%
  et(amt=10000, rate=5000, ii=24, until=set_units(5, "days"), cmt="centr")

qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(5, "days"), cmt="depot")

et <- seq(infusion,qd)

infusionQd <- rxSolve(mod1, et)

plot(infusionQd, C2)

## 2wk-on, 1wk-off

qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")

et <- seq(qd, set_units(1,"weeks"), qd) %>%
  add.sampling(set_units(seq(0, 5.5,by=0.005),weeks))

wkOnOff <- rxSolve(mod1, et)

plot(wkOnOff, C2)

## You can also repeat the cycle easily with the rep function

```

```

qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")

et <- etRep(qd, times=4, wait=set_units(1,"weeks")) %>%
  add.sampling(set_units(seq(0, 12.5,by=0.005),weeks))

repCycle4 <- rxSolve(mod1, et)

plot(repCycle4, C2)

```

---

etSeq

*Sequence of event tables*


---

### Description

This combines a sequence of event tables.

### Usage

```

etSeq(..., samples = c("clear", "use"), waitII = c("smart", "+ii"), ii = 24)

## S3 method for class 'rxEt'
seq(...)

```

### Arguments

...	The event tables and optionally time between event tables, called waiting times in this help document.
samples	How to handle samples when repeating an event table. The options are: <ul style="list-style-type: none"> <li>• "clear" Clear sampling records before combining the datasets</li> <li>• "use" Use the sampling records when combining the datasets</li> </ul>
waitII	This determines how waiting times between events are handled. The options are: <ul style="list-style-type: none"> <li>• "smart" This "smart" handling of waiting times is the default option. In this case, if the waiting time is above the last observed inter-dose interval in the first combined event table, then the actual time between doses is given by the wait time. If it is smaller than the last observed inter-dose interval, the time between event tables is given by the inter-dose interval + the waiting time between event tables.</li> <li>• "+ii" In this case, the wait time is added to the inter-dose interval no matter the length of the wait time or inter-dose interval</li> </ul>
ii	If there was no inter-dose intervals found in the event table, assume that the interdose interval is given by this ii value. By default this is 24.

**Details**

This sequences all the event tables in added in the argument list . . . . By default when combining the event tables the offset is at least by the last inter-dose interval in the prior event table (or *i*). If you separate any of the event tables by a number, the event tables will be separated at least the wait time defined by that number or the last inter-dose interval.

**Value**

An event table

**Author(s)**

Matthew L Fidler, Wenping Wang

**References**

Wang W, Hallow K, James D (2015). "A Tutorial on RxODE: Simulating Differential Equation Pharmacometric Models in R." *CPT: Pharmacometrics & Systems Pharmacology*, 5(1), 3-10. ISSN 2163-8306, <URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4728294/>>.

**See Also**

[eventTable](#), [add.sampling](#), [add.dosing](#), [et](#), [etRep](#), [etRbind](#), [RxODE](#)

**Examples**

```
library(RxODE)
library(units)

## Model from RxODE tutorial
mod1 <-RxODE({
  KA=2.94E-01;
  CL=1.86E+01;
  V2=4.02E+01;
  Q=1.05E+01;
  V3=2.97E+02;
  Kin=1;
  Kout=1;
  EC50=200;
  C2 = centr/V2;
  C3 = peri/V3;
  d/dt(depot) =-KA*depot;
  d/dt(centr) = KA*depot - CL*C2 - Q*C2 + Q*C3;
  d/dt(peri) = Q*C2 - Q*C3;
  d/dt(eff) = Kin - Kout*(1-C2/(EC50+C2))*eff;
});

## These are making the more complex regimens of the RxODE tutorial
```

```

## bid for 5 days
bid <- et(timeUnits="hr") %>%
  et(amt=10000,ii=12,until=set_units(5, "days"))

## qd for 5 days
qd <- et(timeUnits="hr") %>%
  et(amt=20000,ii=24,until=set_units(5, "days"))

## bid for 5 days followed by qd for 5 days

et <- seq(bid,qd) %>% et(seq(0,11*24,length.out=100));

bidQd <- rxSolve(mod1, et)

plot(bidQd, C2)

## Now Infusion for 5 days followed by oral for 5 days

## note you can dose to a named compartment instead of using the compartment number
infusion <- et(timeUnits = "hr") %>%
  et(amt=10000, rate=5000, ii=24, until=set_units(5, "days"), cmt="centr")

qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(5, "days"), cmt="depot")

et <- seq(infusion,qd)

infusionQd <- rxSolve(mod1, et)

plot(infusionQd, C2)

## 2wk-on, 1wk-off

qd <- et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")

et <- seq(qd, set_units(1,"weeks"), qd) %>%
  add.sampling(set_units(seq(0, 5.5,by=0.005),weeks))

wkOnOff <- rxSolve(mod1, et)

plot(wkOnOff, C2)

## You can also repeat the cycle easily with the rep function

qd <-et(timeUnits = "hr") %>% et(amt=10000, ii=24, until=set_units(2, "weeks"), cmt="depot")

et <- etRep(qd, times=4, wait=set_units(1,"weeks")) %>%
  add.sampling(set_units(seq(0, 12.5,by=0.005),weeks))

repCycle4 <- rxSolve(mod1, et)

```



```
plot(repCycle4, C2)
```

---

eventTable	<i>Create an event table object</i>
------------	-------------------------------------

---

### Description

Initializes an object of class 'EventTable' with methods for adding and querying dosing and observation records

### Usage

```
eventTable(amount.units = NA, time.units = NA)
```

### Arguments

`amount.units` string denoting the amount dosing units, e.g., "mg", "ug". Default to NA to denote unspecified units. It could also be a solved RxODE object. In that case, `eventTable(obj)` returns the eventTable that was used to solve the RxODE object.

`time.units` string denoting the time units, e.g., "hours", "days". Default to "hours".  
An eventTable is an object that consists of a data.frame storing ordered time-stamped events of an (unspecified) PK/PD dynamic system, units (strings) for dosing and time records, plus a list of functions to add and extract event records. Currently, events can be of two types: dosing events that represent inputs to the system and sampling time events that represent observations of the system with 'amount.units' and 'time.units', respectively.

### Value

A modified data.frame with the following accessible functions:

- `get.EventTable()` returns the current event table
- `add.dosing()` adds dosing records to the event table.
- `get.dosing()` returns a data.frame of dosing records.
- `clear.dosing()` clears or deletes all dosing from event table
- `add.sampling()` adds sampling time observation records to the event table.
- `get.sampling()` returns a data.frame of sampled observation records.
- `clear.sampling()` removes all sampling from event table.
- `get.obs.rec()` returns a logical vector indicating whether each event record represents an observation or not.
- `get.nobs()` returns the number of observation (not dosing) records.
- `get.units()` returns a two-element character vector with the dosing and time units, respectively

- `copy()` makes a copy of the current event table. To create a copy of an event table object use `qd2 <- qd$copy()`
- `expand()` Expands the event table for multi-subject solving. This is done by `qd$expand(400)` for a 400 subject data expansion

### Author(s)

Matthew Fidler, Melissa Hallow and Wenping Wang

### See Also

[et\(\)](#), [RxODE\(\)](#)

### Examples

```
# create dosing and observation (sampling) events
# QD 50mg dosing, 5 days followed by 25mg 5 days
#
qd <- eventTable(amount.units = "mg", time.units = "days")
#
qd$add.dosing(dose=50, nbr.doses=5, dosing.interval = 1, do.sampling=FALSE)
#
# sample the system's drug amounts hourly the first day, then every 12 hours
# for the next 4 days
qd$add.sampling(seq(from = 0, to = 1, by = 1/24))
qd$add.sampling(seq(from = 1, to = 5, by = 12/24))
#
#print(qd$get.dosing())      # table of dosing records
print(qd$get.nobs())      # number of observation (not dosing) records
#
# BID dosing, 5 days
bid <- eventTable("mg", "days") # only dosing
bid$add.dosing(dose=10000, nbr.doses=2*5,
              dosing.interval = 12, do.sampling=FALSE)
#
# Use the copy() method to create a copy (clone) of an existing
# event table (simple assignments just create a new reference to
# the same event table object (closure)).
#
bid.ext <- bid$copy()      # three-day extension for a 2nd cohort
bid.ext$add.dosing(dose = 5000, nbr.doses = 2*3,
                 start.time = 120, dosing.interval = 12, do.sampling = FALSE)

# You can also use the Piping operator to create a table

qd2 <- eventTable(amount.units="mg", time.units="days") %>%
  add.dosing(dose=50, nbr.doses=5, dosing.interval=1, do.sampling=FALSE) %>%
  add.sampling(seq(from=0, to=1, by=1 / 24)) %>%
  add.sampling(seq(from=1, to=5, by=12 / 24))
#print(qd2$get.dosing())    # table of dosing records
print(qd2$get.nobs())     # number of observation (not dosing) records
```

```
# Note that piping with %>% will update the original table.

qd3 <- qd2 %>% add.sampling(seq(from=5, to=10, by=6 / 24))
print(qd2$get.nobs())
print(qd3$get.nobs())
```

---

forderForceBase	<i>Force using base order for RxODE radix sorting</i>
-----------------	---

---

### Description

Force using base order for RxODE radix sorting

### Usage

```
forderForceBase(forceBase = FALSE)
```

### Arguments

forceBase      boolean indicating if RxODE should use R's `order()` for radix sorting instead of `data.table`'s parallel radix sorting.

### Value

NILL; called for side effects

### Examples

```
forderForceBase(TRUE) # Use base `order` for RxODE sorts
forderForceBase(FALSE) # Use `data.table` for RxODE sorts
```

---

gammap	<i>Gammap: normalized lower incomplete gamma function</i>
--------	---

---

### Description

This is the `gamma_p` from the boost library

### Usage

```
gammap(a, z)
```

**Arguments**

- a                    The numeric 'a' parameter in the normalized lower incomplete gamma  
z                    The numeric 'z' parameter in the normalized lower incomplete gamma

**Details**

The gamma p function is given by:

$\text{gammap} = \text{lowergamma}(a, z)/\text{gamma}(a)$

**Value**

gammap results

**Author(s)**

Matthew L. Fidler

**Examples**

```
gammap(1,3)
gammap(1:3,3)
gammap(1,1:3)
```

---

gammapDer

*gammapDer: derivative of gammap*

---

**Description**

This is the gamma\_p\_derivative from the boost library

**Usage**

```
gammapDer(a, z)
```

**Arguments**

- a                    The numeric 'a' parameter in the upper incomplete gamma  
z                    The numeric 'z' parameter in the upper incomplete gamma

**Value**

lowergamma results

**Author(s)**

Matthew L. Fidler

**Examples**

```
gammapDer(1:3,3)
```

```
gammapDer(1,1:3)
```

---

gammapInv	<i>gammapInv and gammapInva: Inverses of normalized gammap function</i>
-----------	---

---

**Description**

gammapInv and gammapInva: Inverses of normalized gammap function

**Usage**

```
gammapInv(a, p)
```

```
gammapInva(x, p)
```

**Arguments**

a	The numeric 'a' parameter in the upper incomplete gamma
p	The numeric 'p' parameter in the upper incomplete gamma
x	The numeric 'x' parameter in the upper incomplete gamma

**Details**

With the equation:

$$p = \text{gammap}(a, x)$$

The 'gammapInv' function returns a value 'x' that satisfies the equation above

The 'gammapInva' function returns a value 'q' that satisfies the equation above

NOTE: gammapInva is slow

**Value**

inverse gammap results

**Author(s)**

Matthew L. Fidler

**Examples**

```
gammapInv(1:3,0.5)
gammapInv(1,1:3/3.1)
gammapInv(1:3,1:3/3.1)
gammapInva(1:3,1:3/3.1)
```

---

gammaq

*Gammaq: normalized upper incomplete gamma function*

---

**Description**

This is the gamma\_q from the boost library

**Usage**

```
gammaq(a, z)
```

**Arguments**

a                   The numeric 'a' parameter in the normalized upper incomplete gamma  
z                   The numeric 'z' parameter in the normalized upper incomplete gamma

**Details**

The gamma q function is given by:

$$\text{gammaq} = \text{uppergamma}(a, z) / \text{gamma}(a)$$
**Value**

gammaq results

**Author(s)**

Matthew L. Fidler

**Examples**

```
gammaq(1,3)
gammaq(1:3,3)
gammaq(1,1:3)
```

---

gammaqInv	<i>gammaqInv and gammaqInva: Inverses of normalized gammaq function</i>
-----------	---

---

**Description**

gammaqInv and gammaqInva: Inverses of normalized gammaq function

**Usage**

```
gammaqInv(a, q)
```

```
gammaqInva(x, q)
```

**Arguments**

a	The numeric 'a' parameter in the upper incomplete gamma
q	The numeric 'q' parameter in the upper incomplete gamma
x	The numeric 'x' parameter in the upper incomplete gamma

**Details**

With the equation:

$$q = \text{gammaq}(a, x)$$

The 'gammaqInv' function returns a value 'x' that satisfies the equation above

The 'gammaqInva' function returns a value 'a' that satisfies the equation above

NOTE: gammaqInva is slow

**Value**

inverse gammaq results

**Author(s)**

Matthew L. Fidler

**Examples**

```
gammaqInv(1:3, 0.5)
```

```
gammaqInv(1, 1:3/3)
```

```
gammaqInv(1:3, 1:3/3.1)
```

```
gammaqInva(1:3, 1:3/3.1)
```

---

genShinyApp.template    *Generate an example (template) of a dosing regimen shiny app*

---

## Description

Create a complete shiny application for exploring dosing regimens given a (hardcoded) PK/PD model.

## Usage

```
genShinyApp.template(
  appDir = "shinyExample",
  verbose = TRUE,
  ODE.config = list(ode = "model", params = c(KA = 0.294), inits = c(eff = 1), method =
    "lsoda", atol = 1e-08, rtol = 1e-06)
)

write.template.server(appDir)

write.template.ui(appDir, statevars)
```

## Arguments

appDir	a string with a directory where to store the shiny app, by default is "shinyExample". The directory appDir will be created if it does not exist.
verbose	logical specifying whether to write messages as the shiny app is generated. Defaults to TRUE.
ODE.config	model name compiled and list of parameters sent to <code>rxSolve()</code> .
statevars	List of statevars passed to to the <code>write.template.ui()</code> function. This usually isn't called directly.

A PK/PD model is defined using `RxODE()`, and a set of parameters and initial values are defined. Then the appropriate R scripts for the shiny's user interface `ui.R` and the server logic `server.R` are created in the directory `appDir`.

The function evaluates the following PK/PD model by default:

$$\begin{aligned} C2 &= \text{centr}/V2; \\ C3 &= \text{peri}/V3; \\ d/dt(\text{depot}) &= -KA*\text{depot}; \\ d/dt(\text{centr}) &= KA*\text{depot} - CL*C2 - Q*C2 + Q*C3; \\ d/dt(\text{peri}) &= Q*C2 - Q*C3; \\ d/dt(\text{eff}) &= \text{Kin} - \text{Kout}*(1-C2/(EC50+C2))*\text{eff}; \end{aligned}$$

This can be changed by the `ODE.config` parameter.

To launch the shiny app, simply issue the `runApp(appDir)` R command.

## Value

None, these functions are used for their side effects.



**Note**

These functions create a simple, but working example of a dosing regimen simulation web application. Users may want to modify the code to experiment creating shiny applications for their specific RxODE models.

**See Also**

`RxODE()`, `eventTable()`, and the package **shiny** (<https://shiny.rstudio.com>).

**Examples**

```
# create the shiny app example (template)
genShinyApp.template(appDir = "myapp")
# run the shiny app
library(shiny)
# runApp("myapp") # Won't launch in environments without browsers
unlink("myapp", recursive=TRUE, force=TRUE) # remove myapp
```

---

getRxThreads

*Get/Set the number of threads that RxODE uses*


---

**Description**

Get/Set the number of threads that RxODE uses

**Usage**

```
getRxThreads(verbose = FALSE)

setRxThreads(threads = NULL, percent = NULL, throttle = NULL)

rxCores(verbose = FALSE)
```

**Arguments**

verbose	Display the value of relevant OpenMP settings
threads	NULL (default) rereads environment variables. 0 means to use all logical CPUs available. Otherwise a number $\geq 1$
percent	If provided it should be a number between 2 and 100; the percentage of logical CPUs to use. By default on startup, 50 percent.
throttle	2 (default) means that, roughly speaking, a single thread will be used when number subjects solved for is $\leq 2$ , 2 threads when the number of all points is $\leq 4$ , etc. The throttle is to speed up small data tasks (especially when repeated many times) by not incurring the overhead of managing multiple threads.

The throttle will also suppress sorting which ID will be solved first when there are  $(n_{\text{subject solved}}) * \text{throttle} \leq n_{\text{threads}}$ . In RxODE this sorting occurs to minimize the time for waiting for another thread to finish. If the last item solved is has a long solving time, all the other solving have to wait for that last costly solving to occur. If the items which are likely to take more time are solved first, this wait is less likely to have an impact on the overall solving time.

In RxODE the IDs are sorted by the individual number of solving points (largest first). It also has a C interface that allows these IDs to be resorted by total time spent solving the equation. This allows packages like nlmixr to sort by solving time if needed.

Overall the the number of threads is throttled (restricted) for small tasks and sorting for IDs are suppressed.

## Value

number of threads that RxODE uses

---

guide\_none

*Empty Guide*

---

## Description

This empty guide draws nothing; It is included in RxODE for compatibility with ggplot 3.2

## Usage

```
guide_none(title = waiver(), position = waiver())
```

## Arguments

title	A character string or expression indicating a title of guide. If NULL, the title is not shown. By default ( <code>waiver()</code> ), the name of the scale object or the name specified in <code>labs()</code> is used for the title.
position	Where this guide should be drawn: one of top, bottom, left, or right.

## Value

nothing, simply included to be compatible with ggplot 3.2

---

invWR1d	<i>One correlation sample from the Inverse Wishart distribution</i>
---------	---

---

**Description**

This correlation is constructed by transformation of the Inverse Wishart random covariate to a correlation.

**Usage**

```
invWR1d(d, nu, omegaIsChol = FALSE)
```

**Arguments**

d	The dimension of the correlation matrix
nu	Degrees of freedom of the Wishart distribution
omegaIsChol	is an indicator of if the omega matrix is in the Cholesky decomposition. This is only used when codetype="invWishart"

**Value**

One correlation sample from the inverse wishart

**Author(s)**

Matthew Fidler

---

is.rxEt	<i>Check to see if this is an rxEt object.</i>
---------	--

---

**Description**

Check to see if this is an rxEt object.

**Usage**

```
is.rxEt(x)
```

**Arguments**

x	object to check to see if it is rxEt If this is an rxEt object that has expired strip all rxEt information.
---	--

**Value**

Boolean indicating if this is a RxODE event table

**Author(s)**

Matthew L.Fidler

---

is.rxSolve	<i>Check to see if this is an rxSolve object.</i>
------------	---

---

**Description**

Check to see if this is an rxSolve object.

**Usage**

is.rxSolve(x)

**Arguments**

x	object to check to see if it is rxSolve If this is an rxSolve object that has expired strip all rxSolve information.
---	---

**Value**

boolean indicating if this is a rxSolve object

**Author(s)**

Matthew L.Fidler

---

logit	<i>logit and inverse logit (expit) functions</i>
-------	--

---

**Description**

logit and inverse logit (expit) functions

**Usage**

logit(x, low = 0, high = 1)

expit(alpha, low = 0, high = 1)

logitNormInfo(mean = 0, sd = 1, low = 0, high = 1, abs.tol = 1e-06, ...)

probitNormInfo(mean = 0, sd = 1, low = 0, high = 1, abs.tol = 1e-06, ...)

**Arguments**

x	Input value(s) in range [low,high] to translate -Inf to Inf
low	Lowest value in the range
high	Highest value in the range
alpha	Infinite value(s) to translate to range of [low, high]
mean	logit-scale mean
sd	logit-scale standard deviation
abs.tol	absolute accuracy requested.
...	other parameters passed to integrate()

**Details**

logit is given by:

$$\text{logit}(p) = -\log(1/p-1)$$

where:

$$p = x - \text{low} / \text{high} - \text{low}$$

expit is given by:

$$\text{expit}(p, \text{low}, \text{high}) = (\text{high} - \text{low}) / (1 + \exp(-\alpha)) + \text{low}$$

The `logitNormInfo()` gives the mean, variance and coefficient of variability on the untransformed scale.

**Value**

values from logit and expit

**Examples**

```
logit(0.25)
```

```
expit(-1.09)
```

```
logitNormInfo(logit(0.25),sd=0.1)
```

```
logitNormInfo(logit(1,0,10), sd=1, low=0, high=10)
```

lowergamma

*lowergamma: upper incomplete gamma function*

---

**Description**

This is the `tgamma_lower` from the boost library

**Usage**

```
lowergamma(a, z)
```

**Arguments**

a	The numeric 'a' parameter in the upper incomplete gamma
z	The numeric 'z' parameter in the upper incomplete gamma

**Details**

The lowergamma function is given by:

$$\text{lowergamma}(a, z) = \int_0^z t^{a-1} \cdot e^{-t} dt$$

**Value**

lowergamma results

**Author(s)**

Matthew L. Fidler

**Examples**

```
lowergamma(1, 3)
```

```
lowergamma(1:3, 3)
```

```
lowergamma(1, 1:3)
```

---

phi *Cumulative distribution of standard normal*

---

**Description**

Cumulative distribution of standard normal

**Usage**

phi(q)

**Arguments**

q                    vector of quantiles.

**Value**

cumulative distribution of standard normal distribution

**Author(s)**

Matthew Fidler

**Examples**

```
# phi is equivalent to pnorm(x)
phi(3)

# See
pnorm(3)

# This is provided for NONMEM-like compatibility in RxODE models
```

---

print.RxODE *Print information about the RxODE object.*

---

**Description**

This prints the model name and its status for being able to be solved

**Usage**

```
## S3 method for class 'RxODE'
print(x, ...)
```

**Arguments**

x                    An rxode object  
...                   Ignored parameters

**Value**

original object

**Author(s)**

Matthew L.Fidler

---

probit                                    *probit and inverse probit functions*

---

**Description**

probit and inverse probit functions

**Usage**

```
probit(x, low = 0, high = 1)
probitInv(x, low = 0, high = 1)
```

**Arguments**

x                    Input value(s) in range [low,high] to translate -Inf to Inf  
low                   Lowest value in the range  
high                   Highest value in the range

**Value**

values from probit, probitInv and probitNormInfo

**Examples**

```
probit(0.25)
probitInv(-0.674)
probitNormInfo(probit(0.25),sd=0.1)
probitNormInfo(probit(1,0,10), sd=1, low=0, high=10)
```



---

rinvchisq	<i>Scaled Inverse Chi Squared distribution</i>
-----------	--

---

**Description**

Scaled Inverse Chi Squared distribution

**Usage**

```
rinvchisq(n = 1L, nu = 1, scale = 1)
```

**Arguments**

n	Number of random samples
nu	degrees of freedom of inverse chi square
scale	Scale of inverse chi squared distribution (default is 1).

**Value**

a vector of inverse chi squared deviates.

**Examples**

```
rinvchisq(3, 4, 1) ## Scale = 1, degrees of freedom = 4
rinvchisq(2, 4, 2) ## Scale = 2, degrees of freedom = 4
```

---

rLKJ1	<i>One correlation sample from the LKJ distribution</i>
-------	---

---

**Description**

One correlation sample from the LKJ distribution

**Usage**

```
rLKJ1(d, eta = 1, cholesky = FALSE)
```

**Arguments**

d	The dimension of the correlation matrix
eta	The scaling parameter of the LKJ distribution. Must be $> 1$ . Also related to the degrees of freedom $\nu$ . $\eta = (\nu - 1)/2$ .
cholesky	boolean; If TRUE return the cholesky decomposition.

**Value**

A correlation sample from the LKJ distribution

**Author(s)**

Matthew Fidler (translated to RcppArmadillo) and Emma Schwager

---

rxAddReturn                      *Add a return statment to a function.*

---

**Description**

Add a return statment to a function.

**Usage**

```
rxAddReturn(fn, ret = TRUE)
```

**Arguments**

fn	Function to deparse
ret	boolean stating if a return statement will be added.

**Value**

Function with parens removed and add a return statment.

**Author(s)**

Matthew L. Fidler

---

rxAllowUnload                      *Allow unloading of dlls*

---

**Description**

Allow unloading of dlls

**Usage**

```
rxAllowUnload(allow)
```

**Arguments**

allow	boolean indicating if garbage collection will unload of RxODE dlls.
-------	---

**Value**

Boolean allow; called for side effects

**Author(s)**

Matthew Fidler

**Examples**

```
# Garbage collection will not unload un-used RxODE dlls
rxAllowUnload(FALSE);

# Garbage collection will unload unused RxODE dlls
rxAllowUnload(TRUE);
```

---

*rxAssignPtr*

*Assign pointer based on model variables*

---

**Description**

Assign pointer based on model variables

**Usage**

```
rxAssignPtr(object = NULL)
```

**Arguments**

object            RxODE family of objects

**Value**

nothing, called for side effects

---

`rxbeta`*Simulate beta variable from threefry generator*

---

### Description

Care should be taken with this method not to encounter the birthday problem, described <https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/>. Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

### Usage

```
rxbeta(shape1, shape2, n = 1L, ncores = 1L)
```

### Arguments

<code>shape1</code>	non-negative parameters of the Beta distribution.
<code>shape2</code>	non-negative parameters of the Beta distribution.
<code>n</code>	number of observations. If <code>length(n) &gt; 1</code> , the length is taken to be the number required.
<code>ncores</code>	Number of cores for the simulation rxnorm simulates using the threefry sitmo generator; rxnormV uses the vander-corput generator

### Details

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to run all simulations in the RxODE environment once (therefore one seed or series of seeds for the whole simulation) or to pre-generate all random variables used for the simulation.

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

### Value

beta random deviates

### Examples

```
## Use threefry engine
```

```

rxbeta(0.5, 0.5, n=10) # with rxbeta you have to explicitly state n
rxbeta(5, 1, n=10, ncores=2) # You can parallelize the simulation using openMP

rxbeta(1, 3)

## This example uses `rxbeta` directly in the model

rx <- RxODE({
  a = rxbeta(2, 2)
})

et <- et(1,id=1:2)

s <- rxSolve(rx,et)

```

---

 rxbinom

*Simulate Binomial variable from threefry generator*


---

### Description

Care should be taken with this method not to encounter the birthday problem, described <https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/>. Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

### Usage

```
rxbinom(size, prob, n = 1L, ncores = 1L)
```

### Arguments

size	number of trials (zero or more).
prob	probability of success on each trial.
n	number of observations. If <code>length(n) &gt; 1</code> , the length is taken to be the number required.
ncores	Number of cores for the simulation rxnorm simulates using the threefry sitmo generator; rxnormV uses the vander-corput generator

### Details

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to run all simulations in the RxODE environment once (therefore one seed or series of seeds for the whole simulation) or to pre-generate all random variables used for the simulation.

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

## Value

binomial random deviates

## Examples

```
## Use threefry engine

rxbinom(10, 0.9, n=10) # with rxbinom you have to explicitly state n
rxbinom(3, 0.5, n=10, ncores=2) # You can parallelize the simulation using openMP

rxbinom(4,0.7)

## This example uses `rxbinom` directly in the model

rx <- RxODE({
  a = rxbinom(1,0.5)
})

et <- et(1,id=1:2)

s <- rxSolve(rx,et)
```

---

rxcauchy

*Simulate Cauchy variable from threefry generator*

---

## Description

Care should be taken with this method not to encounter the birthday problem, described <https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/>. Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

## Usage

```
rxcauchy(location = 0, scale = 1, n = 1L, ncores = 1L)
```

**Arguments**

location	location and scale parameters.
scale	location and scale parameters.
n	number of observations. If <code>length(n) &gt; 1</code> , the length is taken to be the number required.
ncores	Number of cores for the simulation rxnorm simulates using the threefry sitmo generator; rxnormV uses the vander-corput generator

**Details**

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to run all simulations in the RxODE environment once (therefore one seed or series of seeds for the whole simulation) or to pre-generate all random variables used for the simulation.

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

**Value**

Cauchy random deviates

**Examples**

```
## Use threefry engine

rxcauchy(0, 1, n=10) # with rxcauchy you have to explicitly state n
rxcauchy(0.5, n=10, ncores=2) # You can parallelize the simulation using openMP

rxcauchy(3)

## This example uses `rxcauchy` directly in the model

rx <- RxODE({
  a = rxcauchy(2)
})

et <- et(1,id=1:2)

s <- rxSolve(rx,et)
```

---

rxChain	<i>rxChain Chain or add item to solved system of equations</i>
---------	--

---

**Description**

Add item to solved system of equations

**Usage**

```
rxChain(obj1, obj2)
```

```
## S3 method for class 'solveRxD11'  
obj1 + obj2
```

**Arguments**

obj1	Solved object.
obj2	New object to be added/piped/chained to solved object.

**Value**

When newObject is an event table, return a new solved object with the new event table.

**Author(s)**

Matthew L. Fidler

---

rxchisq	<i>Simulate chi-squared variable from threefry generator</i>
---------	--

---

**Description**

Care should be taken with this method not to encounter the birthday problem, described <https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/>. Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

**Usage**

```
rxchisq(df, n = 1L, ncores = 1L)
```



**Arguments**

df	degrees of freedom (non-negative, but can be non-integer).
n	number of observations. If <code>length(n) &gt; 1</code> , the length is taken to be the number required.
ncores	Number of cores for the simulation rxnorm simulates using the threefry sitmo generator; rxnormV uses the vander-corput generator

**Details**

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to run all simulations in the RxODE environment once (therefore one seed or series of seeds for the whole simulation) or to pre-generate all random variables used for the simulation.

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

**Value**

chi squared random deviates

**Examples**

```
## Use threefry engine

rxchisq(0.5, n=10) # with rxchisq you have to explicitly state n
rxchisq(5, n=10, ncores=2) # You can parallelize the simulation using openMP

rxchisq(1)

## This example uses `rxchisq` directly in the model

rx <- RxODE({
  a = rxchisq(2)
})

et <- et(1,id=1:2)

s <- rxSolve(rx,et)
```

---

rxClean	<i>Cleanup anonymous DLLs by unloading them</i>
---------	---

---

**Description**

This cleans up any RxODE loaded DLLs

**Usage**

```
rxClean(wd)
```

**Arguments**

wd	What directory should be cleaned; (DEPRECATED), this no longer does anything. This unloads all RxODE anonymous dlls.
----	---

**Value**

TRUE if successful

**Author(s)**

Matthew L. Fidler

---

rxCompile	<i>Compile a model if needed</i>
-----------	----------------------------------

---

**Description**

This is the compilation workhorse creating the RxODE model DLL files.

**Usage**

```
rxCompile(
  model,
  dir,
  prefix,
  force = FALSE,
  modName = NULL,
  package = NULL,
  ...
)

## S3 method for class 'rxModelVars'
rxCompile(
```

```

    model,
    dir = NULL,
    prefix = NULL,
    force = FALSE,
    modName = NULL,
    package = NULL,
    ...
)

## S3 method for class 'character'
rxCompile(
  model,
  dir = NULL,
  prefix = NULL,
  force = FALSE,
  modName = NULL,
  package = NULL,
  ...
)

## S3 method for class 'rxDll'
rxCompile(model, ...)

## S3 method for class 'RxODE'
rxCompile(model, ...)

```

## Arguments

model	<p>This is the ODE model specification. It can be:</p> <ul style="list-style-type: none"> <li>• a string containing the set of ordinary differential equations (ODE) and other expressions defining the changes in the dynamic system.</li> <li>• a file name where the ODE system equation is contained</li> </ul> <p>An ODE expression enclosed in <code>\{\}</code> (see also the filename argument). For details, see the sections “Details” and RxODE Syntax below.</p>
dir	<p>This is the model directory where the C file will be stored for compiling.</p> <p>If unspecified, the C code is stored in a temporary directory, then the model is compiled and moved to the current directory. Afterwards the C code is removed.</p> <p>If specified, the C code is stored in the specified directory and then compiled in that directory. The C code is not removed after the DLL is created in the same directory. This can be useful to debug the c-code outputs.</p>
prefix	<p>is a string indicating the prefix to use in the C based functions. If missing, it is calculated based on file name, or md5 of parsed model.</p>

force	is a boolean stating if the (re)compile should be forced if RxODE detects that the models are the same as already generated.
modName	a string to be used as the model name. This string is used for naming various aspects of the computations, including generating C symbol names, dynamic libraries, etc. Therefore, it is necessary that modName consists of simple ASCII alphanumeric characters starting with a letter.
package	Package name for pre-compiled binaries.
...	Other arguments sent to the <code>rxTrans()</code> function.

**Value**

An `rxDll` object that has the following components

- `dllDLL` path
- `modelmodel` specification
- `.cA` function to call C code in the correct context from the DLL using the `.C()` function.
- `.callA` function to call C code in the correct context from the DLL using the `.Call()` function.
- `argsA` list of the arguments used to create the `rxDll` object.

**Author(s)**

Matthew L.Fidler

**See Also**

[RxODE\(\)](#)

---

<code>rxCreateCache</code>	<i>This will create the cache directory for RxODE to save between sessions</i>
----------------------------	--

---

**Description**

When run, if the `R_user_dir` for RxODE's cache isn't present, create the cache

**Usage**

```
rxCreateCache()
```

**Value**

nothing

**Author(s)**

Matthew Fidler

---

rxD *Add to RxODE's derivative tables*

---

**Description**

Add to RxODE's derivative tables

**Usage**

```
rxD(name, derivatives)
```

**Arguments**

name	Function Name
derivatives	A list of functions. Each function takes the same number of arguments as the original function. The first function will construct the derivative with respect to the first argument; The second function will construct the derivative with respect to the second argument, and so on.

**Value**

nothing

**Author(s)**

Matthew Fidler

**Examples**

```
## Add an arbitrary list of derivative functions
## In this case the fun(x,y) is assumed to be 0.5*x^2+0.5*y^2

rxD("fun", list(function(x,y){return(x)},
                 function(x,y){return(y)}
                ))
```

---

rxDelete *Delete the DLL for the model*

---

**Description**

This function deletes the DLL, but doesn't delete the model information in the object.

**Usage**

```
rxDelete(obj)
```

**Arguments**

obj                    RxODE family of objects

**Value**

A boolean stating if the operation was successful.

**Author(s)**

Matthew L.Fidler

---

rxDerived	<i>Calculate derived parameters for the 1-, 2-, and 3- compartment linear models.</i>
-----------	---

---

**Description**

This calculates the derived parameters based on what is provided in a data frame or arguments

**Usage**

```
rxDerived(..., verbose = FALSE, digits = 0)
```

**Arguments**

...                    The input can be:

- A data frame with PK parameters in it; This should ideally be a data frame with one pk parameter per row since it will output a data frame with one PK parameter per row.
- PK parameters as either a vector or a scalar

verbose                boolean that when TRUE provides a message about the detected pk parameters and the detected compartmental model. By default this is FALSE.

digits                 represents the number of significant digits for the output; If the number is zero or below (default), do not round.

**Value**

Return a data.frame of derived PK parameters for a 1-, 2-, or 3-compartment linear model given provided clearances and volumes based on the inferred model type.

The model parameters that will be provided in the data frame are:

- vc: Central Volume (for 1-, 2- and 3- compartment models)
- kel: First-order elimination rate (for 1-, 2-, and 3-compartment models)
- k12: First-order rate of transfer from central to first peripheral compartment; (for 2- and 3-compartment models)

- k21: First-order rate of transfer from first peripheral to central compartment, (for 2- and 3-compartment models)
- k13: First-order rate of transfer from central to second peripheral compartment; (3-compartment model)
- k31: First-order rate of transfer from second peripheral to central compartment (3-compartment model)
- vp: Peripheral Volume (for 2- and 3- compartment models)
- vp2: Peripheral Volume for 3rd compartment (3- compartment model)
- vss: Volume of distribution at steady state; (1-, 2-, and 3-compartment models)
- t12alpha:  $t_{1/2,\alpha}$ ; (1-, 2-, and 3-compartment models)
- t12beta:  $t_{1/2,\beta}$ ; (2- and 3-compartment models)
- t12gamma:  $t_{1/2,\gamma}$ ; (3-compartment model)
- alpha:  $\alpha$ ; (1-, 2-, and 3-compartment models)
- beta:  $\beta$ ; (2- and 3-compartment models)
- gamma:  $\beta$ ; (3-compartment model)
- A: true A; (1-, 2-, and 3-compartment models)
- B: true B; (2- and 3-compartment models)
- C: true C; (3-compartment model)
- fracA: fractional A; (1-, 2-, and 3-compartment models)
- fracB: fractional B; (2- and 3-compartment models)
- fracC: fractional C; (3-compartment model)

### Author(s)

Matthew Fidler and documentation from Justin Wilkins, <justin.wilkins@occams.com>

### References

Shafer S. L. CONVERT.XLS

Rowland M, Tozer TN. Clinical Pharmacokinetics and Pharmacodynamics: Concepts and Applications (4th). Clipping Williams & Wilkins, Philadelphia, 2010.

### Examples

```
## Note that RxODE parses the names to figure out the best PK parameter
params <- rxDerived(c1=29.4, v=23.4, Vp=114, vp2=4614, q=270, q2=73)

## That is why this gives the same results as the value before
params <- rxDerived(CL=29.4, V1=23.4, V2=114, V3=4614, Q2=270, Q3=73)

## You may also use micro-constants alpha/beta etc.
```

```
params <- rxDerived(k12=0.1, k21=0.2, k13=0.3, k31=0.4, kel=10, v=10)

## or you can mix vectors and scalars

params <- rxDerived(CL=29.4, V=1:3)

## If you want, you can round to a number of significant digits
## with the `digits` argument:

params <- rxDerived(CL=29.4, V=1:3, digits=2)
```

---

rxDfdy

*Jacobian and parameter derivatives*

---

### Description

Return Jacobian and parameter derivatives

### Usage

```
rxDfdy(obj)
```

### Arguments

obj                    RxODE family of objects

### Value

A list of the jacobian parameters defined in this RxODE object.

### Author(s)

Matthew L. Fidler

---

rxEvid

*EVID formatting for tibble and other places.*

---

### Description

This is to make an EVID more readable by non pharmacometricians. It displays what each means and allows it to be displayed in a tibble.



**Usage**

```
rxEvid(x)

as.rxEvid(x)

## S3 method for class 'rxEvid'
c(x, ...)

## S3 method for class 'rxEvid'
x[...]

## S3 method for class 'rxEvid'
as.character(x, ...)

## S3 method for class 'rxEvid'
x[[...]]

type_sum.rxEvid(x)

pillar_shaft.rxEvid(x, ...)

## S3 method for class 'rxRateDur'
c(x, ...)

## S3 method for class 'rxEvid'
format(x, ...)

## S3 method for class 'rxRateDur'
format(x, ...)

## S3 method for class 'rxEvid'
print(x, ...)
```

**Arguments**

x	Item to be converted to a RxODE EVID specification.
...	Other parameters

**Value**

rxEvid specification

**Examples**

```
rxEvid(1:7)
```

rxexp

*Simulate exponential variable from threefry generator***Description**

Care should be taken with this method not to encounter the birthday problem, described <https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/>. Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

**Usage**

```
rxexp(rate, n = 1L, ncores = 1L)
```

**Arguments**

rate	vector of rates.
n	number of observations. If length(n) > 1, the length is taken to be the number required.
ncores	Number of cores for the simulation rxnorm simulates using the threefry sitmo generator; rxnormV uses the vander-corput generator

**Details**

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to run all simulations in the RxODE environment once (therefore one seed or series of seeds for the whole simulation) or to pre-generate all random variables used for the simulation.

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

**Value**

exponential random deviates

**Examples**

```
## Use threefry engine

rxexp(0.5, n=10) # with rxexp you have to explicitly state n
rxexp(5, n=10, ncores=2) # You can parallelize the simulation using openMP
```

```

rxexp(1)

## This example uses `rxexp` directly in the model

rx <- RxODE({
  a = rxexp(2)
})

et <- et(1,id=1:2)

s <- rxSolve(rx,et)

```

---

rx

*Simulate F variable from threefry generator*


---

### Description

Care should be taken with this method not to encounter the birthday problem, described <https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/>. Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

### Usage

```
rx(df1, df2, n = 1L, ncores = 1L)
```

### Arguments

df1	degrees of freedom. Inf is allowed.
df2	degrees of freedom. Inf is allowed.
n	number of observations. If <code>length(n) &gt; 1</code> , the length is taken to be the number required.
ncores	Number of cores for the simulation rxnorm simulates using the threefry sitmo generator; rxnormV uses the vander-corput generator

### Details

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to run all simulations in the RxODE environment once (therefore one seed or series of seeds for the whole simulation) or to pre-generate all random variables used for the simulation.

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

### Value

f random deviates

### Examples

```
## Use threefry engine

rxf(0.5, 0.5, n=10) # with rxf you have to explicitly state n
rxf(5, 1, n=10, ncores=2) # You can parallelize the simulation using openMP

rxf(1, 3)

## This example uses `rxf` directly in the model

rx <- RxODE({
  a = rxf(2, 2)
})

et <- et(1,id=1:2)

s <- rxSolve(rx,et)
```

---

 rxFun

---

*Add user function to RxODE*


---

### Description

This adds a user function to RxODE that can be called. If needed, these functions can be differentiated by numerical differences or by adding the derivatives to RxODE's internal derivative table with `rxD()`

### Usage

```
rxFun(name, args, cCode)
```

```
rxRmFun(name)
```

**Arguments**

name	This gives the name of the user function
args	This gives the arguments of the user function
cCode	This is the C-code for the new function

**Value**

nothing

**Author(s)**

Matthew L. Fidler

**Examples**

```
## Right now RxODE is not aware of the function f
## Therefore it cannot translate it to symengine or
## Compile a model with it.

try(RxODE("a=fun(a,b,c)"))

## Note for this approach to work, it cannot interfere with C
## function names or reserved RxODE special terms. Therefore
## f(x) would not work since f is an alias for bioavailability.

fun <- "
double fun(double a, double b, double c) {
  return a*a+b*a+c;
}
" ## C-code for function

rxFun("fun",c("a","b","c"), fun) ## Added function

## Now RxODE knows how to translate this function to symengine

rxToSE("fun(a,b,c)")

## And will take a central difference when calculating derivatives

rxFromSE("Derivative(fun(a,b,c),a)")

## Of course, you could specify the derivative table manually
rxD("fun", list(function(a,b,c){
  paste0("2*",a,"+",b);
}),
  function(a,b,c){
    return(a)
  },
)
```

```

    function(a,b,c){
      return("0.0")
    } ))

rxFromSE("Derivative(fun(a,b,c),a)")

# You can also remove the functions by `rxRmFun`

rxRmFun("fun")

```

---

 rxgamma

*Simulate gamma variable from threefry generator*


---

### Description

Care should be taken with this method not to encounter the birthday problem, described <https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/>. Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

### Usage

```
rxgamma(shape, rate = 1/scale, scale = 1, n = 1L, ncores = 1L)
```

### Arguments

shape	shape and scale parameters. Must be positive, scale strictly.
rate	an alternative way to specify the scale.
scale	shape and scale parameters. Must be positive, scale strictly.
n	number of observations. If <code>length(n) &gt; 1</code> , the length is taken to be the number required.
ncores	Number of cores for the simulation rxnorm simulates using the threefry sitmo generator; rxnormV uses the vander-corput generator

### Details

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to run all simulations in the RxODE environment once (therefore one seed or series of seeds for the whole simulation) or to pre-generate all random variables used for the simulation.

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

**Value**

gamma random deviates

**Examples**

```
## Use threefry engine

rxgamma(0.5, n=10) # with rxgamma you have to explicitly state n
rxgamma(5, n=10, ncores=2) # You can parallelize the simulation using openMP

rxgamma(1)

## This example uses `rxbeta` directly in the model

rx <- RxODE({
  a = rxgamma(2)
})

et <- et(1,id=1:2)

s <- rxSolve(rx,et)
```

---

rxGenSaem

*Generate pred-only SAEM RxODE model*

---

**Description**

Generate pred-only SAEM RxODE model

**Usage**

```
rxGenSaem(
  obj,
  predfn,
  pkpars = NULL,
  sum.prod = FALSE,
  optExpression = TRUE,
  loadSymengine = TRUE
)
```

**Arguments**

obj	RxODE model (text or actual model)
predfn	prediction function
pkpars	PKpars function
sum.prod	Change addition and deletion to sum() and prod() to stabilize round off errors
optExpression	Boolean for optimizing expression to minimize more expensive calls
loadSymengine	Boolean indicating if the model should be loaded into symengine. This cause all the ODEs to be collapsed into one expression that is eventually optimized if optExpression is TRUE.

**Value**

RxODE text

**Author(s)**

Matthew Fidler

---

rxgeom

*Simulate geometric variable from threefry generator*

---

**Description**

Care should be taken with this method not to encounter the birthday problem, described <https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/>. Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

**Usage**

```
rxgeom(prob, n = 1L, ncores = 1L)
```

**Arguments**

prob	probability of success in each trial. $0 < \text{prob} \leq 1$ .
n	number of observations. If <code>length(n) &gt; 1</code> , the length is taken to be the number required.
ncores	Number of cores for the simulation rxnorm simulates using the threefry sitmo generator; rxnormV uses the vander-corput generator



## Details

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to run all simulations in the RxODE environment once (therefore one seed or series of seeds for the whole simulation) or to pre-generate all random variables used for the simulation.

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

## Value

geometric random deviates

## Examples

```
## Use threefry engine

rxgeom(0.5, n=10) # with rxgeom you have to explicitly state n
rxgeom(0.25, n=10, ncores=2) # You can parallelize the simulation using openMP

rxgeom(0.75)

## This example uses `rxgeom` directly in the model

rx <- RxODE({
  a = rxgeom(0.24)
})

et <- et(1,id=1:2)

s <- rxSolve(rx,et)
```

---

rxGetLin

*Get the linear compartment model true function*

---

## Description

Get the linear compartment model true function

**Usage**

```
rxGetLin(
  model,
  linCmtSens = c("linCmtA", "linCmtB", "linCmtC"),
  verbose = FALSE
)
```

**Arguments**

model	This is the ODE model specification. It can be: <ul style="list-style-type: none"> <li>• a string containing the set of ordinary differential equations (ODE) and other expressions defining the changes in the dynamic system.</li> <li>• a file name where the ODE system equation is contained</li> </ul> An ODE expression enclosed in $\{\}$ (see also the filename argument). For details, see the sections “Details” and RxODE Syntax below.
linCmtSens	The method to calculate the linCmt() solutions
verbose	When TRUE be verbose with the linear compartmental model

**Value**

model with linCmt() replaced with linCmtA()

**Author(s)**

Matthew Fidler

---

rxGetRxODE

*Get RxODE model from object*

---

**Description**

Get RxODE model from object

**Usage**

```
rxGetRxODE(obj)
```

**Arguments**

obj	RxODE family of objects
-----	-------------------------

**Value**

RxODE model

---

rxHtml	<i>Format rxSolve and related objects as html.</i>
--------	--

---

**Description**

Format rxSolve and related objects as html.

**Usage**

```
rxHtml(x, ...)
```

```
## S3 method for class 'rxSolve'
```

```
rxHtml(x, ...)
```

**Arguments**

x	RxODE object
...	Extra arguments sent to kable

**Value**

html code for rxSolve object

**Author(s)**

Matthew L. Fidler

---

rxIndLinState	<i>Set the preferred factoring by state</i>
---------------	---

---

**Description**

Set the preferred factoring by state

**Usage**

```
rxIndLinState(preferred = NULL)
```

**Arguments**

preferred	A list of each state's preferred factorization
-----------	--

**Value**

Nothing

**Author(s)**

Matthew Fidler

---

rxIndLinStrategy	<i>This sets the inductive linearization strategy for matrix building</i>
------------------	---

---

**Description**

When there is more than one state in a ODE that cannot be separated this specifies how it is incorporated into the matrix exponential.

**Usage**

```
rxIndLinStrategy(strategy = c("curState", "split"))
```

**Arguments**

strategy	The strategy for inductive linearization matrix building
----------	--

- `curState` Prefer parameterizing in terms of the current state, followed by the first state observed in the term.
- `split` Split the parameterization between all states in the term by dividing each by the number of states in the term and then adding a matrix term for each state.

**Value**

Nothing

**Author(s)**

Matthew L. Fidler

---

rxInv	<i>Invert matrix using RcppArmadillo.</i>
-------	---

---

**Description**

Invert matrix using RcppArmadillo.

**Usage**

```
rxInv(matrix)
```

**Arguments**

matrix	matrix to be inverted.
--------	------------------------

**Value**

inverse or pseudo inverse of matrix.

---

rxIsCurrent	<i>Checks if the RxODE object was built with the current build</i>
-------------	--

---

**Description**

Checks if the RxODE object was built with the current build

**Usage**

```
rxIsCurrent(obj)
```

**Arguments**

obj	RxODE family of objects
-----	-------------------------

**Value**

boolean indicating if this was built with current RxODE

---

rxLhs	<i>Left handed Variables</i>
-------	------------------------------

---

**Description**

This returns the model calculated variables

**Usage**

```
rxLhs(obj)
```

**Arguments**

obj	RxODE family of objects
-----	-------------------------

**Value**

a character vector listing the calculated parameters

**Author(s)**

Matthew L.Fidler

**See Also**

[RxODE](#)

---

rxLock	<i>Lock/unlocking of RxODE dll file</i>
--------	---

---

**Description**

Lock/unlocking of RxODE dll file

**Usage**

rxLock(obj)

rxUnlock(obj)

**Arguments**

obj                    A RxODE family of objects

**Value**

nothing; called for side effects

---

rxNorm	<i>Get the normalized model</i>
--------	---------------------------------

---

**Description**

This get the syntax preferred model for processing

**Usage**

rxNorm(obj, condition = NULL, removeInis, removeJac, removeSens)

**Arguments**

obj	RxODE family of objects
condition	Character string of a logical condition to use for subsetting the normalized model. When missing, and a condition is not set via rxCondition, return the whole code with all the conditional settings intact. When a condition is set with rxCondition, use that condition.
removeInis	A boolean indicating if parameter initialization will be removed from the model
removeJac	A boolean indicating if the Jacobians will be removed.
removeSens	A boolean indicating if the sensitivities will be removed.

**Value**

Normalized Normal syntax (no comments)

**Author(s)**

Matthew L. Fidler

---

rxnorm	<i>Simulate random normal variable from threefry/vandercorput generator</i>
--------	---

---

**Description**

Simulate random normal variable from threefry/vandercorput generator

**Usage**

```
rxnorm(mean = 0, sd = 1, n = 1L, ncores = 1L)
```

```
rxnormV(mean = 0, sd = 1, n = 1L, ncores = 1L)
```

**Arguments**

mean	vector of means.
sd	vector of standard deviations.
n	number of observations
ncores	Number of cores for the simulation rxnorm simulates using the threefry sitmo generator; rxnormV uses the vandercorput generator

**Value**

normal random number deviates

**Examples**

```
## Use threefry engine

rxnorm(n=10) # with rxnorm you have to explicitly state n
rxnorm(n=10,ncores=2) # You can parallelize the simulation using openMP

rxnorm(2,3) ## The first 2 arguments are the mean and standard deviation

## This example uses `rxnorm` directly in the model

rx <- RxODE({
  a = rxnorm()
})
```

```
et <- et(1,id=1:2)

s <- rxSolve(rx,et)

## Use vandercompt generator

rxnormV(n=10) # with rxnorm you have to explicitly state n
rxnormV(n=10,ncores=2) # You can parallelize the simulation using openMP

rxnormV(2,3) ## The first 2 arguments are the mean and standard deviation

## This example uses `rxnormV` directly in the model

rx <- RxODE({
  a = rxnormV()
})

et <- et(1,id=1:2)

s <- rxSolve(rx,et)
```

---

RxODE

*Create an ODE-based model specification*

---

## Description

Create a dynamic ODE-based model object suitably for translation into fast C code

## Usage

```
RxODE(
  model,
  modName = basename(wd),
  wd = getwd(),
  filename = NULL,
  extraC = NULL,
  debug = FALSE,
  calcJac = NULL,
  calcSens = NULL,
  collapseModel = FALSE,
  package = NULL,
  ...,
  linCmtSens = c("linCmtA", "linCmtB", "linCmtC"),
  indLin = FALSE,
  verbose = FALSE
)
```



**Arguments**

model	This is the ODE model specification. It can be: <ul style="list-style-type: none"> <li>• a string containing the set of ordinary differential equations (ODE) and other expressions defining the changes in the dynamic system.</li> <li>• a file name where the ODE system equation is contained</li> </ul> An ODE expression enclosed in <code>\{\}</code> (see also the <code>filename</code> argument). For details, see the sections “Details” and RxODE Syntax below.
modName	a string to be used as the model name. This string is used for naming various aspects of the computations, including generating C symbol names, dynamic libraries, etc. Therefore, it is necessary that <code>modName</code> consists of simple ASCII alphanumeric characters starting with a letter.
wd	character string with a working directory where to create a subdirectory according to <code>modName</code> . When specified, a subdirectory named after the “ <code>modName.d</code> ” will be created and populated with a C file, a dynamic loading library, plus various other working files. If missing, the files are created (and removed) in the temporary directory, and the RxODE DLL for the model is created in the current directory named <code>rx_????_platform</code> , for example <code>rx_129f8f97fb94a87ca49ca8dafe691e1e_i386.dll</code>
filename	A file name or connection object where the ODE-based model specification resides. Only one of <code>model</code> or <code>filename</code> may be specified.
extraC	Extra c code to include in the model. This can be useful to specify functions in the model. These C functions should usually take double precision arguments, and return double precision values.
debug	is a boolean indicating if the executable should be compiled with verbose debugging information turned on.
calcJac	boolean indicating if RxODE will calculate the Jacobain according to the specified ODEs.
calcSens	boolean indicating if RxODE will calculate the sensitivities according to the specified ODEs.
collapseModel	boolean indicating if RxODE will remove all LHS variables when calculating sensitivities.
package	Package name for pre-compiled binaries.
...	ignored arguments.
linCmtSens	The method to calculate the <code>linCmt()</code> solutions
indLin	Calculate inductive linearization matrices and compile with inductive linearization support.
verbose	When TRUE be verbose with the linear compartmental model

**Details**

The Rx in the name RxODE is meant to suggest the abbreviation *Rx* for a medical prescription, and thus to suggest the package emphasis on pharmacometrics modeling, including pharmacokinetics (PK), pharmacodynamics (PD), disease progression, drug-disease modeling, etc.

The ODE-based model specification may be coded inside a character string or in a text file, see Section *RxODE Syntax* below for coding details. An internal RxODE compilation manager object translates the ODE system into C, compiles it, and dynamically loads the object code into the current R session. The call to RxODE produces an object of class RxODE which consists of a list-like structure (environment) with various member functions (see Section *Value* below).

For evaluating RxODE models, two types of inputs may be provided: a required set of time points for querying the state of the ODE system and an optional set of doses (input amounts). These inputs are combined into a single *event table* object created with the function `eventTable()` or `et()`.

An RxODE model specification consists of one or more statements optionally terminated by semi-colons ; and optional comments (comments are delimited by # and an end-of-line).

A block of statements is a set of statements delimited by curly braces, { ... }.

Statements can be either assignments, conditional if/else if/else, while loops (can be exited by break), special statements, or printing statements (for debugging/testing)

Assignment statements can be:

- **simple** assignments, where the left hand is an identifier (i.e., variable)
- special **time-derivative** assignments, where the left hand specifies the change of the amount in the corresponding state variable (compartment) with respect to time e.g.,  $d/dt(\text{depot})$ :
- special **initial-condition** assignments where the left hand specifies the compartment of the initial condition being specified, e.g.  $\text{depot}(0) = 0$
- special model event changes including **bioavailability** ( $f(\text{depot})=1$ ), **lag time** ( $a\text{lag}(\text{depot})=0$ ), **modeled rate** ( $\text{rate}(\text{depot})=2$ ) and **modeled duration** ( $\text{dur}(\text{depot})=2$ ). An example of these model features and the event specification for the modeled infusions the RxODE data specification is found in [RxODE events vignette](#).
- special **change point syntax, or model times**. These model times are specified by  $\text{mtime}(\text{var})=\text{time}$
- special **Jacobian-derivative** assignments, where the left hand specifies the change in the compartment ode with respect to a variable. For example, if  $d/dt(y) = dy$ , then a Jacobian for this compartment can be specified as  $df(y)/dy(dy) = 1$ . There may be some advantage to obtaining the solution or specifying the Jacobian for very stiff ODE systems. However, for the few stiff systems we tried with LSODA, this actually slightly slowed down the solving.

Note that assignment can be done by =, <- or ~.

When assigning with the ~ operator, the **simple assignments** and **time-derivative** assignments will not be output.

Special statements can be:

- **Compartment declaration statements**, which can change the default dosing compartment and the assumed compartment number(s) as well as add extra compartment names at the end (useful for multiple-endpoint nlmixr models); These are specified by `cmt(compartmentName)`
- **Parameter declaration statements**, which can make sure the input parameters are in a certain order instead of ordering the parameters by the order they are parsed. This is useful for keeping the parameter order the same when using 2 different ODE models. These are specified by `param(par1, par2, ...)`

An example model is shown below:

```
# simple assignment
C2 = centr/V2;

# time-derivative assignment
d/dt(centr) = F*KA*depot - CL*C2 - Q*C2 + Q*C3;
```

Expressions in assignment and if statements can be numeric or logical, however, no character nor integer expressions are currently supported.

Numeric expressions can include the following numeric operators +, -, \*, /, ^ and those mathematical functions defined in the C or the R math libraries (e.g., fabs, exp, log, sin, abs).

You may also access the R's functions in the [R math libraries](#), like lgammafn for the log gamma function.

The RxODE syntax is case-sensitive, i.e., ABC is different than abc, Abc, ABC, etc.

### Identifiers:

Like R, Identifiers (variable names) may consist of one or more alphanumeric, underscore \_ or period . characters, but the first character cannot be a digit or underscore \_.

Identifiers in a model specification can refer to:

- State variables in the dynamic system (e.g., compartments in a pharmacokinetics model).
- Implied input variable, t (time), tlast (last time point), and podo (oral dose, in the undocumented case of absorption transit models).
- Special constants like pi or [R's predefined constants](#).
- Model parameters (e.g., ka rate of absorption, CL clearance, etc.)
- Others, as created by assignments as part of the model specification; these are referred as *LHS* (left-hand side) variable.

Currently, the RxODE modeling language only recognizes system state variables and “parameters”, thus, any values that need to be passed from R to the ODE model (e.g., age) should be either passed in the params argument of the integrator function rxSolve() or be in the supplied event data-set.

There are certain variable names that are in the RxODE event tables. To avoid confusion, the following event table-related items cannot be assigned, or used as a state but can be accessed in the RxODE code:

- cmt
- dvid
- addl
- ss
- rate
- id

However the following variables are cannot be used in a model specification:

- evid
- ii

Sometimes RxODE generates variables that are fed back to RxODE. Similarly, nlmixr generates some variables that are used in nlmixr estimation and simulation. These variables start with the either the rx or nlmixr prefixes. To avoid any problems, it is suggested to not use these variables starting with either the rx or nlmixr prefixes.

**Logical Operators:**

Logical operators support the standard R operators `==`, `!=`, `>=`, `<=`, `>` and `<`. Like R these can be in `if()` or `while()` statements, `ifelse()` expressions. Additionally they can be in a standard assignment. For instance, the following is valid:

```
cov1 = covm*(sexf == "female") + covm*(sexf != "female")
```

Notice that you can also use character expressions in comparisons. This convenience comes at a cost since character comparisons are slower than numeric expressions. Unlike R, `as.numeric` or `as.integer` for these logical statements is not only not needed, but will cause a syntax error if you try to use the function.

**Value**

An object (environment) of class `RxODE` (see Chambers and Temple Lang (2001)) consisting of the following list of strings and functions:

- \* ``model`` a character string holding the source model specification.
- \* ``get.modelVars`` a function that returns a list with 3 character vectors, ``params``, ``state``, and ``lhs`` of variable names used in the model specification. These will be output when the model is computed (i.e., the ODE solved by integration).

- \* ``solve``{this function solves (integrates) the ODE. This is done by passing the code to `[rxSolve()]`. This is as if you called ``rxSolve(RxODEobject, ...)``, but returns a matrix instead of a `rxSolve` object.

``params``: a numeric named vector with values for every parameter in the ODE system; the names must correspond to the parameter identifiers used in the ODE specification;

``events``: an ``eventTable`` object describing the input (e.g., doses) to the dynamic system and observation sampling time points (see `[eventTable()]`);

``inits``: a vector of initial values of the state variables (e.g., amounts in each compartment), and the order in this vector must be the same as the state variables (e.g., PK/PD compartments);

``stiff``: a logical (``TRUE`` by default) indicating whether the ODE system is stiff or not.

For stiff ODE systems (``stiff = TRUE``), ``RxODE`` uses the LSODA (Livermore Solver for Ordinary Differential Equations) Fortran package, which implements an automatic method switching for stiff and non-stiff problems along the integration interval, authored by Hindmarsh and Petzold (2003).

For non-stiff systems (``stiff = FALSE``), ``RxODE`` uses ``DOP853``,

an explicit Runge-Kutta method of order 8(5, 3) of Dormand and Prince as implemented in C by Hairer and Wanner (1993).

``trans_abs``: a logical (``FALSE`` by default) indicating whether to fit a transit absorption term (TODO: need further documentation and example);

``atol``: a numeric absolute tolerance (`1e-08` by default);

``rtol``: a numeric relative tolerance (`1e-06` by default).e

The output of `\dQuote{solve}` is a matrix with as many rows as there are sampled time points and as many columns as system variables (as defined by the ODEs and additional assignments in the RxODE model code).}

- \* ``isValid`` a function that (naively) checks for model validity, namely that the C object code reflects the latest model specification.
- \* ``version`` a string with the version of the ``RxODE`` object (not the package).
- \* ``dynLoad`` a function with one ``force = FALSE`` argument that dynamically loads the object code if needed.
- \* ``dynUnload`` a function with no argument that unloads the model object code.
- \* ``delete`` removes all created model files, including C and DLL files. The model object is no longer valid and should be removed, e.g., ``rm(m1)``.
- \* ``run`` deprecated, use ``solve``.
- \* ``get.index`` deprecated.
- \* ``getObj`` internal (not user callable) function.

### Author(s)

Melissa Hallow, Wenping Wang and Matthew Fidler

### References

- Chamber, J. M. and Temple Lang, D. (2001) *Object Oriented Programming in R*. R News, Vol. 1, No. 3, September 2001. [https://cran.r-project.org/doc/Rnews/Rnews\\_2001-3.pdf](https://cran.r-project.org/doc/Rnews/Rnews_2001-3.pdf).
- Hindmarsh, A. C. *ODEPACK, A Systematized Collection of ODE Solvers*. Scientific Computing, R. S. Stepleman et al. (Eds.), North-Holland, Amsterdam, 1983, pp. 55-64.
- Petzold, L. R. *Automatic Selection of Methods for Solving Stiff and Nonstiff Systems of Ordinary Differential Equations*. Siam J. Sci. Stat. Comput. 4 (1983), pp. 136-148.
- Hairer, E., Norsett, S. P., and Wanner, G. *Solving ordinary differential equations I, nonstiff problems*. 2nd edition, Springer Series in Computational Mathematics, Springer-Verlag (1993).
- Plevyak, J. dparser, <http://dparser.sourceforge.net>. Web. 12 Oct. 2015.

**See Also**

[eventTable\(\)](#), [et\(\)](#), [add.sampling\(\)](#), [add.dosing\(\)](#)

**Examples**

```
# Step 1 - Create a model specification
ode <- "
  # A 4-compartment model, 3 PK and a PD (effect) compartment
  # (notice state variable names 'depot', 'centr', 'peri', 'eff')

  C2 = centr/V2;
  C3 = peri/V3;
  d/dt(depota) = -KA*depota;
  d/dt(centr) = KA*depota - CL*C2 - Q*C2 + Q*C3;
  d/dt(peri) = Q*C2 - Q*C3;
  d/dt(eff) = Kin - Kout*(1-C2/(EC50+C2))*eff;
"

m1 <- RxODE(model = ode)
print(m1)

# Step 2 - Create the model input as an EventTable,
# including dosing and observation (sampling) events

# QD (once daily) dosing for 5 days.

qd <- eventTable(amount.units = "ug", time.units = "hours")
qd$add.dosing(dose = 10000, nbr.doses = 5, dosing.interval = 24)

# Sample the system hourly during the first day, every 8 hours
# then after

qd$add.sampling(0:24)
qd$add.sampling(seq(from = 24+8, to = 5*24, by = 8))

# Step 3 - set starting parameter estimates and initial
# values of the state

theta <-
  c(KA = .291, CL = 18.6,
    V2 = 40.2, Q = 10.5, V3 = 297.0,
    Kin = 1.0, Kout = 1.0, EC50 = 200.0)

# init state variable
inits <- c(0, 0, 0, 1);

# Step 4 - Fit the model to the data

qd.cp <- m1$solve(theta, events = qd, inits)
```

```
head(qd.cp)

# This returns a matrix. Note that you can also
# solve using name initial values. For example:

inits <- c(eff = 1);

qd.cp <- solve(m1, theta, events = qd, inits);

print(qd.cp)

plot(qd.cp)
```

---

rxOptExpr

*Optimize RxODE for computer evaluation*

---

### Description

This optimizes RxODE code for computer evaluation by only calculating redundant expressions once.

### Usage

```
rxOptExpr(x, msg = "model")
```

### Arguments

x	RxODE model that can be accessed by rxNorm
msg	This is the name of type of object that RxODE is optimizing that will in the message when optimizing. For example "model" will produce the following message while optimizing the model: finding duplicate expressions in model...

### Value

Optimized RxODE model text. The order and type lhs and state variables is maintained while the evaluation is sped up. While parameters names are maintained, their order may be modified.

### Author(s)

Matthew L. Fidler

---

`rxParams`*Parameters specified by the model*

---

**Description**

This returns the model's parameters that are required to solve the ODE system, and can be used to pipe parameters into an RxODE solve

**Usage**

```
rxParams(obj, ...)  
  
## S3 method for class 'RxODE'  
rxParams(  
  obj,  
  constants = TRUE,  
  ...,  
  params = NULL,  
  inits = NULL,  
  iCov = NULL,  
  keep = NULL,  
  thetaMat = NULL,  
  omega = NULL,  
  dfSub = NULL,  
  sigma = NULL,  
  dfObs = NULL,  
  nSub = NULL,  
  nStud = NULL  
)  
  
## S3 method for class 'rxSolve'  
rxParams(  
  obj,  
  constants = TRUE,  
  ...,  
  params = NULL,  
  inits = NULL,  
  iCov = NULL,  
  keep = NULL,  
  thetaMat = NULL,  
  omega = NULL,  
  dfSub = NULL,  
  sigma = NULL,  
  dfObs = NULL,  
  nSub = NULL,  
  nStud = NULL  
)
```



```
## S3 method for class 'rxEt'
rxParams(
  obj,
  ...,
  params = NULL,
  inits = NULL,
  iCov = NULL,
  keep = NULL,
  thetaMat = NULL,
  omega = NULL,
  dfSub = NULL,
  sigma = NULL,
  dfObs = NULL,
  nSub = NULL,
  nStud = NULL
)

rxParam(obj, ...)
```

### Arguments

obj	RxODE family of objects
...	Other arguments including scaling factors for each compartment. This includes S# = numeric will scale a compartment # by a dividing the compartment amount by the scale factor, like NONMEM.
constants	is a boolean indicting if constants should be included in the list of parameters. Currently RxODE parses constants into variables in case you wish to change them without recompiling the RxODE model.
params	a numeric named vector with values for every parameter in the ODE system; the names must correspond to the parameter identifiers used in the ODE specification;
inits	a vector of initial values of the state variables (e.g., amounts in each compartment), and the order in this vector must be the same as the state variables (e.g., PK/PD compartments);
iCov	A data frame of individual non-time varying covariates to combine with the params to form a parameter data.frame.
keep	Columns to keep from either the input dataset or the iCov dataset. With the iCov dataset, the column is kept once per line. For the input dataset, if any records are added to the data LOCF (Last Observation Carried forward) imputation is performed.
thetaMat	Named theta matrix.
omega	Estimate of Covariance matrix. When omega is a list, assume it is a block matrix and convert it to a full matrix for simulations.
dfSub	Degrees of freedom to sample the between subject variability matrix from the inverse Wishart distribution (scaled) or scaled inverse chi squared distribution.

sigma	Named sigma covariance or Cholesky decomposition of a covariance matrix. The names of the columns indicate parameters that are simulated. These are simulated for every observation in the solved system.
dfObs	Degrees of freedom to sample the unexplained variability matrix from the inverse Wishart distribution (scaled) or scaled inverse chi squared distribution.
nSub	Number between subject variabilities (ETAs) simulated for every realization of the parameters.
nStud	Number virtual studies to characterize uncertainty in estimated parameters.

**Value**

When extracting the parameters from an RxODE model, a character vector listing the parameters in the model.

**Author(s)**

Matthew L.Fidler

---

 rxPkg

---

*Creates a package from compiled RxODE models*


---

**Description**

Creates a package from compiled RxODE models

**Usage**

```
rxPkg(
  ...,
  package,
  wd = getwd(),
  action = c("install", "build", "binary", "create"),
  license = c("gpl3", "lgpl", "mit", "agpl3"),
  name = "Firstname Lastname",
  fields = list()
)
```

**Arguments**

...	Models to build a package from
package	String of the package name to create
wd	character string with a working directory where to create a subdirectory according to modName. When specified, a subdirectory named after the "modName.d" will be created and populated with a C file, a dynamic loading library, plus various other working files. If missing, the files are created (and removed) in the temporary directory, and the RxODE DLL for the model is created in the current directory named rx_????_platform, for example rx_129f8f97fb94a87ca49ca8dafe691e1e_i386.dll

action	Type of action to take after package is created
license	is the type of license for the package.
name	Full name of author
fields	A named list of fields to add to DESCRIPTION, potentially overriding default values. See <code>use_description()</code> for how you can set personalized defaults using package options

**Value**

this function returns nothing and is used for its side effects

**Author(s)**

Matthew Fidler

---

rxpois	<i>Simulate random Poisson variable from threefry generator</i>
--------	---

---

**Description**

Care should be taken with this method not to encounter the birthday problem, described <https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/>. Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

**Usage**

```
rxpois(lambda, n = 1L, ncores = 1L)
```

**Arguments**

lambda	vector of (non-negative) means.
n	number of random values to return.
ncores	Number of cores for the simulation rxnorm simulates using the threefry sitmo generator; rxnormV uses the vander-corput generator

**Details**

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to run all simulations in the RxODE environment once (therefore one seed or series of seeds for the whole simulation) or to pre-generate all random variables used for the simulation.

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

**Value**

poission random number deviates

**Examples**

```
## Use threefry engine

rxpois(lambda=3, n=10) # with rxpois you have to explicitly state n
rxpois(lambda=3, n=10, ncores=2) # You can parallelize the simulation using openMP

rxpois(4) ## The first arguments are the lambda parameter

## This example uses `rxpois` directly in the model

rx <- RxODE({
  a = rxpois(3)
})

et <- et(1,id=1:2)

s <- rxSolve(rx,et)
```

---

 rxPp

---

*Simulate a from a Poisson process*


---

**Description**

Simulate a from a Poisson process

**Usage**

```
rxPp(
  n,
  lambda,
  gamma = 1,
  prob = NULL,
  t0 = 0,
  tmax = Inf,
  randomOrder = FALSE
)
```

**Arguments**

n	Number of time points to simulate in the Poisson process
lambda	Rate of Poisson process

gamma	<p>Asymmetry rate of Poisson process. When gamma=1.0, this simulates a homogenous Poisson process. When gamma&lt;1.0, the Poisson process has more events early, when gamma &gt; 1.0, the Poisson process has more events late in the process.</p> <p>When gamma is non-zero, the tmax should not be infinite but indicate the end of the Poisson process to be simulated. In most pharamcometric cases, this will be the end of the study. Internally this uses a rate of:</p> $l(t) = \text{lambda} \text{gamma} (t/t_{\text{max}})^{\text{gamma}-1}$
prob	When specified, this is a probability function with one argument, time, that gives the probability that a Poisson time t is accepted as a rejection time.
t0	the starting time of the Poisson process
tmax	the maximum time of the Poisson process
randomOrder	when TRUE randomize the order of the Poisson events. By default (FALSE) it returns the Poisson process is in order of how the events occurred.

**Value**

This returns a vector of the Poisson process times; If the dropout is  $\geq$  tmax, then all the rest of the times are = tmax to indicate the dropout is equal to or after tmax.

**Author(s)**

Matthew Fidler

**Examples**

```
## Sample homogenous Poisson process of rate 1/10
rxPp(10, 1/10)

## Sample inhomogenous Poisson rate of 1/10
rxPp(10, 1/10, gamma=2, tmax=100)

## Typically the Poisson process times are in a sequential order,
## using randomOrder gives the Poisson process in random order
rxPp(10, 1/10, gamma=2, tmax=10, randomOrder=TRUE)

## This uses an arbitrary function to sample a non-homogenous Poisson process
rxPp(10, 1/10, prob=function(x){1/x})
```

---

rxProgress                      *RxODE progress bar functions*

---

**Description**

rxProgress sets up the progress bar

**Usage**

```
rxProgress(num, core = 0L)
rxTick()
rxProgressStop(clear = TRUE)
rxProgressAbort(error = "Aborted calculation")
```

**Arguments**

num	Tot number of operations to track
core	Number of cores to show. If below 1, don't show number of cores
clear	Boolean telling if you should clear the progress bar after completion (as if it wasn't displayed). By default this is TRUE
error	With rxProgressAbort this is the error that is displayed

**Details**

rxTick is a progress bar tick  
rxProgressStop stop progress bar  
rxProgressAbort shows an abort if rxProgressStop wasn't called.

**Value**

All return NULL invisibly.

**Author(s)**

Matthew L. Fidler

**Examples**

```
f <- function(){
  on.exit({rxProgressAbort()});
  rxProgress(100)
  for (i in 1:100) {
    rxTick()
    Sys.sleep(1 / 100)
  }
}
```

```
    }  
    rxProgressStop();  
}
```

```
f()
```

---

rxRandNV

*Create a random "normal" matrix using vandercorput generator*

---

### **Description**

Create a random "normal" matrix using vandercorput generator

### **Usage**

```
rxRandNV(nrow = 1, ncol = 1)
```

### **Arguments**

nrow	Number of rows
ncol	Number of Columns

### **Value**

Matrix of random numbers

### **Author(s)**

Matthew Fidler

### **Examples**

```
rxRandNV(1, 1)  
rxRandNV(3, 2)
```

---

rxRateDur	<i>Creates a rxRateDur object</i>
-----------	-----------------------------------

---

**Description**

This is primarily to display information about rate

**Usage**

```
rxRateDur(x)

## S3 method for class 'rxRateDur'
x[...]

as.rxRateDur(x)

## S3 method for class 'rxRateDur'
as.character(x, ...)

## S3 method for class 'rxRateDur'
x[[...]]

type_sum.rxRateDur(x)

pillar_shaft.rxRateDur(x, ...)
```

**Arguments**

x	rxRateDur data
...	Other parameters

**Value**

rxRateDur object

---

rxReservedKeywords	<i>A list and description of Rode supported reserved keywords</i>
--------------------	---

---

**Description**

A list and description of Rode supported reserved keywords

**Usage**

```
rxReservedKeywords
```



**Format**

A data frame with 3 columns and 98 or more rows

**Reserved Name** Reserved Keyword Name

**Meaning** Reserved Keyword Meaning

**Alias** Keyword Alias

---

 rxRmvn

---

*Simulate from a (truncated) multivariate normal*


---

**Description**

This is simulated with the fast, thread-safe threefry simulator and can use multiple cores to generate the random deviates.

**Usage**

```
rxRmvn(
  n,
  mu = NULL,
  sigma,
  lower = -Inf,
  upper = Inf,
  ncores = 1,
  isChol = FALSE,
  keepNames = TRUE,
  a = 0.4,
  tol = 2.05,
  nlTol = 1e-10,
  nlMaxiter = 100L
)
```

**Arguments**

n	Number of random row vectors to be simulated OR the matrix to use for simulation (faster).
mu	mean vector
sigma	Covariance matrix for multivariate normal or a list of covariance matrices. If a list of covariance matrix, each matrix will simulate n matrices and combine them to a full matrix
lower	is a vector of the lower bound for the truncated multivariate norm
upper	is a vector of the upper bound for the truncated multivariate norm
ncores	Number of cores used in the simulation
isChol	A boolean indicating if sigma is a cholesky decomposition of the covariance matrix.

keepNames	Keep the names from either the mean or covariance matrix.
a	threshold for switching between methods; They can be tuned for maximum speed; There are three cases that are considered: case 1: $a < l < u$ case 2: $l < u < -a$ case 3: otherwise where $l$ =lower and $u$ = upper
tol	When case 3 is used from the above possibilities, the tol value controls the acceptance rejection and inverse-transformation; When $\text{abs}(u-l) > \text{tol}$ , uses accept-reject from randn
n1Tol	Tolerance for newton line-search
n1Maxiter	Maximum iterations for newton line-search

**Value**

If  $n$ =integer (default) the output is an  $(n \times d)$  matrix where the  $i$ -th row is the  $i$ -th simulated vector.

If  $\text{is.matrix}(n)$  then the random vector are store in  $n$ , which is provided by the user, and the function returns NULL invisibly.

**Author(s)**

Matthew Fidler, Zdravko Botev and some from Matteo Fasiolo

**References**

John K. Salmon, Mark A. Moraes, Ron O. Dror, and David E. Shaw (2011). Parallel Random Numbers: As Easy as 1, 2, 3. D. E. Shaw Research, New York, NY 10036, USA.

The thread safe multivariate normal was inspired from the mvnfast package by Matteo Fasiolo <https://CRAN.R-project.org/package=mvnfast>

The concept of the truncated multivariate normal was taken from Zdravko Botev Botev (2017) doi: [10.1111/rssb.12162](https://doi.org/10.1111/rssb.12162) and Botev and L'Ecuyer (2015) doi: [10.1109/WSC.2015.7408180](https://doi.org/10.1109/WSC.2015.7408180) and converted to thread safe simulation;

**Examples**

```
## From mvnfast
## Unlike mvnfast, uses threefry simulation

d <- 5
mu <- 1:d

# Creating covariance matrix
tmp <- matrix(rnorm(d^2), d, d)
mcov <- tcrossprod(tmp, tmp)
```

```

set.seed(414)
rxRmvn(4, 1:d, mcov)

set.seed(414)
rxRmvn(4, 1:d, mcov)

set.seed(414)
rxRmvn(4, 1:d, mcov, ncores = 2) # r.v. generated on the second core are different

##### Here we create the matrix that will hold the simulated
# random variables upfront.
A <- matrix(NA, 4, d)
class(A) <- "numeric" # This is important. We need the elements of A to be of class "numeric".

set.seed(414)
rxRmvn(A, 1:d, mcov, ncores = 2) # This returns NULL ...
A                                # ... but the result is here

## You can also simulate from a truncated normal:

rxRmvn(10, 1:d, mcov, lower=1:d-1, upper=1:d+1)

# You can also simulate from different matrices (if they match
# dimensions) by using a list of matrices.

matL <- lapply(1:4,function(...){
  tmp <- matrix(rnorm(d^2), d, d)
  tcrossprod(tmp, tmp)
})

rxRmvn(4, setNames(1:d,paste0("a",1:d)), matL)

```

---

rxS

*Load a model into a symengine environment*


---

## Description

Load a model into a symengine environment

## Usage

```
rxS(x, doConst = TRUE, promoteLinSens = FALSE)
```

**Arguments**

x	RxODE object
doConst	Load constants into the environment as well.
promoteLinSens	Promote solved linear compartment systems to sensitivity-based solutions.

**Value**

RxODE/symengine environment

**Author(s)**

Matthew Fidler

---

rxSetIni0	<i>Set Initial conditions to time zero instead of the first observed/dosed time</i>
-----------	---

---

**Description**

Set Initial conditions to time zero instead of the first observed/dosed time

**Usage**

```
rxSetIni0(ini0 = TRUE)
```

**Arguments**

ini0	When TRUE (default), set initial conditions to time zero. Otherwise the initial conditions are the first observed time.
------	---

**Value**

the boolean ini0, though this is called for its side effects

---

rxSetProd	<i>Defunct setting of product</i>
-----------	-----------------------------------

---

**Description**

Defunct setting of product

**Usage**

```
rxSetProd(type = c("long double", "double", "logify"))
```

**Arguments**

type	used to be type of product
------	----------------------------

**Value**

nothing

---

rxSetProgressBar	<i>Set timing for progress bar</i>
------------------	------------------------------------

---

**Description**

Set timing for progress bar

**Usage**

```
rxSetProgressBar(seconds = 1)
```

**Arguments**

seconds	This sets the number of seconds that need to elapse before drawing the next segment of the progress bar. When this is zero or below this turns off the progress bar.
---------	--

**Value**

nothing, used for side effects

**Author(s)**

Matthew Fidler

---

rxSetSum	<i>Defunct setting of sum</i>
----------	-------------------------------

---

**Description**

Defunct setting of sum

**Usage**

```
rxSetSum(type = c("pairwise", "fsum", "kahan", "neumaier", "c"))
```

**Arguments**

type                    used to be type of product

**Value**

nothing

---

rxShiny	<i>Use Shiny to help develop an RxODE model</i>
---------	---

---

**Description**

Use Shiny to help develop an RxODE model

**Usage**

```
rxShiny(
  object,
  params = NULL,
  events = NULL,
  inits = NULL,
  ...,
  data = data.frame()
)

## S3 method for class 'rxSolve'
rxShiny(
  object,
  params = NULL,
  events = NULL,
  inits = NULL,
  ...,
  data = data.frame()
)
```

```
## Default S3 method:
rxShiny(
  object = NULL,
  params = NULL,
  events = NULL,
  inits = NULL,
  ...,
  data = data.frame()
)
```

### Arguments

object	A RxODE family of objects. If not supplied a 2-compartment indirect effect model is used. If it is supplied, use the model associated with the RxODE object for the model exploration.
params	Initial parameters for model
events	Event information (currently ignored)
inits	Initial estimates for model
...	Other arguments passed to rxShiny. Currently doesn't do anything.
data	Any data that you would like to plot. If the data has a time variable as well as a compartment or calculated variable that matches the RxODE model, the data will be added to the plot of a specific compartment or calculated variable.

### Value

Nothing; Starts a shiny server

### Author(s)

Zufar Mulyukov and Matthew L. Fidler

---

rxSimThetaOmega

*Simulate Parameters from a Theta/Omega specification*

---

### Description

Simulate Parameters from a Theta/Omega specification

### Usage

```
rxSimThetaOmega(
  params = NULL,
  omega = NULL,
  omegaDf = NULL,
  omegaLower = as.numeric(c(R_NegInf)),
```

```

omegaUpper = as.numeric(c(R_PosInf)),
omegaIsChol = FALSE,
omegaSeparation = "auto",
omegaXform = 1L,
nSub = 1L,
thetaMat = NULL,
thetaLower = as.numeric(c(R_NegInf)),
thetaUpper = as.numeric(c(R_PosInf)),
thetaDf = NULL,
thetaIsChol = FALSE,
nStud = 1L,
sigma = NULL,
sigmaLower = as.numeric(c(R_NegInf)),
sigmaUpper = as.numeric(c(R_PosInf)),
sigmaDf = NULL,
sigmaIsChol = FALSE,
sigmaSeparation = "auto",
sigmaXform = 1L,
nCoresRV = 1L,
nObs = 1L,
dfSub = 0,
dfObs = 0,
simSubjects = TRUE
)

```

### Arguments

params	Named Vector of RxODE model parameters
omega	Estimate of Covariance matrix. When omega is a list, assume it is a block matrix and convert it to a full matrix for simulations.
omegaDf	The degrees of freedom of a t-distribution for simulation. By default this is NULL which is equivalent to Inf degrees, or to simulate from a normal distribution instead of a t-distribution.
omegaLower	Lower bounds for simulated ETAs (by default -Inf)
omegaUpper	Upper bounds for simulated ETAs (by default Inf)
omegaIsChol	Indicates if the omega supplied is a Cholesky decomposed matrix instead of the traditional symmetric matrix.
omegaSeparation	<p>Omega separation strategy</p> <p>Tells the type of separation strategy when simulating covariance with parameter uncertainty with standard deviations modeled in the thetaMat matrix.</p> <ul style="list-style-type: none"> <li>• "lkj" simulates the correlation matrix from the rLKJ1 matrix with the distribution parameter eta equal to the degrees of freedom nu by (nu-1)/2</li> <li>• "separation" simulates from the identity inverse Wishart covariance matrix with nu degrees of freedom. This is then converted to a covariance</li> </ul>



	matrix and augmented with the modeled standard deviations. While computationally more complex than the "lkj" prior, it performs better when the covariance matrix size is greater or equal to 10
	<ul style="list-style-type: none"> <li>• "auto" chooses "lkj" when the dimension of the matrix is less than 10 and "separation" when greater than equal to 10.</li> </ul>
omegaXform	<p>When taking omega values from the thetaMat simulations (using the separation strategy for covariance simulation), how should the thetaMat values be turned into standard deviation values:</p> <ul style="list-style-type: none"> <li>• identity This is when standard deviation values are directly modeled by the params and thetaMat matrix</li> <li>• variance This is when the params and thetaMat simulates the variance that are directly modeled by the thetaMat matrix</li> <li>• log This is when the params and thetaMat simulates <math>\log(sd)</math></li> <li>• nlmixrSqrt This is when the params and thetaMat simulates the inverse cholesky decomposed matrix with the <math>x^2</math> modeled along the diagonal. This only works with a diagonal matrix.</li> <li>• nlmixrLog This is when the params and thetaMat simulates the inverse cholesky decomposed matrix with the <math>\exp(x^2)</math> along the diagonal. This only works with a diagonal matrix.</li> <li>• nlmixrIdentity This is when the params and thetaMat simulates the inverse cholesky decomposed matrix. This only works with a diagonal matrix.</li> </ul>
nSub	Number between subject variabilities (ETAs) simulated for every realization of the parameters.
thetaMat	Named theta matrix.
thetaLower	Lower bounds for simulated population parameter variability (by default -Inf)
thetaUpper	Upper bounds for simulated population unexplained variability (by default Inf)
thetaDf	The degrees of freedom of a t-distribution for simulation. By default this is NULL which is equivalent to Inf degrees, or to simulate from a normal distribution instead of a t-distribution.
thetaIsChol	Indicates if the theta supplied is a Cholesky decomposed matrix instead of the traditional symmetric matrix.
nStud	Number virtual studies to characterize uncertainty in estimated parameters.
sigma	Named sigma covariance or Cholesky decomposition of a covariance matrix. The names of the columns indicate parameters that are simulated. These are simulated for every observation in the solved system.
sigmaLower	Lower bounds for simulated unexplained variability (by default -Inf)
sigmaUpper	Upper bounds for simulated unexplained variability (by default Inf)
sigmaDf	Degrees of freedom of the sigma t-distribution. By default it is equivalent to Inf, or a normal distribution.
sigmaIsChol	Boolean indicating if the sigma is in the Cholesky decomposition instead of a symmetric covariance

<code>sigmaSeparation</code>	<p>separation strategy for sigma;</p> <p>Tells the type of separation strategy when simulating covariance with parameter uncertainty with standard deviations modeled in the <code>thetaMat</code> matrix.</p> <ul style="list-style-type: none"> <li>• "lkj" simulates the correlation matrix from the rLKJ1 matrix with the distribution parameter <math>\eta</math> equal to the degrees of freedom <math>\nu</math> by <math>(\nu-1)/2</math></li> <li>• "separation" simulates from the identity inverse Wishart covariance matrix with <math>\nu</math> degrees of freedom. This is then converted to a covariance matrix and augmented with the modeled standard deviations. While computationally more complex than the "lkj" prior, it performs better when the covariance matrix size is greater or equal to 10</li> <li>• "auto" chooses "lkj" when the dimension of the matrix is less than 10 and "separation" when greater than equal to 10.</li> </ul>
<code>sigmaXform</code>	<p>When taking <code>sigma</code> values from the <code>thetaMat</code> simulations (using the separation strategy for covariance simulation), how should the <code>thetaMat</code> values be turned into standard deviation values:</p> <ul style="list-style-type: none"> <li>• <code>identity</code> This is when standard deviation values are directly modeled by the <code>params</code> and <code>thetaMat</code> matrix</li> <li>• <code>variance</code> This is when the <code>params</code> and <code>thetaMat</code> simulates the variance that are directly modeled by the <code>thetaMat</code> matrix</li> <li>• <code>log</code> This is when the <code>params</code> and <code>thetaMat</code> simulates <math>\log(\text{sd})</math></li> <li>• <code>nlmixrSqrt</code> This is when the <code>params</code> and <code>thetaMat</code> simulates the inverse cholesky decomposed matrix with the <math>x^2</math> modeled along the diagonal. This only works with a diagonal matrix.</li> <li>• <code>nlmixrLog</code> This is when the <code>params</code> and <code>thetaMat</code> simulates the inverse cholesky decomposed matrix with the <math>\exp(x^2)</math> along the diagonal. This only works with a diagonal matrix.</li> <li>• <code>nlmixrIdentity</code> This is when the <code>params</code> and <code>thetaMat</code> simulates the inverse cholesky decomposed matrix. This only works with a diagonal matrix.</li> </ul>
<code>nCoresRV</code>	<p>Number of cores used for the simulation of the <code>sigma</code> variables. By default this is 1. To reproduce the results you need to run on the same platform with the same number of cores. This is the reason this is set to be one, regardless of what the number of cores are used in threaded ODE solving.</p>
<code>nObs</code>	<p>Number of observations to simulate (with <code>sigma</code> matrix)</p>
<code>dfSub</code>	<p>Degrees of freedom to sample the between subject variability matrix from the inverse Wishart distribution (scaled) or scaled inverse chi squared distribution.</p>
<code>dfObs</code>	<p>Degrees of freedom to sample the unexplained variability matrix from the inverse Wishart distribution (scaled) or scaled inverse chi squared distribution.</p>
<code>simSubjects</code>	<p>boolean indicated R<sub>x</sub>ODE should simulate subjects in studies (TRUE, default) or studies (FALSE)</p>

**Value**

a data frame with the simulated subjects

**Author(s)**

Matthew L.Fidler

---

rxSolve	<i>Solving &amp; Simulation of a ODE/solved system (and solving options) equation</i>
---------	---

---

**Description**

This uses RxODE family of objects, file, or model specification to solve a ODE system. There are many options for a solved RxODE model, the first are the required object, and events with the some-times optional params and inits.

**Usage**

```
rxSolve(
  object,
  params = NULL,
  events = NULL,
  inits = NULL,
  scale = NULL,
  method = c("liblsoda", "lsoda", "dop853", "indLin"),
  transitAbs = NULL,
  atol = 1e-08,
  rtol = 1e-06,
  maxsteps = 70000L,
  hmin = 0,
  hmax = NA_real_,
  hmaxSd = 0,
  hini = 0,
  maxordn = 12L,
  maxords = 5L,
  ...,
  cores,
  covsInterpolation = c("locf", "linear", "nocb", "midpoint"),
  addCov = FALSE,
  matrix = FALSE,
  sigma = NULL,
  sigmaDf = NULL,
  sigmaLower = -Inf,
  sigmaUpper = Inf,
  nCoresRV = 1L,
  sigmaIsChol = FALSE,
  sigmaSeparation = c("auto", "lkj", "separation"),
  sigmaXform = c("identity", "variance", "log", "nlmixrSqrt", "nlmixrLog",
    "nlmixrIdentity"),
```

```
nDisplayProgress = 10000L,  
amountUnits = NA_character_,  
timeUnits = "hours",  
stiff,  
theta = NULL,  
thetaLower = -Inf,  
thetaUpper = Inf,  
eta = NULL,  
addDosing = FALSE,  
stateTrim = Inf,  
updateObject = FALSE,  
omega = NULL,  
omegaDf = NULL,  
omegaIsChol = FALSE,  
omegaSeparation = c("auto", "lkj", "separation"),  
omegaXform = c("variance", "identity", "log", "nlmixrSqrt", "nlmixrLog",  
  "nlmixrIdentity"),  
omegaLower = -Inf,  
omegaUpper = Inf,  
nSub = 1L,  
thetaMat = NULL,  
thetaDf = NULL,  
thetaIsChol = FALSE,  
nStud = 1L,  
dfSub = 0,  
dfObs = 0,  
returnType = c("rxSolve", "matrix", "data.frame", "data.frame.TBS", "data.table",  
  "tbl", "tibble"),  
seed = NULL,  
nsim = NULL,  
minSS = 10L,  
maxSS = 1000L,  
infSSstep = 12,  
strictSS = TRUE,  
istateReset = TRUE,  
subsetNonmem = TRUE,  
maxAtoIRtolFactor = 0.1,  
from = NULL,  
to = NULL,  
by = NULL,  
length.out = NULL,  
iCov = NULL,  
keep = NULL,  
indLinPhiTol = 1e-07,  
indLinPhiM = 0L,  
indLinMatExpType = c("expokit", "Al-Mohy", "arma"),  
indLinMatExpOrder = 6L,  
drop = NULL,
```

```
idFactor = TRUE,
mxhnil = 0,
hmxi = 0,
warnIdSort = TRUE,
warnDrop = TRUE,
ssAto1 = 1e-08,
ssRto1 = 1e-06,
safeZero = TRUE,
sumType = c("pairwise", "fsum", "kahan", "neumaier", "c"),
prodType = c("long double", "double", "logify"),
sensType = c("advan", "autodiff", "forward", "central"),
linDiff = c(tlag = 1.5e-05, f = 1.5e-05, rate = 1.5e-05, dur = 1.5e-05, tlag2 =
  1.5e-05, f2 = 1.5e-05, rate2 = 1.5e-05, dur2 = 1.5e-05),
linDiffCentral = c(tlag = TRUE, f = TRUE, rate = TRUE, dur = TRUE, tlag2 = TRUE, f2 =
  TRUE, rate2 = TRUE, dur2 = TRUE),
resample = NULL,
resampleID = TRUE,
maxwhile = 1e+05
)

## Default S3 method:
rxSolve(
  object,
  params = NULL,
  events = NULL,
  inits = NULL,
  ...,
  theta = NULL,
  eta = NULL
)

## S3 method for class 'rxSolve'
update(object, ...)

## S3 method for class 'RxODE'
predict(object, ...)

## S3 method for class 'rxSolve'
predict(object, ...)

## S3 method for class 'rxEt'
predict(object, ...)

## S3 method for class 'rxParams'
predict(object, ...)

## S3 method for class 'RxODE'
simulate(object, nsim = 1L, seed = NULL, ...)
```

```

## S3 method for class 'rxSolve'
simulate(object, nsim = 1L, seed = NULL, ...)

## S3 method for class 'rxParams'
simulate(object, nsim = 1L, seed = NULL, ...)

## S3 method for class 'rxSolve'
solve(a, b, ...)

## S3 method for class 'RxODE'
solve(a, b, ...)

## S3 method for class 'rxParams'
solve(a, b, ...)

## S3 method for class 'rxEt'
solve(a, b, ...)

rxControl(..., params = NULL, events = NULL, inits = NULL)

```

## Arguments

object	is either a RxODE family of objects, or a file-name with a RxODE model specification, or a string with a RxODE model specification.
params	a numeric named vector with values for every parameter in the ODE system; the names must correspond to the parameter identifiers used in the ODE specification;
events	an eventTable object describing the input (e.g., doses) to the dynamic system and observation sampling time points (see <a href="#">eventTable()</a> );
inits	a vector of initial values of the state variables (e.g., amounts in each compartment), and the order in this vector must be the same as the state variables (e.g., PK/PD compartments);
scale	a numeric named vector with scaling for ode parameters of the system. The names must correspond to the parameter identifiers in the ODE specification. Each of the ODE variables will be divided by the scaling factor. For example <code>scale=c(center=2)</code> will divide the center ODE variable by 2.
method	The method for solving ODEs. Currently this supports: <ul style="list-style-type: none"> <li>• "liblsoda" thread safe lsoda. This supports parallel thread-based solving, and ignores user Jacobian specification.</li> <li>• "lsoda" – LSODA solver. Does not support parallel thread-based solving, but allows user Jacobian specification.</li> <li>• "dop853" – DOP853 solver. Does not support parallel thread-based solving nor user Jacobian specification</li> <li>• "indLin" – Solving through inductive linearization. The RxODE dll must be setup specially to use this solving routine.</li> </ul>

transitAbs	boolean indicating if this is a transit compartment absorption
atol	a numeric absolute tolerance (1e-8 by default) used by the ODE solver to determine if a good solution has been achieved; This is also used in the solved linear model to check if prior doses do not add anything to the solution.
rtol	a numeric relative tolerance (1e-6 by default) used by the ODE solver to determine if a good solution has been achieved. This is also used in the solved linear model to check if prior doses do not add anything to the solution.
maxsteps	maximum number of (internally defined) steps allowed during one call to the solver. (5000 by default)
hmin	The minimum absolute step size allowed. The default value is 0.
hmax	The maximum absolute step size allowed. When hmax=NA (default), uses the average difference + hmaxSd*sd in times and sampling events. The hmaxSd is a user specified parameter and which defaults to zero. When hmax=NULL RxODE uses the maximum difference in times in your sampling and events. The value 0 is equivalent to infinite maximum absolute step size.
hmaxSd	The number of standard deviations of the time difference to add to hmax. The default is 0
hini	The step size to be attempted on the first step. The default value is determined by the solver (when hini = 0)
maxordn	The maximum order to be allowed for the nonstiff (Adams) method. The default is 12. It can be between 1 and 12.
maxords	The maximum order to be allowed for the stiff (BDF) method. The default value is 5. This can be between 1 and 5.
...	Other arguments including scaling factors for each compartment. This includes S# = numeric will scale a compartment # by a dividing the compartment amount by the scale factor, like NONMEM.
cores	Number of cores used in parallel ODE solving. This is equivalent to calling <a href="#">setRxThreads()</a>
covsInterpolation	<p>specifies the interpolation method for time-varying covariates. When solving ODEs it often samples times outside the sampling time specified in events. When this happens, the time varying covariates are interpolated. Currently this can be:</p> <ul style="list-style-type: none"> <li>• "linear" interpolation, which interpolates the covariate by solving the line between the observed covariates and extrapolating the new covariate value.</li> <li>• "constant" – Last observation carried forward (the default).</li> <li>• "NOCB" – Next Observation Carried Backward. This is the same method that NONMEM uses.</li> <li>• "midpoint" Last observation carried forward to midpoint; Next observation carried backward to midpoint.</li> </ul>
addCov	A boolean indicating if covariates should be added to the output matrix or data frame. By default this is disabled.
matrix	A boolean indicating if a matrix should be returned instead of the RxODE's solved object.

sigma	Named sigma covariance or Cholesky decomposition of a covariance matrix. The names of the columns indicate parameters that are simulated. These are simulated for every observation in the solved system.
sigmaDf	Degrees of freedom of the sigma t-distribution. By default it is equivalent to Inf, or a normal distribution.
sigmaLower	Lower bounds for simulated unexplained variability (by default -Inf)
sigmaUpper	Upper bounds for simulated unexplained variability (by default Inf)
nCoresRV	Number of cores used for the simulation of the sigma variables. By default this is 1. To reproduce the results you need to run on the same platform with the same number of cores. This is the reason this is set to be one, regardless of what the number of cores are used in threaded ODE solving.
sigmaIsChol	Boolean indicating if the sigma is in the Cholesky decomposition instead of a symmetric covariance
sigmaSeparation	<p>separation strategy for sigma; Tells the type of separation strategy when simulating covariance with parameter uncertainty with standard deviations modeled in the thetaMat matrix.</p> <ul style="list-style-type: none"> <li>• "lkj" simulates the correlation matrix from the rLKJ1 matrix with the distribution parameter eta equal to the degrees of freedom nu by <math>(nu-1)/2</math></li> <li>• "separation" simulates from the identity inverse Wishart covariance matrix with nu degrees of freedom. This is then converted to a covariance matrix and augmented with the modeled standard deviations. While computationally more complex than the "lkj" prior, it performs better when the covariance matrix size is greater or equal to 10</li> <li>• "auto" chooses "lkj" when the dimension of the matrix is less than 10 and "separation" when greater than equal to 10.</li> </ul>
sigmaXform	<p>When taking sigma values from the thetaMat simulations (using the separation strategy for covariance simulation), how should the thetaMat values be turned into standard deviation values:</p> <ul style="list-style-type: none"> <li>• identity This is when standard deviation values are directly modeled by the params and thetaMat matrix</li> <li>• variance This is when the params and thetaMat simulates the variance that are directly modeled by the thetaMat matrix</li> <li>• log This is when the params and thetaMat simulates <math>\log(sd)</math></li> <li>• nlmixrSqrt This is when the params and thetaMat simulates the inverse cholesky decomposed matrix with the <math>x^2</math> modeled along the diagonal. This only works with a diagonal matrix.</li> <li>• nlmixrLog This is when the params and thetaMat simulates the inverse cholesky decomposed matrix with the <math>\exp(x^2)</math> along the diagonal. This only works with a diagonal matrix.</li> <li>• nlmixrIdentity This is when the params and thetaMat simulates the inverse cholesky decomposed matrix. This only works with a diagonal matrix.</li> </ul>
nDisplayProgress	An integer indicating the minimum number of c-based solves before a progress bar is shown. By default this is 10,000.



amountUnits	This supplies the dose units of a data frame supplied instead of an event table. This is for importing the data as an RxODE event table.
timeUnits	This supplies the time units of a data frame supplied instead of an event table. This is for importing the data as an RxODE event table.
stiff	<p>a logical (TRUE by default) indicating whether the ODE system is stiff or not.</p> <p>For stiff ODE systems (<code>`stiff = TRUE`</code>), <code>`RxODE`</code> uses the LSODA (Livermore Solver for Ordinary Differential Equations) Fortran package, which implements an automatic method switching for stiff and non-stiff problems along the integration interval, authored by Hindmarsh and Petzold (2003).</p> <p>For non-stiff systems (<code>`stiff = FALSE`</code>), <code>`RxODE`</code> uses DOP853, an explicit Runge-Kutta method of order 8(5, 3) of Dormand and Prince as implemented in C by Hairer and Wanner (1993).</p> <p>If <code>stiff</code> is not specified, the <code>`method`</code> argument is used instead.</p>
theta	A vector of parameters that will be named THETA[#] and added to parameters
thetaLower	Lower bounds for simulated population parameter variability (by default <code>-Inf</code> )
thetaUpper	Upper bounds for simulated population unexplained variability (by default <code>Inf</code> )
eta	A vector of parameters that will be named ETA[#] and added to parameters
addDosing	<p>Boolean indicating if the solve should add RxODE EVID and related columns. This will also include dosing information and estimates at the doses. By default, RxODE only includes estimates at the observations. (default <code>FALSE</code>). When <code>addDosing</code> is <code>NULL</code>, only include <code>EVID=0</code> on solve and exclude any model-times or <code>EVID=2</code>. If <code>addDosing</code> is <code>NA</code> the classic RxODE EVID events are returned. When <code>addDosing</code> is <code>TRUE</code> add the event information in NONMEM-style format; If <code>subsetNonmem=FALSE</code> RxODE will also include extra event types (EVID) for ending infusion and modeled times:</p> <ul style="list-style-type: none"> <li>• <code>EVID=-1</code> when the modeled rate infusions are turned off (matches <code>rate=-1</code>)</li> <li>• <code>EVID=-2</code> When the modeled duration infusions are turned off (matches <code>rate=-2</code>)</li> <li>• <code>EVID=-10</code> When the specified rate infusions are turned off (matches <code>rate&gt;0</code>)</li> <li>• <code>EVID=-20</code> When the specified dur infusions are turned off (matches <code>dur&gt;0</code>)</li> <li>• <code>EVID=101,102,103,...</code> Modeled time where 101 is the first model time, 102 is the second etc.</li> </ul>
stateTrim	When amounts/concentrations in one of the states are above this value, trim them to be this value. By default <code>Inf</code> . Also trims to <code>-stateTrim</code> for large negative amounts/concentrations. If you want to trim between a range say <code>c(0, 2000000)</code> you may specify 2 values with a lower and upper range to make sure all state values are in the reasonable range.
updateObject	This is an internally used flag to update the RxODE solved object (when supplying an RxODE solved object) as well as returning a new object. You probably should not modify its <code>FALSE</code> default unless you are willing to have unexpected results.

omega	Estimate of Covariance matrix. When omega is a list, assume it is a block matrix and convert it to a full matrix for simulations.
omegaDf	The degrees of freedom of a t-distribution for simulation. By default this is NULL which is equivalent to Inf degrees, or to simulate from a normal distribution instead of a t-distribution.
omegaIsChol	Indicates if the omega supplied is a Cholesky decomposed matrix instead of the traditional symmetric matrix.
omegaSeparation	<p>Omega separation strategy</p> <p>Tells the type of separation strategy when simulating covariance with parameter uncertainty with standard deviations modeled in the thetaMat matrix.</p> <ul style="list-style-type: none"> <li>• "lkj" simulates the correlation matrix from the rLKJ1 matrix with the distribution parameter eta equal to the degrees of freedom nu by (nu-1)/2</li> <li>• "separation" simulates from the identity inverse Wishart covariance matrix with nu degrees of freedom. This is then converted to a covariance matrix and augmented with the modeled standard deviations. While computationally more complex than the "lkj" prior, it performs better when the covariance matrix size is greater or equal to 10</li> <li>• "auto" chooses "lkj" when the dimension of the matrix is less than 10 and "separation" when greater than equal to 10.</li> </ul>
omegaXform	<p>When taking omega values from the thetaMat simulations (using the separation strategy for covariance simulation), how should the thetaMat values be turned int standard deviation values:</p> <ul style="list-style-type: none"> <li>• identity This is when standard deviation values are directly modeled by the params and thetaMat matrix</li> <li>• variance This is when the params and thetaMat simulates the variance that are directly modeled by the thetaMat matrix</li> <li>• log This is when the params and thetaMat simulates log(sd)</li> <li>• nlmixrSqrt This is when the params and thetaMat simulates the inverse cholesky decomposed matrix with the <math>x^2</math> modeled along the diagonal. This only works with a diagonal matrix.</li> <li>• nlmixrLog This is when the params and thetaMat simulates the inverse cholesky decomposed matrix with the <math>\exp(x^2)</math> along the diagonal. This only works with a diagonal matrix.</li> <li>• nlmixrIdentity This is when the params and thetaMat simulates the inverse cholesky decomposed matrix. This only works with a diagonal matrix.</li> </ul>
omegaLower	Lower bounds for simulated ETAs (by default -Inf)
omegaUpper	Upper bounds for simulated ETAs (by default Inf)
nSub	Number between subject variabilities (ETAs) simulated for every realization of the parameters.
thetaMat	Named theta matrix.
thetaDf	The degrees of freedom of a t-distribution for simulation. By default this is NULL which is equivalent to Inf degrees, or to simulate from a normal distribution instead of a t-distribution.

thetaIsChol	Indicates if the theta supplied is a Cholesky decomposed matrix instead of the traditional symmetric matrix.
nStud	Number virtual studies to characterize uncertainty in estimated parameters.
dfSub	Degrees of freedom to sample the between subject variability matrix from the inverse Wishart distribution (scaled) or scaled inverse chi squared distribution.
dfObs	Degrees of freedom to sample the unexplained variability matrix from the inverse Wishart distribution (scaled) or scaled inverse chi squared distribution.
returnType	This tells what type of object is returned. The currently supported types are: <ul style="list-style-type: none"> <li>• "rxSolve" (default) will return a reactive data frame that can change easily change different pieces of the solve and update the data frame. This is the currently standard solving method in RxODE, is used for rxSolve(object, ...), solve(object, ...),</li> <li>• "data.frame" – returns a plain, non-reactive data frame; Currently very slightly faster than returnType="matrix"</li> <li>• "matrix" – returns a plain matrix with column names attached to the solved object. This is what is used object\$run as well as object\$solve</li> <li>• "data.table" – returns a data.table; The data.table is created by reference (ie setDt()), which should be fast.</li> <li>• "tbl" or "tibble" returns a tibble format.</li> </ul>
seed	an object specifying if and how the random number generator should be initialized
nsim	represents the number of simulations. For RxODE, if you supply single subject event tables (created with [eventTable()])
minSS	Minimum number of iterations for a steady-state dose
maxSS	Maximum number of iterations for a steady-state dose
infSSstep	Step size for determining if a constant infusion has reached steady state. By default this is large value, 420.
strictSS	Boolean indicating if a strict steady-state is required. If a strict steady-state is (TRUE) required then at least minSS doses are administered and the total number of steady states doses will continue until maxSS is reached, or atol and rtol for every compartment have been reached. However, if ODE solving problems occur after the minSS has been reached the whole subject is considered an invalid solve. If strictSS is FALSE then as long as minSS has been reached the last good solve before ODE solving problems occur is considered the steady state, even though either atol, rtol or maxSS have not been achieved.
istateReset	When TRUE, reset the ISTATE variable to 1 for lsoda and liblsoda with doses, like deSolve; When FALSE, do not reset the ISTATE variable with doses.
subsetNonmem	subset to NONMEM compatible EVIDs only. By default TRUE.
maxAtolRtolFactor	The maximum atol/rtol that FOCEi and other routines may adjust to. By default 0.1
from	When there is no observations in the event table, start observations at this value. By default this is zero.

to	When there is no observations in the event table, end observations at this value. By default this is 24 + maximum dose time.
by	When there are no observations in the event table, this is the amount to increment for the observations between from and to.
length.out	The number of observations to create if there isn't any observations in the event table. By default this is 200.
iCov	A data frame of individual non-time varying covariates to combine with the params to form a parameter data.frame.
keep	Columns to keep from either the input dataset or the iCov dataset. With the iCov dataset, the column is kept once per line. For the input dataset, if any records are added to the data LOCF (Last Observation Carried forward) imputation is performed.
indLinPhiTol	the requested accuracy tolerance on exponential matrix.
indLinPhiM	the maximum size for the Krylov basis
indLinMatExpType	This is them matrix exponential type that is use for RxODE. Currently the following are supported: <ul style="list-style-type: none"> <li>• Al-Mohy Uses the exponential matrix method of Al-Mohy Higham (2009)</li> <li>• arma Use the exponential matrix from RcppArmadillo</li> <li>• expokit Use the exponential matrix from Roger B. Sidje (1998)</li> </ul>
indLinMatExpOrder	an integer, the order of approximation to be used, for the Al-Mohy and expokit values. The best value for this depends on machine precision (and slightly on the matrix). We use 6 as a default.
drop	Columns to drop from the output
idFactor	This boolean indicates if original ID values should be maintained. This changes the default sequentially ordered ID to a factor with the original ID values in the original dataset. By default this is enabled.
mxhnil	maximum number of messages printed (per problem) warning that $T + H = T$ on a step ( $H =$ step size). This must be positive to result in a non-default value. The default value is 0 (or infinite).
hmxi	inverse of the maximum absolute value of H to are used. $hmxi = 0.0$ is allowed and corresponds to an infinite $hmax1$ (default). $hmin$ and $hmxi$ may be changed at any time, but will not take effect until the next change of $H$ is considered. This option is only considered with <code>method="liblsoda"</code> .
warnIdSort	Warn if the ID is not present and RxODE assumes the order of the parameters/iCov are the same as the order of the parameters in the input dataset.
warnDrop	Warn if column(s) were supposed to be dropped, but were not present.
ssAtol	Steady state atol convergence factor. Can be a vector based on each state.
ssRtol	Steady state rtol convergence factor. Can be a vector based on each state.
safeZero	Use safe zero divide and log routines. By default this is turned on but you may turn it off if you wish.

sumType	Sum type to use for sum() in RxODE code blocks. pairwise uses the pairwise sum (fast, default) fsum uses Python's fsum function (most accurate) kahan uses Kahan correction neumaier uses Neumaier correction c uses no correction: default/native summing
prodType	Product to use for prod() in RxODE blocks long double converts to long double, performs the multiplication and then converts back. double uses the standard double scale for multiplication.
sensType	Sensitivity type for linCmt() model: advan Use the direct advan solutions autodiff Use the autodiff advan solutions forward Use forward difference solutions central Use central differences
linDiff	This gives the linear difference amount for all the types of linear compartment model parameters where sensitivities are not calculated. The named components of this numeric vector are: <ul style="list-style-type: none"> <li>• "lag" Central compartment lag</li> <li>• "f" Central compartment bioavailability</li> <li>• "rate" Central compartment modeled rate</li> <li>• "dur" Central compartment modeled duration</li> <li>• "lag2" Depot compartment lag</li> <li>• "f2" Depot compartment bioavailability</li> <li>• "rate2" Depot compartment modeled rate</li> <li>• "dur2" Depot compartment modeled duration</li> </ul>
linDiffCentral	This gives the which parameters use central differences for the linear compartment model parameters. The are the same components as linDiff
resample	A character vector of model variables to resample from the input dataset; This sampling is done with replacement. When NULL or FALSE no resampling is done. When TRUE resampling is done on all covariates in the input dataset
resampleID	boolean representing if the resampling should be done on an individual basis TRUE (ie. a whole patient is selected) or each covariate is resampled independent of the subject identifier FALSE. When resampleID=TRUE correlations of parameters are retained, where as when resampleID=FALSE ignores patient covariate correaltions. Hence the default is resampleID=TRUE.
maxwhile	represents the maximum times a while loop is evaluated before exiting. By default this is 100000
a	when using solve(), this is equivalent to the object argument. If you specify object later in the argument list it overwrites this parameter.
b	when using solve(), this is equivalent to the params argument. If you specify params as a named argument, this overwrites the output

## Details

The rest of the document focus on the different ODE solving methods, followed by the core solving method's options, RxODE event handling options, RxODE's numerical stability options, RxODE's output options, and finally internal RxODE options or compatibility options.

## Value

An "rxSolve" solve object that stores the solved value in a special data.frame or other type as determined by `returnType`. By default this has as many rows as there are sampled time points and as many columns as system variables (as defined by the ODEs and additional assignments in the RxODE model code). It also stores information about the call to allow dynamic updating of the solved object.

The operations for the object are similar to a data-frame, but expand the `$` and `[[""]]` access operators and assignment operators to resolve based on different parameter values, initial conditions, solver parameters, or events (by updating the time variable).

You can call the `eventTable()` methods on the solved object to update the event table and resolve the system of equations.

## Author(s)

Matthew Fidler, Melissa Hallow and Wenping Wang

## References

"New Scaling and Squaring Algorithm for the Matrix Exponential", by Awad H. Al-Mohy and Nicholas J. Higham, August 2009

Roger B. Sidje (1998). EXPOKIT: Software package for computing matrix exponentials. *ACM - Transactions on Mathematical Software* 24(1), 130-156.

Hindmarsh, A. C. *ODEPACK, A Systematized Collection of ODE Solvers*. Scientific Computing, R. S. Stepleman et al. (Eds.), North-Holland, Amsterdam, 1983, pp. 55-64.

Petzold, L. R. *Automatic Selection of Methods for Solving Stiff and Nonstiff Systems of Ordinary Differential Equations*. *Siam J. Sci. Stat. Comput.* 4 (1983), pp. 136-148.

Hairer, E., Norsett, S. P., and Wanner, G. *Solving ordinary differential equations I, nonstiff problems*. 2nd edition, Springer Series in Computational Mathematics, Springer-Verlag (1993).

## See Also

[RxODE\(\)](#)

---

rxStack	<i>Stack a solved object for things like ggplot</i>
---------	---

---

**Description**

Stack a solved object for things like ggplot

**Usage**

```
rxStack(Data, vars = NULL)
```

**Arguments**

Data	is a RxODE object to be stacked.
vars	Variables to include in stacked data; By default this is all the variables when vars is NULL.

**Value**

Stacked data with value and trt, where value is the values and trt is the state and lhs variables.

**Author(s)**

Matthew Fidler

---

rxState	<i>State variables</i>
---------	------------------------

---

**Description**

This returns the model's compartments or states.

**Usage**

```
rxState(obj = NULL, state = NULL)
```

**Arguments**

obj	RxODE family of objects
state	is a string indicating the state or compartment that you would like to lookup.

**Value**

If state is missing, return a character vector of all the states.

If state is a string, return the compartment number of the named state.

**Author(s)**

Matthew L.Fidler

**See Also**

[RxODE\(\)](#)

---

rxSumProdModel

*Recast model in terms of sum/prod*

---

**Description**

Recast model in terms of sum/prod

**Usage**

```
rxSumProdModel(model, expand = FALSE, sum = TRUE, prod = TRUE)
```

**Arguments**

model	RxODE model
expand	Boolean indicating if the expression is expanded.
sum	Use sum(...)
prod	Use prod(...)

**Value**

model string with prod(.) and sum(.) for all these operations.

**Author(s)**

Matthew L. Fidler



---

rxSupportedFuns	<i>Get list of supported functions</i>
-----------------	--

---

**Description**

Get list of supported functions

**Usage**

```
rxSupportedFuns()
```

**Value**

list of supported functions in RxODE

**Examples**

```
rxSupportedFuns()
```

---

rxSuppressMsg	<i>Respect suppress messages</i>
---------------	----------------------------------

---

**Description**

This turns on the silent REprintf in C when suppressMessages() is turned on. This makes the REprintf act like messages in R, they can be suppressed with suppressMessages()

**Usage**

```
rxSuppressMsg()
```

**Value**

Nothing

**Author(s)**

Matthew Fidler

**Examples**

```

# rxSupressMsg() is called with RxODE()

# Note the errors are output to the console

try(RxODE("d/dt(matt)=/3"),silent=TRUE)

# When using suppressMessages, the output is suppressed

suppressMessages(try(RxODE("d/dt(matt)=/3"),silent=TRUE))

# In RxODE, we use REprintf so that interrupted threads do not crash R
# if there is a user interrupt. This isn't captured by R's messages, but
# This interface allows the `suppressMessages()` to suppress the C printing
# as well

# If you want to suppress messages from RxODE in other packages, you can use
# this function

```

---

rxSymInvChol

*Get Omega<sup>-1</sup> and derivatives*


---

**Description**

Get Omega<sup>-1</sup> and derivatives

**Usage**

```

rxSymInvChol(
  invObjOrMatrix,
  theta = NULL,
  type = "cholOmegaInv",
  thetaNumber = 0L
)

```

**Arguments**

- |                |  |
|----------------|--|
| invObjOrMatrix | Object for inverse-type calculations. If this is a matrix, setup the object for inversion <code>rxSymInvCholCreate()</code> with the default arguments and return a reactive s3 object. Otherwise, use the inversion object to calculate the requested derivative/inverse. |
| theta          | Thetas to be used for calculation. If missing (NULL), a special s3 class is created and returned to access Omega <sup>-1</sup> objects as needed and cache them based on the theta that is used.   |
| type           | The type of object. Currently the following types are supported: <ul style="list-style-type: none"> <li>cholOmegaInv gives the Cholesky decomposition of the Omega Inverse matrix.</li> </ul>  |

- `omegaInv` gives the Omega Inverse matrix.
  - `d(omegaInv)` gives the  $d(\Omega^{-1})$  with respect to the theta parameter specified in `thetaNumber`.
  - `d(D)` gives the  $d(\text{diagonal}(\Omega^{-1}))$  with respect to the theta parameter specified in the `thetaNumber` parameter
- `thetaNumber` For types `d(omegaInv)` and `d(D)`, the theta number that the derivative is taken against. This must be positive from 1 to the number of thetas defining the Omega matrix.

**Value**

Matrix based on parameters or environment with all the matrixes calculated in variables `omega`, `omegaInv`, `dOmega`, `dOmegaInv`.

**Author(s)**

Matthew L. Fidler

---

`rxSyncOptions`      *Sync options with RxODE variables*

---

**Description**

Accessing RxODE options via `getOption` slows down solving. This allows the options to be synced with variables.

**Usage**

```
rxSyncOptions(setDefaults = c("none", "permissive", "strict"))
```

**Arguments**

- `setDefaults` This will setup RxODE's default solving options with the following options:
- "none" leave the options alone
  - "permissive" This is a permissive option set similar to R language specifications.
  - "strict" This is a strict option set similar to the original RxODE(). It requires semicolons at the end of lines and equals for assignment

**Value**

nothing; called for side effects

**Author(s)**

Matthew L. Fidler

---

rxSyntaxFunctions	<i>A list and description of Rode supported syntax functions</i>
-------------------	--

---

**Description**

A list and description of Rode supported syntax functions

**Usage**

```
rxSyntaxFunctions
```

**Format**

A data frame with 3 columns and 98 or more rows

**Function** Reserved function Name

**Description** Description of function

**Aliases** Function Aliases

---

rxt	<i>Simulate student t variable from threefry generator</i>
-----	--

---

**Description**

Care should be taken with this method not to encounter the birthday problem, described <https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/>. Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

**Usage**

```
rxt(df, n = 1L, ncores = 1L)
```

**Arguments**

df	degrees of freedom (> 0, maybe non-integer). df = Inf is allowed.
n	number of observations. If length(n) > 1, the length is taken to be the number required.
ncores	Number of cores for the simulation rxnorm simulates using the threefry sitmo generator; rxnormV uses the vander-corput generator

## Details

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to run all simulations in the RxODE environment once (therefore one seed or series of seeds for the whole simulation) or to pre-generate all random variables used for the simulation.

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

## Value

t-distribution random numbers

## Examples

```
## Use threefry engine

rxt(df=3, n=10) # with rxt you have to explicitly state n
rxt(df=3, n=10, ncores=2) # You can parallelize the simulation using openMP

rxt(4) ## The first argument is the df parameter

## This example uses `rxt` directly in the model

rx <- RxODE({
  a = rxt(3)
})

et <- et(1,id=1:2)

s <- rxSolve(rx,et)
```

---

rxTempDir

*Get the RxODE temporary directory*

---

## Description

Get the RxODE temporary directory

## Usage

```
rxTempDir()
```

**Value**

RxODE temporary directory.

---

rxTheme	<i>rxTheme is the RxODE theme for plots</i>
---------	---

---

**Description**

rxTheme is the RxODE theme for plots

**Usage**

```
rxTheme(  
  base_size = 11,  
  base_family = "",  
  base_line_size = base_size/22,  
  base_rect_size = base_size/22,  
  grid = TRUE  
)
```

**Arguments**

base_size	base font size, given in pts.
base_family	base font family
base_line_size	base size for line elements
base_rect_size	base size for rect elements
grid	a Boolean indicating if the grid is on (TRUE) or off (FALSE). This could also be a character indicating x or y.

**Value**

ggplot2 theme used in RxODE

---

rxToSE	<i>RxODE to symengine environment</i>
--------	---------------------------------------

---

**Description**

RxODE to symengine environment

**Usage**

```

rxToSE(x, envir = NULL, progress = FALSE, promoteLinSens = TRUE)

.rxToSE(x, envir = NULL, progress = FALSE)

rxFromSE(x, unknownDerivatives = c("forward", "central", "error"))

.rxFromSE(x)

```

**Arguments**

**x** expression

**envir** default is NULL; Environment to put symengine variables in.

**progress** shows progress bar if true.

**promoteLinSens** Promote solved linear compartment systems to sensitivity-based solutions.

**unknownDerivatives** When handling derivatives from unknown functions, the translator will translate into different types of numeric derivatives. The currently supported methods are:

- `forward` for forward differences
- `central` for central differences
- `error` for throwing an error for unknown derivatives

**Value**

An rxode symengine environment

**Author(s)**

Matthew L. Fidler

---

rxTrans

*Translate the model to C code if needed*

---

**Description**

This function translates the model to C code, if needed

**Usage**

```

rxTrans(
  model,
  modelPrefix = "",
  md5 = "",
  modName = NULL,

```

```

    modVars = FALSE,
    ...
)

## Default S3 method:
rxTrans(
  model,
  modelPrefix = "",
  md5 = "",
  modName = NULL,
  modVars = FALSE,
  ...
)

## S3 method for class 'character'
rxTrans(
  model,
  modelPrefix = "",
  md5 = "",
  modName = NULL,
  modVars = FALSE,
  ...
)

```

## Arguments

model	This is the ODE model specification. It can be: <ul style="list-style-type: none"> <li>• a string containing the set of ordinary differential equations (ODE) and other expressions defining the changes in the dynamic system.</li> <li>• a file name where the ODE system equation is contained</li> </ul> An ODE expression enclosed in $\{\}$ (see also the filename argument). For details, see the sections “Details” and RxODE Syntax below.
modelPrefix	Prefix of the model functions that will be compiled to make sure that multiple RxODE objects can coexist in the same R session.
md5	Is the md5 of the model before parsing, and is used to embed the md5 into DLL, and then provide for functions like <code>rxModelVars()</code> .
modName	a string to be used as the model name. This string is used for naming various aspects of the computations, including generating C symbol names, dynamic libraries, etc. Therefore, it is necessary that modName consists of simple ASCII alphanumeric characters starting with a letter.
modVars	returns the model variables instead of the named vector of translated properties.
...	Ignored parameters.

## Value

a named vector of translated model properties including what type of jacobian is specified, the C function prefixes, as well as the C functions names to be called through the compiled model.



**Author(s)**

Matthew L.Fidler

**See Also**[RxODE\(\)](#), [rxCompile\(\)](#).

---

`rxunif`*Simulate uniform variable from threefry generator*

---

**Description**

Care should be taken with this method not to encounter the birthday problem, described <https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/>. Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

**Usage**

```
rxunif(min = 0, max = 1, n = 1L, ncores = 1L)
```

**Arguments**

<code>min</code>	lower and upper limits of the distribution. Must be finite.
<code>max</code>	lower and upper limits of the distribution. Must be finite.
<code>n</code>	number of observations. If <code>length(n) &gt; 1</code> , the length is taken to be the number required.
<code>ncores</code>	Number of cores for the simulation <code>rxnorm</code> simulates using the threefry sitmo generator; <code>rxnormV</code> uses the vander-corput generator

**Details**

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to run all simulations in the RxODE environment once (therefore one seed or series of seeds for the whole simulation) or to pre-generate all random variables used for the simulation.

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

**Value**

uniform random numbers

## Examples

```
## Use threefry engine

rxunif(min=0, max=4, n=10) # with rxunif you have to explicitly state n
rxunif(min=0, max=4, n=10, ncores=2) # You can parallelize the simulation using openMP

rxunif()

## This example uses `rxunif` directly in the model

rx <- RxODE({
  a = rxunif(0,3)
})

et <- et(1,id=1:2)

s <- rxSolve(rx,et)
```

---

rxUnloadAll

*Unloads all RxODE compiled DLLs*

---

## Description

Unloads all RxODE compiled DLLs

## Usage

```
rxUnloadAll()
```

## Value

List of RxODE dlls still loaded  
boolean of if all RxODE dlls have been unloaded

## Examples

```
print(rxUnloadAll())
```

---

rxUse	<i>Use model object in your package</i>
-------	---

---

**Description**

Use model object in your package

**Usage**

```
rxUse(obj, overwrite = TRUE, compress = "bzip2", internal = FALSE)
```

**Arguments**

obj	model to save.
overwrite	By default, <code>use_data()</code> will not overwrite existing files. If you really want to do so, set this to TRUE.
compress	Choose the type of compression used by <code>save()</code> . Should be one of "gzip", "bzip2", or "xz".
internal	If this is run internally. By default this is FALSE

**Value**

Nothing; This is used for its side effects and shouldn't be called by a user

---

rxValidate	<i>Validate RxODE This allows easy validation/qualification of nlmixr by running the testing suite on your system.</i>
------------	--

---

**Description**

Validate RxODE This allows easy validation/qualification of nlmixr by running the testing suite on your system.

**Usage**

```
rxValidate(type = NULL)
```

```
rxTest(type = NULL)
```

**Arguments**

type	Type of test or filter of test type
------	-------------------------------------

**Value**

nothing

**Author(s)**

Matthew L. Fidler

rxweibull

*Simulate Weibull variable from threefry generator***Description**

Care should be taken with this method not to encounter the birthday problem, described <https://www.johndcook.com/blog/2016/01/29/random-number-generator-seed-mistakes/>. Since the sitmo threefry, this currently generates one random deviate from the uniform distribution to seed the engine threefry and then run the code.

**Usage**

```
rxweibull(shape, scale = 1, n = 1L, ncores = 1L)
```

**Arguments**

shape	shape and scale parameters, the latter defaulting to 1.
scale	shape and scale parameters, the latter defaulting to 1.
n	number of observations. If <code>length(n) &gt; 1</code> , the length is taken to be the number required.
ncores	Number of cores for the simulation rxnorm simulates using the threefry sitmo generator; rxnormV uses the vander-corput generator

**Details**

Therefore, a simple call to the random number generated followed by a second call to random number generated may have identical seeds. As the number of random number generator calls are increased the probability that the birthday problem will increase.

The key to avoid this problem is to run all simulations in the RxDDE environment once (therefore one seed or series of seeds for the whole simulation) or to pre-generate all random variables used for the simulation.

Also care should be made that the computer you will be running on can run the same number of cores as you are running so they can reproduce your results.

**Value**

Weibull random deviates

**Examples**

```
## Use threefry engine

rxweibull(shape=1, scale=4, n=10) # with rxweibull you have to explicitly state n
rxweibull(shape=1, scale=4, n=10, ncores=2) # You can parallelize the simulation using openMP

rxweibull(3)

## This example uses `rxweibull` directly in the model

rx <- RxODE({
  a = rxweibull(1,3)
})

et <- et(1,id=1:2)

s <- rxSolve(rx,et)
```

---

rxWinSetup

*Setup Windows components for RxODE*

---

**Description**

Setup Windows components for RxODE

**Usage**

```
rxWinSetup(rm.rtools = TRUE)
```

**Arguments**

rm.rtools      Remove the Rtools from the current path specs.

**Value**

nothing, used for its side effects

**Author(s)**

Matthew L. Fidler

---

stat_amt	<i>Dosing/Amt geom/stat</i>
----------	-----------------------------

---

### Description

This is a dosing geom that shows the vertical lines where a dose occurs

### Usage

```
stat_amt(
  mapping = NULL,
  data = NULL,
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)

geom_amt(
  mapping = NULL,
  data = NULL,
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

### Arguments

mapping	Set of aesthetic mappings created by <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
show.legend	logical. Should this layer be included in the legends? <code>NA</code> , the default, includes if any aesthetics are mapped. <code>FALSE</code> never includes, and <code>TRUE</code> always includes. It can also be a named logical vector to finely select the aesthetics to display.

`inherit.aes` If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.

... Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

### Details

Requires the following aesthetics:

- `x` representing the x values, usually time
- `amt` representing the dosing values; They are missing or zero when no dose is given

### Value

This returns a `stat_amt` in context of a `ggplot2` plot

---

<code>stat_cens</code>	<i>Censoring geom/stat</i>
------------------------	----------------------------

---

### Description

This is a censoring geom that shows the left or right censoring specified in the `nlmixr` input data-set or fit

### Usage

```
stat_cens(
  mapping = NULL,
  data = NULL,
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  width = 0.01,
  ...
)
```

```
geom_cens(
  mapping = NULL,
  data = NULL,
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  width = 0.01,
  ...
)
```

## Arguments

mapping	Set of aesthetic mappings created by <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
show.legend	logical. Should this layer be included in the legends? <code>NA</code> , the default, includes if any aesthetics are mapped. <code>FALSE</code> never includes, and <code>TRUE</code> always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If <code>FALSE</code> , overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. <code>borders()</code> .
width	represents the width (in <code>\</code> censoring box
...	Other arguments passed on to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> . They may also be parameters to the paired <code>geom/stat</code> .

## Details

Requires the following aesthetics:

- `x` Represents the independent variable, often the time scale
- `y` represents the dependent variable
- `CENS` for the censoring information; (-1 right censored, 0 no censoring or 1 left censoring)
- `LIMIT` which represents the corresponding limit ()

Will add boxes representing the areas of the fit that were censored.

## Value

This returns a `ggplot2` stat



---

```
summary.RxODE          Print expanded information about the RxODE object.
```

---

**Description**

This prints the expanded information about the RxODE object.

**Usage**

```
## S3 method for class 'RxODE'
summary(object, ...)
```

**Arguments**

```
object          RxODE object
...             Ignored parameters
```

**Value**

object is returned

**Author(s)**

Matthew L.Fidler

---

```
uppergamma          uppergamma: upper incomplete gamma function
```

---

**Description**

This is the tgamma from the boost library

**Usage**

```
uppergamma(a, z)
```

**Arguments**

```
a              The numeric 'a' parameter in the upper incomplete gamma
z              The numeric 'z' parameter in the upper incomplete gamma
```

**Details**

The uppergamma function is given by:

$$\text{uppergamma}(a, z) = \int_z^\infty t^{a-1} \cdot e^{-t} dt$$

**Value**

uppergamma results

**Author(s)**

Matthew L. Fidler

**Examples**

`uppergamma(1, 3)`

`uppergamma(1:3, 3)`

`uppergamma(1, 1:3)`

# Index

- \* **Nonlinear regression**
  - eventTable, 41
  - RxODE, 88
- \* **ODE models**
  - RxODE, 88
- \* **Ordinary differential equations**
  - RxODE, 88
- \* **PK/PD**
  - genShinyApp.template, 48
- \* **Pharmacodynamics (PD)**
  - eventTable, 41
  - RxODE, 88
- \* **Pharmacokinetics (PK)**
  - eventTable, 41
  - RxODE, 88
- \* **datasets**
  - rxReservedKeywords, 104
  - rxSyntaxFunctions, 132
- \* **data**
  - eventTable, 41
- \* **models**
  - eventTable, 41
  - RxODE, 88
- \* **nonlinear**
  - genShinyApp.template, 48
  - RxODE, 88
- \* **ordinary differential equations**
  - eventTable, 41
- \* **pharmacometrics**
  - genShinyApp.template, 48
- \* **simulation**
  - genShinyApp.template, 48
- + .solveRxDll (rxChain), 64
- .C(), 68
- .Call(), 68
- .rxFinalizeInner, 5
- .rxFinalizePred, 6
- .rxFromSE (rxToSE), 134
- .rxGenEBE, 6
- .rxGenEtaS, 7
- .rxGenFoce, 8
- .rxGenFocei, 9
- .rxGenFunState, 10
- .rxGenHdEta, 11
- .rxGenPred, 12
- .rxRmPrint, 13
- .rxToSE (rxToSE), 134
- .rxWinRtoolsPath, 13
- .rxWithOptions, 14
- .rxWithWd, 14
- .setWarnIdSort, 15
- [.rxEvid (rxEvid), 72
- [.rxRateDur (rxRateDur), 104
- [[.rxEvid (rxEvid), 72
- [[.rxRateDur (rxRateDur), 104
- add.dosing, 16, 17, 19, 29, 33, 36, 39
- add.dosing(), 41, 94
- add.sampling, 17, 19, 19, 29, 33, 36, 39
- add.sampling(), 41, 94
- aes(), 142, 144
- aes\_(), 142, 144
- as.character.rxEvid (rxEvid), 72
- as.character.rxRateDur (rxRateDur), 104
- as.data.table.rxEt, 21
- as.et, 22
- as.rxEvid (rxEvid), 72
- as.rxRateDur (rxRateDur), 104
- as\_tibble.rxEt, 22
- borders(), 143, 144
- c.rxEvid (rxEvid), 72
- c.rxRateDur (rxEvid), 72
- coef.RxODE, 23
- cvPost, 23
- environment, 27
- et, 17, 19, 26, 29, 33, 36, 39

- et(), [16](#), [42](#), [90](#), [94](#)
- etExpand, [31](#)
- etRbind, [17](#), [19](#), [29](#), [32](#), [33](#), [36](#), [39](#)
- etRep, [17](#), [19](#), [29](#), [33](#), [35](#), [36](#), [39](#)
- etSeq, [38](#)
- eventTable, [17](#), [19](#), [29](#), [33](#), [36](#), [39](#), [41](#)
- eventTable(), [49](#), [90](#), [94](#), [118](#), [126](#)
- expit (logit), [52](#)
  
- forderForceBase, [43](#)
- format.rxEvid (rxEvid), [72](#)
- format.rxRateDur (rxRateDur), [72](#)
- fortify(), [142](#), [144](#)
  
- gammap, [43](#)
- gammapDer, [44](#)
- gammapInv, [45](#)
- gammapInva (gammapInv), [45](#)
- gammaq, [46](#)
- gammaqInv, [47](#)
- gammaqInva (gammaqInv), [47](#)
- genShinyApp.template, [48](#)
- geom\_amt (stat\_amt), [142](#)
- geom\_cens (stat\_cens), [143](#)
- getRxThreads, [49](#)
- ggplot(), [142](#), [144](#)
- guide\_none, [50](#)
  
- invWR1d, [51](#)
- is.rxEt, [51](#)
- is.rxSolve, [52](#)
  
- labs(), [50](#)
- layer(), [143](#), [144](#)
- logit, [52](#)
- logitNormInfo (logit), [52](#)
- lowergamma, [54](#)
  
- order(), [43](#)
  
- phi, [55](#)
- pillar\_shaft.rxEvid (rxEvid), [72](#)
- pillar\_shaft.rxRateDur (rxRateDur), [104](#)
- predict.rxEt (rxSolve), [115](#)
- predict.RxODE (rxSolve), [115](#)
- predict.rxParams (rxSolve), [115](#)
- predict.rxSolve (rxSolve), [115](#)
- print.rxEvid (rxEvid), [72](#)
- print.RxODE, [55](#)
- probit, [56](#)
- probitInv (probit), [56](#)
- probitNormInfo (logit), [52](#)
  
- rbind.rxEt (etRbind), [32](#)
- rep.rxEt (etRep), [35](#)
- rinvchisq, [57](#)
- rLKJ1, [57](#)
- rLKJ1(), [24](#)
- rxAddReturn, [58](#)
- rxAllowUnload, [58](#)
- rxAssignPtr, [59](#)
- rxbeta, [60](#)
- rxbinom, [61](#)
- rxcauchy, [62](#)
- rxChain, [64](#)
- rxchisq, [64](#)
- rxClean, [66](#)
- rxCompile, [66](#)
- rxCompile(), [137](#)
- rxControl (rxSolve), [115](#)
- rxCores (getRxThreads), [49](#)
- rxCreateCache, [68](#)
- rxD, [69](#)
- rxD(), [76](#)
- rxDelete, [69](#)
- rxDerived, [70](#)
- rxDfdy, [72](#)
- rxEvid, [72](#)
- rxexp, [74](#)
- rxf, [75](#)
- rxFromSE (rxToSE), [134](#)
- rxFun, [76](#)
- rxgamma, [78](#)
- rxGenSaem, [79](#)
- rxgeom, [80](#)
- rxGetLin, [81](#)
- rxGetRxODE, [82](#)
- rxHtml, [83](#)
- rxIndLinState, [83](#)
- rxIndLinStrategy, [84](#)
- rxInv, [84](#)
- rxIsCurrent, [85](#)
- rxLhs, [85](#)
- rxLock, [86](#)
- rxModelVars(), [136](#)
- rxNorm, [86](#)
- rxnorm, [87](#)
- rxnormV (rxnorm), [87](#)
- RxODE, [17](#), [19](#), [29](#), [33](#), [36](#), [39](#), [85](#), [88](#)

RxODE(), 42, 48, 49, 68, 126, 128, 137  
 rxOptExpr, 95  
 rxParam (rxParams), 96  
 rxParams, 96  
 rxPkg, 98  
 rxpois, 99  
 rxPp, 100  
 rxProgress, 102  
 rxProgressAbort (rxProgress), 102  
 rxProgressStop (rxProgress), 102  
 rxRandNV, 103  
 rxRateDur, 104  
 rxReservedKeywords, 104  
 rxRmFun (rxFun), 76  
 rxRmvn, 105  
 rxS, 107  
 rxSetIni0, 108  
 rxSetProd, 109  
 rxSetProgressBar, 109  
 rxSetSum, 110  
 rxShiny, 110  
 rxSimThetaOmega, 111  
 rxSolve, 115  
 rxSolve(), 48  
 rxStack, 127  
 rxState, 127  
 rxSumProdModel, 128  
 rxSupportedFuns, 129  
 rxSuppressMsg, 129  
 rxSymInvChol, 130  
 rxSymInvCholCreate(), 130  
 rxSyncOptions, 131  
 rxSyntaxFunctions, 132  
 rxt, 132  
 rxTempDir, 133  
 rxTest (rxValidate), 139  
 rxTheme, 134  
 rxTick (rxProgress), 102  
 rxToSE, 134  
 rxTrans, 135  
 rxTrans(), 68  
 rxunif, 137  
 rxUnloadAll, 138  
 rxUnlock (rxLock), 86  
 rxUse, 139  
 rxValidate, 139  
 rxweibull, 140  
 rxWinSetup, 141  
 save(), 139  
 seq.rxEt (etSeq), 38  
 setRxThreads (getRxThreads), 49  
 setRxThreads(), 119  
 simulate.RxODE (rxSolve), 115  
 simulate.rxParams (rxSolve), 115  
 simulate.rxSolve (rxSolve), 115  
 solve.rxEt (rxSolve), 115  
 solve.RxODE (rxSolve), 115  
 solve.rxParams (rxSolve), 115  
 solve.rxSolve (rxSolve), 115  
 stat\_amt, 142  
 stat\_cens, 143  
 summary.RxODE, 145  
 sys.call, 27  
 type\_sum.rxEvid (rxEvid), 72  
 type\_sum.rxRateDur (rxRateDur), 104  
 update.rxSolve (rxSolve), 115  
 uppergamma, 145  
 use\_description(), 99  
 waiver(), 50  
 write.template.server  
     (genShinyApp.template), 48  
 write.template.ui  
     (genShinyApp.template), 48  
 write.template.ui(), 48