Package 'NMdata'

February 21, 2024

Type Package

Title Preparation, Checking and Post-Processing Data for PK/PD Modeling

Version 0.1.5

Maintainer Philip Delff <philip@delff.dk>

Description Efficient tools for preparation, checking and post-processing of data in PK/PD (pharmacokinetics/pharmacodynamics) modeling, with focus on use of Nonmem. Attention is paid to ensure consistency, traceability, and Nonmem compatibility of Data. Rigorously checks final Nonmem datasets. Implemented in 'data.table', but easily integrated with 'base' and 'tidyverse'.

License MIT + file LICENSE

RoxygenNote 7.2.3

Depends R (>= 3.1.0)

Imports data.table, fst

Suggests testthat, knitr, formatR, mime, rmarkdown, ggplot2, tibble, covr, htmltools, spelling

Encoding UTF-8

URL https://philipdelff.github.io/NMdata/

BugReports https://github.com/philipdelff/NMdata/issues

Language en-US

NeedsCompilation no

Author Philip Delff [aut, cre]

Repository CRAN

Date/Publication 2024-02-21 08:10:02 UTC

R topics documented:

addTAPD *Add time since previous dose to data, time of previous dose, most recent dose amount, cumulative number of doses, and cumulative dose amount.*

Description

For now, doses have to be in data as EVID=1 and/or EVID=4 records. They can be in the format of one row per dose or repeated dosing notation using ADDL and II.

Usage

```
addTAPD(
  data,
  col.time = "TIME",
  col.evid = "EVID",
  col.amt = "AMT",
  col.tpdos = "TPDOS",
  col.tapd = "TAPD",
  col.pdosamt = "PDOSAMT",
  col.doscuma = "DOSCUMA",
  col.doscumn = "DOSCUMN",
  prefix.cols,
  suffix.cols,
  subset.dos,
  subset.is.complete,
  order.evid = c(3, 0, 2, 4, 1),
  by = "ID",
  SDOS = 1,
  as.fun,
  col.ndoses
)
```

Arguments

data	The data set to add the variables to.
col.time	Name of time column (created by addTAPD). Default it TIME.
col.evid	The name of the event ID column. This must exist in data. Default is EVID.
col.amt	col.evid The name of the dose amount column. This must exist in data. Default is AMT.
col.tpdos	Name of the time of previous dose column (created by addTAPD). Default is TPDOS. Set to NULL to not create this column.
col.tapd	Name of the time of previous dose column (created by addTAPD). Default is TAPD. Set to NULL to not create this column.
col.pdosamt	The name of the column to be created holding the previous dose amount. Set to NULL to not create this column.

col.doscuma — The name of the column to be created holding the cumulative dose amount. Set to NULL to not create this column.

col.doscumn — The name of the column (created by addTAPD) that holds the cumulative number of doses administered to the subject. Set to NULL to not create this column.

prefix.cols — String to be prepended to all generated column names, that is each of col.tpdos, col.tapd, col.ndoses, col.pdosamt, col.doscuma that are not NULL.

suffix.cols — String to be appended to all generated column names, that is each of col.tpdos, col.tapd, col.ndoses, col.pdosamt, col.doscuma that are not NULL.

subset.dos — A string that will be evaluated as a custom expression to identify relevant events. See subset.is.complete as well.

subset.is.complete — Only used in combination with non-missing subset.dos. By default, subset.dos is used in addition to the impact of col.evid (must be 1 or 4) and col.amt (greater than zero). If subset.is.complete=TRUE, subset.dos is used alone, and col.evid and col.amt are completely ignored. This is typically useful if the events are not doses but other events that are not expressed as a typical dose combination of EVID and AMT columns.

order.evid — Order of events. This will only matter if there are simultaneous events of different event types within subjects. Typically if using nominal time, it may be important to specify whether samples at dosing times are pre-dose samples. The default is c(3,0,4,1,2) - i.e. samples and simulations are pre-dose. See details.

by — Columns to do calculations within. Default is ID.

SDOS — Scaling value for columns related to dose amount, relative to AMT values. col.pdosamt and col.doscuma are affected and will be derived as AMT/SDOSE.

as.fun — The default is to return data as a data.frame. Pass a function (say tibble::as_tibble) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.

col.ndoses — Deprecated. Use col.doscumn instead.

Details

addTAPD does not require the data to be ordered, and it will not order it. This means you can run addTAPD before ordering data (which may be one of the final steps) in data set preparation. The argument called order.evid is important because of this. If a dosing event and a sample occur at the same time, when which dose was the previous for that sample? Default is to assume the sample is a pre-dose sample, and hence output will be calculated in relation to the dose before. If no dose event is found before, NA's will be assigned.

Value

A data.frame with additional columns

See Also

Other DataCreate: NMorderColumns(), NMstamp(), NMwriteData(), findCovs(), findVars(), flagsAssign(), flagsCount(), mergeCheck(), tmpcol()

cc *Create character vectors without quotation marks*

Description

When creating character vectors with several elements, it becomes a lot of quotes to type. cc provides a simple way to skip the quotes - but only for simple strings.

Usage

```
cc(...)
```

Arguments

...	The unquoted names that will become character values in the returned vector.

Details

Don't use cc with any special characters - only alphanumerics and no spaces supported. Also, remember that numerics are converted using as.character. Eg, this means that leading zeros are dropped.

Value

A character vector

See Also

cl

Examples

```
cc(a,b,`a b`)
cc(a,b,"a b")
## be careful with spaces and special characters
cc( d)
cc(" d")
cc()
## Numerics are converted using as.character
cc(001,1,13e3)
```

cl — *Define a vector with factor levels in the same order as occurring in the vector.*

Description

This is a shortcut for creating factors with levels as the order of appearance of the specified levels.

Usage

```
cl(...)
```

Arguments

...	unique elements or vectors with unique elements

Value

A factor (vector)

See Also

cc

Examples

```
factor("b","a")
cl("b","a")
x <- c("b","a")
factor(x)
cl(x)
```

colLabels — *Extract column labels as defined in SAS*

Description

Extract column labels as defined in SAS

Usage

```
colLabels(...)
```

Arguments

...	See ‘?compareCols‘

compareCols *Compare elements in lists with aim of combining*

Description

Useful interactive tool when merging or binding objects together. It lists the names of elements that differ in presence or class across multiple datasets. Before running rbind, you may want to check the compatibility of the data.

Usage

```
compareCols(
  ...,
  list.data,
  keep.names = TRUE,
  test.equal = FALSE,
  diff.only = TRUE,
  cols.wanted,
  fun.class = base::class,
  quiet,
  as.fun,
  keepNames,
  testEqual
)
```

Arguments

`...`	objects which element names to compare
`list.data`	As alternative to ..., you can supply the data sets in a list here.
`keep.names`	If TRUE, the original dataset names are used in reported table. If not, generic x1, x2,... are used. The latter may be preferred for readability.
`test.equal`	Do you just want a TRUE/FALSE to whether the names of the two objects are the same? Default is FALSE which means to return an overview for interactive use. You might want to use TRUE in programming. However, notice that this check may be overly rigorous. Many classes are compatible enough (say numeric and integer), and compareCols doesn't take this into account.
`diff.only`	If TRUE, don't report columns where no difference found. Default is TRUE if number of data sets supplied is greater than one. If only one data set is supplied, the full list of columns is shown by default.
`cols.wanted`	Columns of special interest. These will always be included in overview and indicated by a prepended * to the column names. This argument is often useful when you start by defining a set of columns that you want to end up with by combining a number of data sets.

fun.class	the function that will be run on each column to check for differences. base::class is default. Notice that the alternative base::typeof is different in certain ways. For instance, typeof will not report a difference on numeric vs difftime. You could basically submit any function that takes a vector and returns a single value.
quiet	The default is to give some information along the way on what data is found. But consider setting this to TRUE for non-interactive use. Default can be configured using NMdataConf.
as.fun	A function that will be run on the result before returning. If first input data set is a data.table, the default is to return a data.table, if not the default is to return a data.frame. Use whatever to get what fits in with your workflow. Default can be configured with NMdataConf.
keepNames	Deprecated. Use keep.names instead.
testEqual	Deprecated. Use test.equal instead.

Details

technically, this function compares classes of elements in lists. However, in relation to NMdata, this will most of the time be columns in data.frames.

Despite the name of the argument fun.class, it can be any function to be evaluated on each element in '...'. See examples for how to extract SAS labels on an object read with 'read_sas' from the 'haven' package.

Value

A data.frame with an overview of elements and their classes of objects in ... Class as defined by as.fun.

See Also

Other DataWrangling: dims(), listMissings()

Examples

```
## get SAS labels from objects read with haven::read_sas
## Not run:
compareCols(...,fun.class=function(x)attributes(x)$label)

## End(Not run)
```

dims *Get dimensions of multiple objects*

Description

Get dimensions of multiple objects

Usage

```
dims(..., list.data, keep.names = TRUE, as.fun = NULL, keepNames)
```

Arguments

...	data sets
list.data	As alternative to ..., you can supply the data sets in a list here.
keep.names	If TRUE, the original dataset names are used in reported table. If not, generic x1, x2,... are used. The latter may be preferred for readability in some cases.
as.fun	A function that will be run on the result before returning. If first input data set is a data.table, the default is to return a data.table, if not the default is to return a data.frame. Use whatever to get what fits in with your workflow. Default can be configured with NMdataConf.
keepNames	Deprecated. Use keep.names instead.

Value

A data.frame with dimensions of objects in ... Actual class defined by as.fun.

See Also

Other DataWrangling: `compareCols()`, `listMissings()`

editCharCols *Replace strings in character character columns of a data set*

Description

Replace strings in character character columns of a data set

Usage

```
editCharCols(data, pattern, replacement, as.fun, ...)
```

Arguments

data	The data set to edit.
pattern	Pattern to search for in character columns. Passed to ‘gsub()‘. By default, ‘gsub()‘ works with regular expressions. See ... for how to disable this if you want to replace a specific string.
replacement	pattern or string to replace with. Passed to ‘gsub()‘.
as.fun	The default is to return data as a data.frame. Pass a function (say tibble::as_tibble) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.
...	Additional arguments passed to ‘gsub()‘. Especially, notice fixed=TRUE will disable interpretation of ‘pattern‘ and ‘replace‘ as regular expressions.

Examples

```
### remove commas from character columns
dat <- data.frame(A=1:3,text=cc(a,"a,d","g"))
editCharCols(dat,pattern=",","")
### factors are not edited but result in an error
## Not run:
dat <- data.frame(A=1:3,text=cc(a,"a,d",g),fac=cl("a","a,d","g"))
editCharCols(dat,pattern=",","")

## End(Not run)
```

---

egdt *Expand grid of data.tables*

Description

Expand grid of data.tables

Usage

```
egdt(dt1, dt2, quiet)
```

Arguments

dt1	a data.table.
dt2	another data.table.
quiet	The default is to give some information along the way on what data is found. But consider setting this to TRUE for non-interactive use. Default can be configured using NMdataConf.

Details

Merging works mostly similarly for data.table and data.table. However, for data.table the merge must be done by one or more columns. This means that the convenient way to expand all combinations of all rows in two data.frames is not available for data.tables. This functions provides that functionality. It always returns data.tables.

Value

a data.table that expands combinations of rows in dt1 and dt2.

Examples

```
df1 <- data.frame(a=1:2,b=3:4)
df2 <- data.frame(c=5:6,d=7:8)
merge(df1,df2)
library(data.table)
## This is not possible
## Not run:
merge(as.data.table(df1),as.data.table(df2),allow.cartesian=TRUE)

## End(Not run)
## Use egdt instead
egdt(as.data.table(df1),as.data.table(df2),quiet=TRUE)
## Dimensions are conveniently listed for interactive use
res <- egdt(as.data.table(df1),as.data.table(df2))
```

---

findCovs *Extract columns that vary within values of other columns*

Description

This function provides an automated method to extract covariate-like columns. The user decides which columns these variables cannot vary within. So if you have repeated measures for each ID, this function can find the columns that are constant within ID and their unique values for each ID. Or, you can provide a combination of id.cols, say ID and STUDY, and get variables that do not vary within unique combinations of these.

Usage

```
findCovs(data, by = NULL, cols.id, as.fun = NULL)
```

Arguments

data	data.frame in which to look for covariates
by	covariates will be searched for in combinations of values in these columns. Often by will be either empty or ID. But it can also be both say c("ID","DRUG") or c("ID","TRT").
cols.id	Deprecated. Use by instead.

as.fun — The default is to return a data.table if data is a data.table and return a data.frame in all other cases. Pass a function in as.fun to convert to something else. If data is not a data.table, the default can be configured using NMdataConf.

Value

a data set with one observation per combination of values of variables listed in by.

See Also

Other DataCreate: NMorderColumns(), NMstamp(), NMwriteData(), addTAPD(), findVars(), flagsAssign(), flagsCount(), mergeCheck(), tmpcol()

Examples

```
dt1=data.frame(ID=c(1,1,2,2),
               OCC=c(1,2,1,2),
               ## ID level
               eta1=c(1,1,3,3),
               ## occasion level
               eta2=c(1,3,1,5),
               ## not used
               eta3=0
               )
## model level
findCovs(dt1)
## ID level
findCovs(dt1,"ID")
## acual ID level
findVars(findCovs(dt1,"ID"))
## occasion level
findCovs(findVars(dt1,"ID"),c("ID","OCC"))
## Based on a "real data example"
## Not run:
dat <- NMscanData(system.file("examples/nonmem/xgxr001.lst", package = "NMdata"))
findCovs(dat,by="ID")
### Without an ID column we get non-varying columns
findCovs(dat)

## End(Not run)
```

---

findVars — *Extract columns that vary within values of other columns in a data.frame*

Description

If you want to look at the variability of a number of columns and you want to disregard those that are constant. Like for findCovs, by can be of arbitrary length.

Usage

```
findVars(data, by = NULL, cols.id, as.fun = NULL)
```

Arguments

data	data.frame in which to look for covariates
by	optional covariates will be searched for in combinations of values in these columns. Often by will be either empty or ID. But it can also be both say c("ID","DRUG") or c("ID","TRT").
cols.id	Deprecated. Use by instead.
as.fun	The default is to return a data.table if data is a data.table and return a data.frame in all other cases. Pass a function in as.fun to convert to something else. If data is not a data.table, the default can be configured using NMdataConf.

Details

Use this to exclude columns that are constant within by. If by=ID, this could be to get only time-varying covariates.

Value

a data set with as many rows as in data.

See Also

Other DataCreate: NMorderColumns(), NMstamp(), NMwriteData(), addTAPD(), findCovs(), flagsAssign(), flagsCount(), mergeCheck(), tmpcol()

Examples

```
dt1 <- data.frame(ID=c(1,1,2,2),
                  OCC=c(1,2,1,2),
                ## ID level
                  eta1=c(1,1,3,3),
                ## occasion level
                  eta2=c(1,3,1,5),
                ## not used
                  eta3=0
                )
## model level
findCovs(dt1)
## ID level
findCovs(dt1,"ID")
## acual ID level
findVars(findCovs(dt1,"ID"))
## occasion level
findCovs(findVars(dt1,"ID"),c("ID","OCC"))
```

`flagsAssign` *Assign exclusion flags to a dataset based on specified table*

Description

The aim with this function is to take a (say PK) dataset and a pre-specified table of flags, assign the flags automatically.

Usage

```
flagsAssign(
  data,
  tab.flags,
  subset.data,
  col.flagn,
  col.flagc,
  flags.increasing = FALSE,
  grp.incomp = "EVID",
  flagc.0 = "Analysis set",
  as.fun = NULL
)
```

Arguments

`data`	The dataset to assign flags to.
`tab.flags`	A data.frame containing at least these named columns: FLAG, flag, condition. Condition is disregarded for FLAG==0. FLAG must be numeric and non-negative, flag and condition are characters.
`subset.data`	An optional string that provides a subset of data to assign flags to. A common example is subset=\"EVID==0\" to only assign to observations. Numerical and character flags will be missing in rows that are not matched by this subset.
`col.flagn`	The name of the column containing the numerical flag values in tab.flags. This will be added to data. Default value is FLAG and can be configured using NMdataConf.
`col.flagc`	The name of the column containing the character flag values in tab.flags. This will be added to data. Default value is flag and can be configured using NMdataConf.
`flags.increasing`	The flags are applied by either decreasing (default) or increasing value of col.flagn. Decreasing order means that conditions associated with higher values of col.flagn will be evaluated first. By using decreasing order, you can easily adjust the Nonmem IGNORE statement from IGNORE(FLAG.NE.0) to say IGNORE(FLAG.GT.10) if BLQ's have FLAG=10, and you decide to include these in the analysis.
`grp.incomp`	Column(s) that distinct incompatible subsets of data. Default is "EVID" meaning that if different values of EVID are found in data, the function will return an error. This is a safeguard not to mix data unintentionally when counting flags.

`flagc.0` The character flag to assign to rows that are not matched by exclusion conditions (numerical flag 0).

`as.fun` The default is to return data.tables if input data is a data.table, and return a data.frame for all other input classes. Pass a function in as.fun to convert to something else. If return.all=FALSE, this is applied to data and tab.flags independently.

Details

dt.flags must contain a column with numerical exclusion flags, one with character exclusion flags, and one with a expressions to evaluate for whether to apply the exclusion flag. The flags are applied sequentially, by increasing value of the numerical exclusion flag.

Value

The dataset with flags added. Class as defined by as.fun. See parameter flags.return as well.

See Also

Other DataCreate: `NMorderColumns()`, `NMstamp()`, `NMwriteData()`, `addTAPD()`, `findCovs()`, `findVars()`, `flagsCount()`, `mergeCheck()`, `tmpcol()`

Examples

```
## Not run:
pk <- readRDS(file=system.file("examples/data/xgxr2.rds",package="NMdata"))
dt.flags <- data.frame(
       flagn=10,
       flagc="Below LLOQ",
       condition=c("BLQ==1")
)
pk <- flagsAssign(pk,dt.flags,subset.data="EVID==0",col.flagn="flagn",col.flagc="flagc")
pk <- flagsAssign(pk,subset.data="EVID==1",flagc.0="Dosing",
        col.flagn="flagn",col.flagc="flagc")
unique(pk[,c("EVID","flagn","flagc","BLQ")])
flagsCount(pk[EVID==0],dt.flags,col.flagn="flagn",col.flagc="flagc")

## End(Not run)
```

---

flagsCount *Create an overview of number of retained and discarded datapoints.*

Description

Generate an overview of number of observations disregarded due to different reasons. And how many are left after each exclusion flag.

Usage

```
flagsCount(
  data,
  tab.flags,
  file,
  col.id = "ID",
  col.flagn,
  col.flagc,
  by = NULL,
  flags.increasing = FALSE,
  flagc.0 = "Analysis set",
  name.all.data = "All available data",
  grp.incomp = "EVID",
  save = TRUE,
  as.fun = NULL
)
```

Arguments

data	The dataset including both FLAG and flag columns.
tab.flags	A data.frame containing at least these named columns: FLAG, flag, condition. Condition is disregarded for FLAG==0.
file	A file to write the table of flag counts to. Will probably be removed and put in a separate function.
col.id	The name of the subject ID column. Default is "ID".
col.flagn	The name of the column containing the numerical flag values in tab.flags. This will be added to data. Use the same as when flagsAssign was called (if that was used). Default value is FLAG and can be configured using NMdataConf.
col.flagc	The name of the column containing the character flag values in data and tab.flags. Use the same as when flagsAssign was called (if that was used). Default value is flag and can be configured using NMdataConf.
by	An optional column to group the counting by. This could be "STUDY", "DRUG", "EVID", or a combination of multiple columns.
flags.increasing	The flags are applied by either decreasing (default) or increasing value of col.flagn. By using decreasing order, you can easily adjust the Nonmem IGNORE statement from IGNORE(FLAG.NE.0) to say IGNORE(FLAG.GT.10) if BLQ's have FLAG=10, and you decide to include these in the analysis.
flagc.0	The character flag to assign to rows that are not matched by exclusion conditions (numerical flag 0).
name.all.data	What to call the total set of data before applying exclusion flags. Default is "All available data".
grp.incomp	Column(s) that distinct incompatible subsets of data. Default is "EVID" meaning that if different values of EVID are found in data, the function will return an error. This is a safeguard not to mix data unintentionally when counting flags.

save	Save file? Default is TRUE, meaning that a file will be written if file argument is supplied.
as.fun	The default is to return a data.table if input data is a data.table, and return a data.frame for all other input classes. Pass a function in as.fun to convert to something else. If data is not a data.table, default can be configured using NMdataConf.

Details

This function is used to count flags as assigned by the flagsAssign function.

Notice that the character flags reported in the output table are taken from tab.flags. The data column named by the value of col.flagc (default is flag) is not used.

In the returned table, N.discarded is the difference in number of subjects since previous step. If two is reported, it can mean that the remaining one observation of these two subjects are discarded due to this flag. The majority of the samples can have been discarded by earlier flags.

Value

A summary table with number of discarded and retained subjects and observations when applying each condition in the flag table. "discarded" means that the reduction of number of observations and subjects resulting from the flag, "retained" means the numbers that are left after application of the flag. The default is "both" which will report both. Class as defined by as.fun.

See Also

Other DataCreate: NMorderColumns(), NMstamp(), NMwriteData(), addTAPD(), findCovs(), findVars(), flagsAssign(), mergeCheck(), tmpcol()

Examples

```
## Not run:
pk <- readRDS(file=system.file("examples/data/xgxr2.rds",package="NMdata"))
dt.flags <- data.frame(
       flagn=10,
       flagc="Below LLOQ",
       condition=c("BLQ==1")
)
pk <- flagsAssign(pk,dt.flags,subset.data="EVID==0",col.flagn="flagn",col.flagc="flagc")
pk <- flagsAssign(pk,subset.data="EVID==1",flagc.0="Dosing",
        col.flagn="flagn",col.flagc="flagc")
unique(pk[,c("EVID","flagn","flagc","BLQ")])
flagsCount(pk[EVID==0],dt.flags,col.flagn="flagn",col.flagc="flagc")

## End(Not run)
```

fnAppend *paste something before file name extension.*

Description

Append a file name like file.mod to file_1.mod or file_pk.mod. If it's a number, we can pad some zeros if wanted. The separator (default is underscore) can be modified.

Usage

```
fnAppend(fn, x, pad0 = 0, sep = "_")
```

Arguments

fn	The file name to modify
x	A character string or a numeric to add to the file name
pad0	In case x is numeric, a number of zeros to pad before the appended number. This is useful if you are generating say more than 10 files, and your counter will be 01, 02,.., 10,... and not 1, 2,...,10,...
sep	The separator between the existing file name (until extension) and the addition.

Value

A character (vector)

fnExtension *Change file name extension*

Description

Very simple but often applicable function to retrieve or change the file name extension (from say file.lst to file.mod)

Usage

```
fnExtension(fn, ext)
```

Arguments

fn	file name. Often ending in an extension after a period but the extension is not needed.
ext	new file name extension. If omitted or NULL, the extension of fn is returned.

Value

A text string

Examples

```
fnExtension("file.lst",".mod")
fnExtension("file.lst","mod")
fnExtension("file.lst","..mod")
fnExtension("file.lst",cc(.mod,xml))
fnExtension(cc(file1.lst,file2.lst),cc(.xml))
fnExtension(cc(file1.lst,file2.lst),cc(.xml,.cov))
fnExtension("file.lst","")
fnExtension("file.lst")
```

---

is.NMdata *Check if an object is 'NMdata'*

Description

Check if an object is 'NMdata'

Usage

```
is.NMdata(x)
```

Arguments

x	Any object

Value

logical if x is an 'NMdata' object

---

listMissings *List rows with missing values across multiple columns*

Description

Missing can be NA and for character variables it can be certain strings too. This function is experimental and design may change in future releases.

Usage

```
listMissings(data, cols, by, na.strings = c("", "."), quiet = FALSE, as.fun)
```

Arguments

data	The data to look into.
cols	The columns to look for missings in.
by	If supplied, we are keeping track of the missings within the values of the by columns. In summary, by is included too.
na.strings	Strings that should be interpreted as missing. All spaces will be removed before we compare to na.strings. The default is c("",".") so say " . " is a missing by default.
quiet	Keep quiet? Default is not to.
as.fun	A function that will be run on the result before returning. If first input data set is a data.table, the default is to return a data.table, if not the default is to return a data.frame. Use whatever to get what fits in with your workflow. Default can be configured with NMdataConf.

Value

Invisibly, a data.frame including all findings

See Also

Other DataWrangling: `compareCols()`, `dims()`

---

mergeCheck *Merge, order, and check resulting rows and columns.*

Description

Stop checking that the number of rows is unchanged after a merge - mergeCheck checks what you really want - i.e. x is extended with columns from y while all rows in x are retained, and no new rows are created (plus some more checks). mergeCheck is not a merge implementation - it is a useful merge wrapper. The advantage over using much more flexible merge or join function lies in the fully automated checking that the results are consistent with the simple merge described above.

Usage

```
mergeCheck(
  x,
  y,
  by,
  by.x,
  by.y,
  fun.commoncols = base::warning,
  ncols.expect,
  track.msg = FALSE,
  quiet,
```

```
  df1,
  df2,
  fun.na.by = base::stop,
  as.fun,
  ...
)
```

Arguments

x	A data.frame with the number of rows must should be obtained from the merge. The resulting data.frame will be ordered like x.
y	A data.frame that will be merged onto x.
by	The column(s) to merge by. Character string (vector). by or by.x and by.y must be supplied.
by.x	If the columns to merge by in x and y are named differently. by or by.x and by.y must be supplied.
by.y	If the columns to merge by in x and y are named differently. by or by.x and by.y must be supplied.
fun.commoncols	If common columns are found in x and y, and they are not used in by, this will create columns named like col.x and col.y in result (see ?merge). Often, this is a mistake, and the default is to throw a warning if this happens. If using mergeCheck in a function, you may want to make sure this is not happening and use fun.commoncols=stop. If you want nothing to happen, you can do fun.commoncols=NULL.
ncols.expect	If you want to include a check of the number of columns being added to the dimensions of x. So if ncols.expect=1, the resulting data must have exactly one column more than x - if not, an error will be returned.
track.msg	If using mergeCheck inside other functions, it can be useful to use track.msg=TRUE. This will add information to messages/warnings/errors that they came from mergCheck.
quiet	If FALSE, the names of the added columns are reported. Default value controlled by NMdataConf.
df1	Deprecated. Use x.
df2	Deprecated. Use y.
fun.na.by	If NA's are found in (matched) by columns in both x and why, what should we do? This could be OK, but in many cases, it's because something unexpected is happening. Use fun.na.by=NULL if you don't want to be notified and want to go ahead regardless.
as.fun	The default is to return a data.table if x is a data.table and return a data.frame in all other cases. Pass a function in as.fun to convert to something else.
...	additional arguments passed to data.table::merge. If all is among them, an error will be returned.

Details

Besides merging and checking rows, mergeCheck makes sure the order in x is retained in the resulting data (both rows and column order). Also, a warning is given if column names are overlapping, making merge create new column names like col.x and col.y. Merges and other operations are done using data.table. If x is a data.frame (and not a data.table), it will internally be converted to a data.table, and the resulting data.table will be converted back to a data.frame before returning.

mergeCheck is for the kind of merges where we think of x as the data to be enriched with columns from y - rows unchanged. This is even further limited than a left join where you can match rows multiple times. A common example of the use of mergeCheck is for adding covariates to a pk/pd data set. We do not want that to remove or duplicate doses, observations, or simulation records. In those cases, mergeCheck does all needed checks, and you can run full speed without checking dimensions (which is anyway not exactly the right thing to do in the general case) or worry that something might go wrong.

Checks performed:

- x has >0 rows
- by columns are present in x an y
- Merge is not performed on NA values. If by=ID and both x$ID and y$ID contain NA's, an error is thrown (see argument fun.na.by).
- Merge is done by all common column names in x and y. A warning is thrown if there are column names that are not being used to merge by. This will result in two columns named like BW.x and BW.y and is often unintended.
- Before merging a row counter is added to x. After the merge, the result is assured to have exactly one occurrence of each of the values of the row counter in x.

Moreover, row and column order from x is retained in the result.

Value

a data.frame resulting from merging x and y. Class as defined by as.fun.

See Also

Other DataCreate: `NMorderColumns()`, `NMstamp()`, `NMwriteData()`, `addTAPD()`, `findCovs()`, `findVars()`, `flagsAssign()`, `flagsCount()`, `tmpcol()`

Examples

```
df1 <- data.frame(x = 1:10,
                  y=letters[1:10],
                  stringsAsFactors=FALSE)
df2 <- data.frame(y=letters[1:11],
                  x2 = 1:11,
                  stringsAsFactors=FALSE)

mc1 <- mergeCheck(x=df1,y=df2,by="y")

## Notice as opposed to most merge/join algorithms, mergeCheck by
```

```
#default retains both row and column order from x
library(data.table)
merge(as.data.table(df1),as.data.table(df2))
## Here we get a duplicate of a df1 row in the result. If we only
## check dimensions, we make a mistake. mergeCheck captures the
## error - and tell us where to find the problem (ID 31 and 180):
## Not run:
pk <- readRDS(file=system.file("examples/data/xgxr2.rds",package="NMdata"))
dt.cov <- pk[,.(ID=unique(ID))]
dt.cov[,COV:=sample(1:5,size=.N,replace=TRUE)]
dt.cov <- dt.cov[c(1,1:(.N-1))]
dim(pk)
res.merge <- merge(pk,dt.cov,by="ID")
dim(res.merge)
mergeCheck(pk,dt.cov,by="ID")

## End(Not run)
```

`NMcheckColnames` *Compare $INPUT in control stream to column names in input data*

Description

Mis-specification of column names in $DATA is a common source of problems with Nonmem models, and should be one of the first things to check for when seemingly inexplicable things happen. This function lines up input data column names with $DATA and how NMscanData will interpret $DATA so you can easily spot if something is off.

Usage

```
NMcheckColnames(file, as.fun, ...)
```

Arguments

`file`	A Nonmem control stream or list file
`as.fun`	See ?NMdataConf
`...`	Additional arguments passed to

Value

An overview of input column names and how they are translated

NMcheckData *Check data for Nonmem compatibility or check control stream for data compatibility*

Description

Check data in various ways for compatibility with Nonmem. Some findings will be reported even if they will not make Nonmem fail but because they are typical dataset issues.

Usage

```
NMcheckData(
  data,
  file,
  covs,
  covs.occ,
  cols.num,
  col.id = "ID",
  col.time = "TIME",
  col.dv = "DV",
  col.mdv = "MDV",
  col.cmt = "CMT",
  col.amt = "AMT",
  col.flagn,
  col.row,
  col.usubjid,
  cols.dup,
  type.data = "est",
  na.strings,
  return.summary = FALSE,
  quiet = FALSE,
  as.fun
)
```

Arguments

data	The data to check. data.frame, data.table, tibble, anything that can be converted to data.table.
file	Alternatively to checking a data object, you can use file to specify a control stream to check. This can either be a (working or non-working) input control stream or an output control stream. In this case, NMdataCheck checks column names in data against control stream (see NMcheckColnames), reads the data as Nonmem would do, and do the same checks on the data as NMdataCheck would do using the data argument. col.flagn is ignored in this case - instead, ACCEPT/IGNORE statements in control stream are applied. The file argument is useful for debugging a Nonmem model.

covs	columns that contain subject-level covariates. They are expected to be non-missing, numeric and not varying within subjects.
covs.occ	A list specifying columns that contain subject:occasion-level covariates. They are expected to be non-missing, numeric and not varying within combinations of subject and occasion. `covs.occ=list(PERIOD=c("FED"))` means that `FED` is the covariate, while `PERIOD` indicates the occasion.
cols.num	Columns that are expected to be present, numeric and non-NA. If a character vector is given, the columns are expected to be used in all rows. If a column is only used for a subset of rows, use a list and name the elements by subsetting strings. See examples.
col.id	The name of the column that holds the subject identifier. Default is "ID".
col.time	The name of the column holding actual time.
col.dv	The name of the column holding the dependent variable. For now, only one column can be specified, and `MDV` is assumed to match this column. Default is `DV`.
col.mdv	The name of the column holding the binary indicator of the dependent variable missing. Default is `MDV`.
col.cmt	The name(s) of the compartment column(s). These will be checked to be positive integers for all rows. They are also used in checks for row duplicates.
col.amt	The name of the dose amount column.
col.flagn	Optionally, the name of the column holding numeric exclusion flags. Default value is `FLAG` and can be configured using `NMdataConf`. Even though `FLAG` is the default value, no finding will be returned if the column is missing unless explicitly defined as `col.flagn="FLAG"`. This is because this way of using exclusion flags is only one of many ways you could choose to handle exclusions. Disable completely by using `col.flagn=FALSE`.
col.row	A column with a unique value for each row. Such a column is recommended to use if possible. Default (`"ROW"`) can be modified using `NMdataConf`.
col.usubjid	Optional unique subject identifier. It is recommended to keep a unique subject identifier (typically a character string including an abbreviated study name and the subject id) from the clinical datasets in the analysis set. If you supply the name of the column holding this identifier, NMcheckData will check that it is non-missing, that it is unique within values of col.id (i.e. that the analysis subject ID's are unique across actual subjects), and that col.id is unique within the unique subject ID (a violation of the latter is less likely).
cols.dup	Additional column names to consider in search of duplicate events. col.id, col.cmt, col.evid, and col.time are always considered if found in data, and cols.dup is added to this list if provided.
type.data	"est" for estimation data (default), and "sim" for simulation data. Differences are that `col.row` is not expected for simulation data, and subjects will be checked to have `EVID==0` rows for estimation data and `EVID==2` rows for simulation data.
na.strings	Strings to be accepted when trying to convert characters to numerics. This will typically be a string that represents missing values. Default is ".". Notice, actual NA, i.e. not a string, is allowed independently of na.strings. See `?NMisNumeric`.

`return.summary`	If TRUE (not default), the table summary that is printed if `quiet=FALSE` is returned as well. In that case, a list is returned, and the findings are in an element called findings.
`quiet`	Keep quiet? Default is not to.
`as.fun`	The default is to return data as a `data.frame`. Pass a function (say `tibble::as_tibble`) in as.fun to convert to something else. If `data.tables` are wanted, use `as.fun="data.table"`. The default can be configured using `NMdataConf`.

Details

The following checks are performed. The term "numeric" does not refer to a numeric representation in R, but compatibility with Nonmem. The character string "2" is in this sense a valid numeric, "id2" is not.

- Column names must be unique and not contain special characters
- If an exclusion flag is used (for ACCEPT/IGNORE in Nonmem), elements must be non-missing and integers. Notice, if an exclusion flag is found, the rest of the checks are performed on rows where that flag equals 0 (zero) only.
- If a unique row identifier is found, it has to be non-missing, increasing integers.
- col.time (TIME), EVID, col.id (ID), col.cmt (CMT), and col.mdv (MDV): If present, elements must be non-missing and numeric.
- col.time (TIME) must be non-negative
- EVID must be in {0,1,2,3,4}.
- CMT must be positive integers. However, can be missing or zero for EVID==3.
- MDV must be the binary (1/0) representation of is.na(DV) for dosing records (EVID==0).
- AMT must be 0 or NA for EVID 0 and 2
- AMT must be positive for EVID 1 and 4
- DV must be numeric
- DV must be missing for EVID in {1,4}.
- If found, RATE must be a numeric, equaling -2 or non-negative for dosing events.
- If found, SS must be a numeric, equaling 0 or 1 for dosing records.
- If found, `ADDL` must be a non-negative integer for dosing records. II must be present.
- If found, II must be a non-negative integer for dosing records. `ADDL` must be present.
- ID must be positive and values cannot be disjoint (all records for each ID must be following each other. This is technically not a requirement in Nonmem but most often an error. Use a second ID column if you deliberately want to soften this check)
- TIME cannot be decreasing within ID, unless EVID in {3,4}.
- all ID's must have doses (EVID in {1,4})
- all ID's must have observations (EVID==0)
- ID's should not have leading zeros since these will be lost when Nonmem read, then write the data.
- If a unique row identifier is used, this must be non-missing, increasing, integer

- Character values must not contain commas (they will mess up writing/reading csv)
- Columns specified in covs argument must be non-missing, numeric and not varying within subjects.
- Columns specified in covs.occ must be non-missing, numeric and not varying within combinations of subject and occasion.
- Columns specified in cols.num must be present, numeric and non-NA.
- If a unique subject identifier column (col.usubjid) is provided, col.id must be unique within values of col.usubjid and vice versa.
- Events should not be duplicated. For all rows, the combination of col.id, col.cmt ,col.evid, col.time plus the optional columns specified in cols.dup must be unique. In other words, if a subject (col.id) that has say observations (col.evid) at the same time (col.time), this is considered a duplicate. The exception is if there is a reset event (col.evid is 3 or 4) in between the two rows. cols.dup can be used to add columns to this analysis. This is useful for different assays run on the same compartment (say a DVID column) or maybe stacked datasets. If col.cmt is of length>1, this search is repeated for each cmt column.

Value

A table with findings

Examples

```
## Not run:
dat <- readRDS(system.file("examples/data/xgxr2.rds", package="NMdata"))
NMcheckData(dat)
dat[EVID==0,LLOQ:=3.5]
## expecting LLOQ only for samples
NMcheckData(dat,cols.num=list(c("STUDY"),"EVID==0"=c("LLOQ")))

## End(Not run)
```

NMdataConf *Configure default behavior of NMdata functions*

Description

Configure default behavior across the functions in NMdata rather than typing the arguments in all function calls. Configure for your file organization, data set column names, and other NMdata behavior. Also, you can control what data class NMdata functions return (say data.tables or tibbles if you prefer one of those over data.frames).

Usage

```
NMdataConf(..., allow.unknown = FALSE)
```

Arguments

...	NMdata options to modify. These are named arguments, like for base::options. Normally, multiple arguments can be used. The exception is if reset=TRUE is used which means all options are restored to default values. If NULL is passed to an argument, the argument is reset to default. is See examples for how to use.
allow.unknown	Allow to store configuration of variables that are not pre-defined in NMdata. This should only be needed in cases where say another package wants to use the NMdata configuration system for variables unknown to NMdata.

Parameters that can be controlled are:

- args.fread Arguments passed to fread when reading _input_ data files (fread options for reading Nonmem output tables cannot be configured at this point). If you change this, you are starting from scratch, except from file. This means that existing default argument values are all disregarded.
- args.fwrite Arguments passed to fwrite when writing csv files (NMwriteData). If you use this, you have to supply all arguments you want to use with fwrite, except for x (the data) and file.
- as.fun A function that will be applied to data returned by various data reading functions (NMscanData, NMreadTab, NMreadCsv, NMscanInput, NMscanTables). Also, data processing functions like mergeCheck, findCovs, findVars, flagsAssign, flagsCount take this into account, but slightly differently. For these functions that take data as arguments, the as.fun configuration is only taken into account if a the data passed to the functions are not of class data.table. The argument as.fun to these functions is always adhered to. Pass an actual function, say as.fun=tibble::as_tibble. If you want data.table, use as.fun="data.table" (not a function).
- check.time Logical, applies to NMscanData only. NMscanData by defaults checks if output control stream is newer than input control stream and input data. Set this to FALSE if you are in an environment where time stamps cannot be relied on.
- col.flagc The name of the column containing the character flag values for data row omission. Default value is flag. Used by flagsAssign, flagsCount.
- col.flagn The name of the column containing numerical flag values for data row omission. Default value is FLAG. Used by flagsAssign, flagsCount, NMcheckData.
- col.model The name of the column that will hold the name of the model. See modelname too (which defines the values that the column will hold).
- col.nmout A column of this name will be a logical representing whether row was in output table or not.
- col.nomtime The name of the column holding nominal time. This is only used for sorting columns by NMorderColumns.
- col.row The name of the column containing a unique row identifier. This is used by NMscanData when merge.by.row=TRUE, and by NMorderColumns (row counter will be first column in data).
- col.id The name of the column holding the numeric subject ID. As of 'NMdata' 0.1.5 this is only used for sorting columns by NMorderColumns.

- col.time The name of the column holding actual time. As of ‘NMdata‘ 0.1.5 this is only used for sorting columns by NMorderColumns.
- dir.psn The directory in which to find psn executables like ‘execute‘ and ‘update_inits‘. Default is "" meaning that executables must be in the system search path. Not used by NMdata.
- dir.res Directory in which ‘NMsim‘ will store simulation results files. Not used by NMdata. See dir.sims too.
- dir.sims Directory in which ‘NMsim‘ will store Nonmem simulations. Not used by NMdata. See dir.res too.
- file.cov A function that will derive the path to the covariance (.cov) output file stream based on the path to the output control stream. Technically, it can be a string too, but when using NMdataConf, this would make little sense because it would direct all output control streams to the same input control streams.
- file.ext A function that will derive the path to the parameter (.ext) output file stream based on the path to the output control stream. Technically, it can be a string too, but when using NMdataConf, this would make little sense because it would direct all output control streams to the same input control streams.
- file.mod A function that will derive the path to the input control stream based on the path to the output control stream. Technically, it can be a string too, but when using NMdataConf, this would make little sense because it would direct all output control streams to the same input control streams.
- file.phi A function that will derive the path to the Nonmem output (.phi) file containing individual ETA, ETC, and/or PHI values stream based on the path to the output control stream. Technically, it can be a string too, but when using NMdataConf, this would make little sense because it would direct all output control streams to the same input control streams.
- file.data A function that will derive the path to the input data based on the path to the output control stream. Technically, it can be a string too, but when using NMdataConf, this would make little sense because it would direct all output control streams to the same input control streams.
- formats.read Prioritized input data file formats to look for and use if found. Default is c("rds","csv") which means `rds` will be used if found, and `csv` if not. `fst` is possible too.
- formats.write character vector of formats.write. Default is c("csv","rds"). "fst" is possible too.
- merge.by.row Adjust the default combine method in NMscanData.
- modelname A function that will translate the output control stream path to a model name. Default is to strip .lst, so /path/to/run1.lst will become run1. Technically, it can be a string too, but when using NMdataConf, this would make little sense because it would translate all output control streams model name.
- path.nonmem Path (a character string) to a nonmem executable. Not used by NMdata. Default is NULL.
- quiet For non-interactive scripts, you can switch off the chatty behavior once and for all using this setting.

- recover.rows In NMscanData, Include rows from input data files that do not exist in output tables? This will be added to the $row dataset only, and $run, $id, and $occ datasets are created before this is taken into account. A column called nmout will be TRUE when the row was found in output tables, and FALSE when not. Default is FALSE.
- use.input In NMscanData, merge with columns in input data? Using this, you don't have to worry about remembering including all relevant variables in the output tables. Default is TRUE.
- use.rds Deprecated, use `formats.read` and `formats.write` instead. Affects `NMscanData()`, `NMscanInput()`, `NMwriteData()`.

Details

Recommendation: Use this function transparently in the code and not in a configuration file hidden from other users.

Value

If no arguments given, a list of active settings. If arguments given and no issues found, TRUE invisibly.

Examples

```
## get current defaults
NMdataConf()
## change a parameter
NMdataConf(check.time=FALSE)
## reset one parameter to default value
NMdataConf(modelname=NULL)
## reset all parameters to defaults
NMdataConf(reset=TRUE)
```

NMdataOperations *Basic arithmetic on NMdata objects*

Description

Basic arithmetic on NMdata objects

Usage

```
## S3 method for class 'NMdata'
merge(x, ...)

## S3 method for class 'NMdata'
t(x, ...)

## S3 method for class 'NMdata'
```

```
dimnames(x, ...)

## S3 method for class 'NMdata'
rbind(x, ...)

## S3 method for class 'NMdata'
cbind(x, ...)
```

Arguments

x	an NMdata object
...	arguments passed to other methods.

Details

When 'dimnames', 'merge', 'cbind', 'rbind', or 't' is called on an 'NMdata' object, the 'NMdata' class is dropped, and then the operation is performed. So if and 'NMdata' object inherits from 'data.frame' and no other classes (which is default), these operations will be performed using the 'data.frame' methods. But for example, if you use 'as.fun' to get a 'data.table' or 'tbl', their respective methods are used instead.

Value

An object that is not of class 'NMdata'.

NMexpandDoses *Transform repeated dosing events (ADDL/II) to individual dosing events*

Description

Replaces single row repeated dosing events by multiple lines, then reorders rows with respect to ID and TIME. If the row order is different, you have to reorder the output manually.

Usage

```
NMexpandDoses(
  data,
  col.time = "TIME",
  col.id = "ID",
  col.evid = "EVID",
  track.expand = FALSE,
  subset.dos,
  quiet = FALSE,
  as.fun
)
```

Arguments

data	The data set to expand
col.time	The name of the column holding the time on which time since previous dose will be based. This is typically actual or nominal time since first dose.
col.id	The subject identifier. All new columns will be derived within unique values of this column.
col.evid	The name of the event ID column. This must exist in data. Default is EVID.
track.expand	Keep track of what rows were in data originally and which ones are added by NMexpandDoses by including a column called nmexpand? nmexpand will be TRUE if the row is "generated" by NMexpandDoses.
subset.dos	A string that will be evaluated as a custom expression to identify relevant events.
quiet	Suppress messages back to user (default is FALSE)
as.fun	The default is to return data as a data.frame. Pass a function (say tibble::as_tibble) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.

Value

A data set with at least as many rows as data. If doses are found to expand, these will be added.

NMextractDataFile *Extract the data file used in a control stream*

Description

A function that identifies the input data file based on a control stream. The default is to look at the $DATA section of of the output control stream (or input control stream if file.mod argument is used). This can be partly or fully overruled by using the dir.data or file.data arguments.

Usage

```
NMextractDataFile(file, dir.data = NULL, file.mod, file.data = NULL)
```

Arguments

file	The input control stream or the list file.
dir.data	See NMscanInput. If used, only the file name mentioned in $DATA is used. dir.data will be used as the path, and the existence of the file in that directory is not checked.
file.mod	The input control stream. Default is to look for \"file\" with extension changed to .mod (PSN style). You can also supply the path to the file, or you can provide a function that translates the output file path to the input file path. The default behavior can be configured using NMdataConf. See dir.data too.
file.data	Specification of the data file path. When this is used, the control streams are not used at all.

Value

The path to the input data file.

`NMextractText` *Versatile text extractor from Nonmem (input or output) control streams*

Description

If you want to extract input sections like $PROBLEM, $DATA etc, see NMreadSection. This function is more general and can be used to extract eg result sections.

Usage

```
NMextractText(
  file,
  lines,
  text,
  section,
  char.section,
  char.end = char.section,
  return = "text",
  keep.empty = FALSE,
  keep.name = TRUE,
  keep.comments = TRUE,
  as.one = TRUE,
  clean.spaces = FALSE,
  simplify = TRUE,
  match.exactly = TRUE,
  type = "mod",
  linesep = "\n",
  keepEmpty,
  keepName,
  keepComments,
  asOne,
  cleanSpaces
)
```

Arguments

`file`	A file path to read from. Normally a .mod or .lst. See lines and text as well.
`lines`	Text lines to process. This is an alternative to using the file and text arguments.
`text`	Use this argument if the text to process is one long character string, and indicate the line separator with the linesep argument. Use only one of file, lines, and text.
`section`	The name of section to extract. Examples: "INPUT", "PK", "TABLE", etc. It can also be result sections like "MINIMIZATION".

char.section	The section denoted as a string compatible with regular expressions. "$" (remember to escape properly) for sections in .mod files, "0" for results in .lst files.
char.end	A regular expression to capture the end of the section. The default is to look for the next occurrence of char.section.
return	If "text", plain text lines are returned. If "idx", matching line numbers are returned. "text" is default.
keep.empty	Keep empty lines in output? Default is FALSE. Notice, comments are removed before empty lines are handled if 'keep.comments=TRUE'.
keep.name	Keep the section name in output (say, "$PROBLEM") Default is TRUE. It can only be FALSE, if return="text".
keep.comments	Default is to keep comments. If FALSE, the will be removed.
as.one	If multiple hits, concatenate into one. This will most often be relevant with name="TABLE". If FALSE, a list will be returned, each element representing a table. Default is TRUE. So if you want to process the tables separately, you probably want FALSE here.
clean.spaces	If TRUE, leading and trailing are removed, and multiplied succeeding white spaces are reduced to single white spaces.
simplify	If asOne=FALSE, do you want the result to be simplified if only one table is found? Default is TRUE which is desirable for interactive analysis. For programming, you probably want FALSE.
match.exactly	Default is to search for exact matches of 'section'. If FALSE, only the first three characters are macthed. E.G., this allows "ESTIMATION" to match "ESTIMATION" or "EST".
type	Either mod, res or NULL. mod is for information that is given in .mod (.lst file can be used but results section is disregarded). If NULL, NA or empty string, everything is considered.
linesep	If using the text argument, use linesep to indicate how lines should be separated.
keepEmpty	Deprecated. See keep.empty.
keepName	Deprecated. See keep.name.
keepComments	Deprecated. See keep.comments.
asOne	Deprecated. See as.one.
cleanSpaces	Deprecated. See clean.spaces.

Details

This function is planned to get a more general name and then be called by NMreadSection.

Value

character vector with extracted lines.

See Also

Other Nonmem: NMapplyFilters(), NMgenText(), NMreadSection(), NMreplaceDataFile(), NMwriteSection()

Examples

```
NMreadSection(system.file("examples/nonmem/xgxr001.lst", package = "NMdata"),section="DATA")
```

NMgenText *Generate text for INPUT and possibly DATA sections of NONMEM control streams.*

Description

The user is provided with text to use in Nonmem. NMwriteSection can use the results to update the control streams. INPUT lists names of the data columns while DATA provides a path to data and ACCEPT/IGNORE statements. Once a column is reached that Nonmem will not be able to read as a numeric and column is not in nm.drop, the list is stopped. Only exception is TIME which is not tested for whether character or not.

Usage

```
NMgenText(
  data,
  drop,
  col.flagn = "FLAG",
  rename,
  copy,
  file,
  dir.data,
  capitalize = FALSE,
  until,
  allow.char.TIME = TRUE,
  width,
  quiet
)
```

Arguments

data	The data that NONMEM will read.
drop	Only used for generation of proposed text for INPUT section. Columns to drop in Nonmem $INPUT. This has two implications. One is that the proposed $INPUT indicates =DROP after the given column names. The other that in case it is a non-numeric column, succeeding columns will still be included in $INPUT and can be read by NONMEM.
col.flagn	Name of a numeric column with zero value for rows to include in Nonmem run, non-zero for rows to skip. The argument is only used for generating the proposed $DATA text to paste into the Nonmem control stream. To skip this feature, use col.flagn=NULL. Default is defined by NMdataConf.

`rename` For the $INPUT text proposal only. If you want to rename columns in NONMEM $DATA, NMwriteData can adjust the suggested $DATA text. If you plan to use BBW instead of BWBASE in Nonmem, consider rename=c(BBW="BWBASE"). The result will include BBW and not BWBASE.

`copy` For the $INPUT text proposal only. If you plan to use additional names for columns in Nonmem $INPUT, NMwriteData can adjust the suggested $INPUT text. Say you plan to use CONC as DV in Nonmem, use copy=c(DV="CONC"), i.e. copy=c(newname="existing"). INPUT suggestion will in this case contain DV=CONC.

`file` The file name NONMEM will read the data from (for the $DATA section). It can be a full path.

`dir.data` For the $DATA text proposal only. The path to the input datafile to be used in the Nonmem $DATA section. Often, a relative path to the actual Nonmem run is wanted here. If this is used, only the file name and not the path from the file argument is used.

`capitalize` For the $INPUT text proposal only. If TRUE, all column names in $INPUT text will be converted to capital letters.

`until` Use this to truncate the columns in $INPUT. until can either be a character (column name) or a numeric (column number). If a character is given, it is matched against the resulting column name representation in $INPUT, i.e. this could be "DV=CONC" if you are using in this case the copy argument. In case until is of length>1, the maximum will be used (probably only interesting if character values are supplied).

`allow.char.TIME` For the $INPUT text proposal only. Assume Nonmem can read TIME even if it can't be translated to numeric. This is necessary if using the 00:00 format. Default is TRUE.

`width` If positive, will be passed to strwrap for the $INPUT text. If missing or NULL, strwrap will be called with default value. If negative or zero, strwrap will not be called.

`quiet` Hold messages back? Default is defined by NMdataConf.

Value

Text for inclusion in Nonmem control stream, invisibly. A list with elements 'DATA' and 'INPUT'.

See Also

Other Nonmem: `NMapplyFilters()`, `NMextractText()`, `NMreadSection()`, `NMreplaceDataFile()`, `NMwriteSection()`

NMinfo *Get metadata from an NMdata object*

Description

Extract metadata such as info on tables, columns and further details in your favorite class

Usage

```
NMinfo(data, info, as.fun)
```

Arguments

data	An object of class NMdata (a result of NMscanData)
info	If not passed, all the metadata is returned. You can use "details", "tables", or "columns" to get only these subsets. If info is "tables" or "columns"
as.fun	The default is to return data as a data.frame. Pass a function (say tibble::as_tibble) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.

Value

A table of class as defined by as.fun in case info is "columns" or "tables". A list if info missing or equal to "details".

NMisNumeric *Test if a variable can be interpreted by Nonmem*

Description

Nonmem can only interpret numeric data. However, a factor or a character variable may very well be interpretable by Nonmem (e.g. "33"). This function tells whether Nonmem will be able to read it.

Usage

```
NMisNumeric(x, na.strings = ".", each = FALSE)
```

Arguments

x	The vector to check Don't export
na.strings	Tolerated strings that do not translate to numerics. Default is to accept "." because it's common to write missing values that way to Nonmem (even if Nonmem will handle them as zeros rather than missing). Notice actual NA's are accepted so you may want to use na.strings=NULL if you don't code missings as "." and just do this when writing the data set to a delimited file (like NMwriteData will do for you).
each	Use each=TRUE to evaluate each element in a vector individually. The default (each=FALSE) is to return a single-length logical for a vector x summarizing whether all the elements are numeric-compatible.

Value

TRUE or FALSE

NMorderColumns *Standardize column order in Nonmem input data*

Description

Order data columns for easy export to Nonmem. No data values are edited. The order is configurable through multiple arguments. See details.

Usage

```
NMorderColumns(
  data,
  first,
  last,
  lower.last = FALSE,
  chars.last = TRUE,
  alpha = TRUE,
  col.id,
  col.nomtime,
  col.time,
  col.row,
  col.flagn,
  col.dv = "DV",
  as.fun = NULL,
  quiet
)
```

Arguments

data	The dataset which columns to reorder.
first	Columns that should come almost first. See details.
last	Columns to move to back of dataset. If you work with a large dataset, and some columns are irrelevant for the Nonmem runs, you can use this argument.
lower.last	Should columns which names contain lowercase characters be moved towards the back? Some people use a standard of lowercase variables (say "race") being character representations ("Asian", "Caucasian", etc.) variables and the uppercase (1,2,...) being the numeric representation for Nonmem.
chars.last	Should columns which cannot be converted to numeric be put towards the end? A column can be a character or a factor in R, but still be valid in Nonmem (often the case for ID which can only contain numeric digits but really is a character or factor). So rather than only looking at the column class, the columns are attempted converted to numeric. Notice, it will attempted to be converted to numeric to test whether Nonmem will be able to make sense of it, but the values in the resulting dataset will be untouched. No values will be edited. If TRUE, logicals will always be put last. NA's must be NA or ".".
alpha	Sort columns alphabetically. Notice, this is the last order priority applied.
col.id	Name of the (numeric) unique subject ID. Can be controlled with 'NMdataConf()'.
col.nomtime	The name of the column containing nominal time. If given, it will put the column quite far left, just after row counter and 'col.id'. Default value is NOMTIME and can be configured with 'NMdataConf()'.
col.time	The name of the column containing actual time. If given, it will put the column quite far left, just after row counter, subject ID, and nominal time. Default value is 'TIME'. Can be controlled with 'NMdataConf()'.
col.row	A row counter column. This will be the first column in the dataset. Technically, you can use it for whatever column you want first. Default value is 'ROW' and can be configured with 'NMdataConf()'.
col.flagn	The name of the column containing numerical flag values for data row omission. Default value is FLAG and can be configured with 'NMdataConf()'.
col.dv	a vector of column names to put early to represent dependent variable(s). Default is DV.
as.fun	The default is to return a data.table if data is a data.table and return a data.frame in all other cases. Pass a function in as.fun to convert to something else. The default can be configured using 'NMdataConf()'. However, if data is a data.table, settings via 'NMdataConf()' are ignored.
quiet	If true, no warning will be given about missing standard Nonmem columns.

Details

This function will change the order of columns but it will never edit values in any columns. The ordering is by the following steps, each step depending on corresponding argument.

"col.row - " Row id if argument row is non-NULL

"not editable - " ID (if a column is called ID)

"col.nomtime - " Nominal time.

"col.time - " Actual time.

"first - " user-specified first columns

"Only col.dv editable - " Standard Nonmem columns: EVID, CMT, AMT, RATE, col.dv, MDV

"last - " user-specified last columns

"chars.last - " numeric, or interpretable as numeric

"not editable - " less often used Nonmem names: col.flagn, OCC, ROUTE, GRP, TRIAL, DRUG, STUDY

"lower.last - " lower case in name

"alpha - " Alphabetic/numeric sorting

Value

data with modified column order.

See Also

Other DataCreate: `NMstamp()`, `NMwriteData()`, `addTAPD()`, `findCovs()`, `findVars()`, `flagsAssign()`, `flagsCount()`, `mergeCheck()`, `tmpcol()`

`NMreadCov` *Read in data file*

Description

Read in data file

Usage

```
NMreadCov(file, ...)
```

Arguments

`file`	The .cov covariance Nonmem matrix file to read
`...`	Passed to fread

Details

This function is taken from nonmem2rx::nmcov which was based on NMdata::NMreadTab.

Value

A matrix with covariance step from NONMEM

Author(s)

Philip Delff and Matthew L. Fidler

NMreadCsv *Read input data formatted for Nonmem*

Description

This function is especially useful if the csv file was written using NMwriteData.

Usage

```
NMreadCsv(
  file,
  args.fread,
  as.fun = NULL,
  format = fnExtension(file),
  args.fst
)
```

Arguments

file	The file to read. Must be pure text.
args.fread	List of arguments passed to fread. Notice that except for "file", you need to supply all arguments to fread if you use this argument. Default values can be configured using NMdataConf.
as.fun	The default is to return data as a data.frame. Pass a function (say tibble::as_tibble) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.
format	Format of file to read. Can be of length>1 in which case the first format found will be used (i.e. format is a prioritized vector). If not one of "rds" or "fst", it is assumed to be a delimited text file. Default is to determine this from the file name extension. Notice, if a delimited format is used, the extension can very well be different from "csv" (say file name is "input.tab")". This will work for any delimited format supported by fread.
args.fst	Optional arguments to pass to `read_fst` if `format="fst"` is used.

Details

This is almost just a shortcut to fread so you don't have to remember how to read the data that was exported for Nonmem. The only added feature is that meta data as written by NMwriteData is read and attached as NMdata metadata before data is returned.

Value

A data set of class as defined by as.fun.

See Also

NMwriteData

Other DataRead: NMreadTab(), NMscanData(), NMscanInput(), NMscanTables()

NMreadExt *Read information from Nonmem ext files*

Description

Read information from Nonmem ext files

Usage

```
NMreadExt(
  file,
  return,
  as.fun,
  modelname,
  col.model,
  auto.ext,
  tableno = "max",
  file.ext
)
```

Arguments

file	Path to the ext file
return	The .ext file contains both final parameter estimates and iterations of the estimates. If return="pars" (default) the final estimates are returned in addition to what other parameter-level information is found, like FIX, sd etc. as columns. If return="iterations", the iterations are returned. If return="both", both er returned, though in separate data.frames compiled in a list.
as.fun	The default is to return data as a data.frame. Pass a function (say tibble::as_tibble) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.
modelname	See ?NMscanData
col.model	See ?NMscanData
auto.ext	If TRUE (default) the extension will automatically be modified using 'NMdataConf()$file.ext'. This means 'file' can be the path to an input or output control stream, and 'NMreadExt' will still read the '.ext' file.
tableno	In case the ext file contains multiple tables, this argument controls which one to choose. The options are • "max" (default) Pick the table with the highest table number. This typically means the results from the last '$ESTIMATION' step are used.

- "min" Pick results from the first table available.
- "all" Keep all results. The tables can be distinguished by the 'tableno' column.
- an integer greater than 0, in which case the table with this table number will be picked.

`file.ext` Deprecated. Please use `file` instead.

Details

The parameter table returned if `return="pars"` or `return="all"` will contain columns based on the Nonmem 7.5 manual. It defines codes for different parameter-level values. They are:

-1e+09: se -1000000002: eigCor -1000000003: cond -1000000004: stdDevCor -1000000005: seStdDevCor -1000000006: FIX -1000000007: termStat -1000000008: partLik

The parameter name is in the `parameter` column. The "parameter type", like "THETA", "OMEGA", "SIGMA" are available in the `par.type` column. Counters are available in `i` and `j` columns. `j` will be `NA` for `par.type=="THETA"`

The objective function value is included as a parameter.

Notice that in case multiple tables are available in the 'ext' file, the column names are taken from the first table. E.g., in case of SAEM/IMP estimation, the objective function values will be in the 'SAEMOBJ' column, even for the IMP step. This may change in the future.

Value

If `return="all"`, a list with a final parameter table and a table of the iterations. If `return="pars"`, only the parameter table, and if `return="iterations"` only the iterations table. If you need both, it may be more efficient to only read the file once and use `return="all"`. Often, only one of the two are needed, and it more convenient to just extract one.

NMreadParsText *Read comments to parameter definitions in Nonmem control streams*

Description

When interpreting parameter estimates, it is often needed to recover information about the meaning of the different parameters from control stream. 'NMreadParsText' provides a flexible way to organize the comments in the parameter sections into a 'data.frame'. This can subsequently easily be merged with parameter values as obtained with 'NMreadExt'.

Usage

```
NMreadParsText(
  file,
  fields,
  fields.omega = fields,
  fields.sigma = fields.omega,
```

```
  use.theta.nums = FALSE,
  spaces.split = FALSE,
  modelname,
  col.model,
  as.fun
)
```

Arguments

file	Path to the control stream to read.
fields	Defines naming and splitting of contents of lines in parameter sections. Default is "%init;%symbol;%num;%label;%unit".
fields.omega	Like 'fields', applied to '$OMEGA' section. Default is to reuse 'fields'.
fields.sigma	Like 'fields', applied to '$SIGMA' section. Default is to reuse 'fields.omega'.
use.theta.nums	If the num field should be used to number thetas. I do not see where this is advantageous to do.
spaces.split	Is a blank in 'fields' to be treated as a field seperator? Default is not to (i.e. neglect spaces in 'fields').
modelname	See ?NMscanData
col.model	See ?NMscanData
as.fun	See ?NMscanData

Details

Off-diagonal omega and sigma elements will only be correctly treated if their num field specifies say 1-2 to specify it is covariance between 1 and 2.

NMreadPhi *Read information from Nonmem phi files*

Description

Read information from Nonmem phi files

Usage

```
NMreadPhi(file, as.fun, modelname, col.model, auto.ext, file.phi)
```

Arguments

file	Path to the phi file. If 'auto.ext=TRUE', the extension will automatically be changed using the setting in 'NMdataConf()$file.fir' - this by default means that the '.phi' extension will be used no matter what extension the provided file name has.

as.fun	The default is to return data as a data.frame. Pass a function (say tibble::as_tibble) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.
modelname	See ?NMscanData
col.model	See ?NMscanData
auto.ext	If TRUE (default) the extension will automatically be modified using 'NMdataConf()$file.phi'. This means 'file' can be the path to an input or output control stream, and 'NMreadPhi' will still read the '.phi' file.
file.phi	Deprecated. Use 'file'.

Value

A list with a final parameter table and a table of the iterations

---

`NMreadSection` *Extract sections of Nonmem control streams*

Description

This is a very commonly used wrapper for the input part of the model file. Look NMextractText for more general functionality suitable for the results part too.

Usage

```
NMreadSection(
  file = NULL,
  lines = NULL,
  text = NULL,
  section,
  return = "text",
  keep.empty = FALSE,
  keep.name = TRUE,
  keep.comments = TRUE,
  as.one = TRUE,
  clean.spaces = FALSE,
  simplify = TRUE,
  keepEmpty,
  keepName,
  keepComments,
  asOne,
  cleanSpaces,
  ...
)

NMgetSection(...)
```

Arguments

file	A file path to read from. Normally a .mod or .lst. See lines also.
lines	Text lines to process. This is an alternative to using the file argument.
text	Use this argument if the text to process is one long character string, and indicate the line separator with the linesep argument (handled by NMextractText). Use only one of file, lines, and text.
section	The name of section to extract without "$". Examples: "INPUT", "PK", "TABLE", etc. Not case sensitive.
return	If "text", plain text lines are returned. If "idx", matching line numbers are returned. "text" is default.
keep.empty	Keep empty lines in output? Default is FALSE. Notice, comments are removed before empty lines are handled if 'keep.comments=TRUE'.
keep.name	Keep the section name in output (say, "$PROBLEM") Default is FALSE. It can only be FALSE, if return="text".
keep.comments	Default is to keep comments. If FALSE, the will be removed.
as.one	If multiple hits, concatenate into one. This will most often be relevant with name="TABLE". If FALSE, a list will be returned, each element representing a table. Default is TRUE. So if you want to process the tables separately, you probably want FALSE here.
clean.spaces	If TRUE, leading and trailing are removed, and multiplied succeeding white spaces are reduced to single white spaces.
simplify	If asOne=FALSE, do you want the result to be simplified if only one section is found? Default is TRUE which is desirable for interactive analysis. For programming, you probably want FALSE.
keepEmpty	Deprecated. See keep.empty.
keepName	Deprecated. See keep.name.
keepComments	Deprecated. See keep.comments.
asOne	Deprecated. See as.one.
cleanSpaces	Deprecated. See clean.spaces.
...	Additional arguments passed to NMextractText

Value

character vector with extracted lines.

character vector with extracted lines.

Functions

- NMgetSection: Deprecated function name. Use NMreadSection.

See Also

Other Nonmem: NMapplyFilters(), NMextractText(), NMgenText(), NMreplaceDataFile(), NMwriteSection()

Examples

```
NMreadSection(system.file("examples/nonmem/xgxr001.lst", package="NMdata"),section="DATA")
```

NMreadTab *Read an output table file from Nonmem*

Description

Read a table generated by a $TABLE statement in Nonmem. Generally, these files cannot be read by read.table or similar because formatting depends on options in the $TABLE statement, and because Nonmem sometimes includes extra lines in the output that have to be filtered out. NMreadTab can do this automatically based on the table file alone.

Usage

```
NMreadTab(
  file,
  col.tableno,
  col.nmrep,
  col.table.name,
  header = TRUE,
  skip,
  quiet = TRUE,
  as.fun,
  ...
)
```

Arguments

file	path to Nonmem table file
col.tableno	In case of simulations where tables are being repeated, a counter of the repetition number can be useful to include in the output. For now, this will only work if the NOHEADER option is not used. This is because NMreadTab searches for the "TABLE NO..." strings in Nonmem output tables. If col.tableno is TRUE (default), a counter of tables is included as a column called NMREP. Notice, the table numbers in NMREP are cumulatively counting the number of tables reported in the file. NMREP is not the actual table number as given by Nonmem.
col.nmrep	col.nmrep If tables are repeated, include a counter? It does not relate to the order of the $TABLE statements but to cases where a $TABLE statement is run repeatedly. E.g., in combination with the SUBPROBLEMS feature in Nonmem, it is useful to keep track of the table (repetition) number. If col.nmrep is TRUE, this will be carried forward and added as a column called NMREP. This is default behavior when more than one $TABLE repetition is found in data. Set it to a different string to request the column with a different name. The argument is passed to NMscanTables.

col.table.name	The name of a column containing the name or description of the table (generated by Nonmem). The default is "table.name". Use FALSE not to include this column.
header	Use header=FALSE if table was created with NOHEADER option in $TABLE.
skip	The number of rows to skip. The default is skip=1 if header==TRUE and skip=0 if header==FALSE.
quiet	logical stating whether or not information is printed about what is being done. Default can be configured using NMdataConf.
as.fun	The default is to return data as a data.frame. Pass a function (say tibble::as_tibble) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.
...	Arguments passed to fread.

Details

The actual reading of data is based on data.table::fread. Generally, the function is fast thanks to data.table.

Value

The Nonmem table data.

See Also

Other DataRead: NMreadCsv(), NMscanData(), NMscanInput(), NMscanTables()

NMreplaceDataFile *Replace data file used in Nonmem control stream*

Description

Replace data file used in Nonmem control stream

Usage

```
NMreplaceDataFile(files, file.pattern, dir, path.data, newfile = file.mod, ...)
```

Arguments

files	Paths to input control streams to modify. See file.pattern and dir too.
file.pattern	A pattern to look for if ‘dir‘ is supplied too (and not ‘file.mod‘). This is used to modify multiple input control streams at once.
dir	Directory in which to look for ‘file.pattern‘. Notice, use either just ‘file.mod‘ or both ‘dir‘ and ‘file.pattern‘.
path.data	Path to input control stream to use in newfile
newfile	A path to a new control stream to write to (and don’t edit contents of ‘file.mod‘). Default is to overwrite ‘file.mod‘.
...	Additional arguments to pass to NMwriteSection.

Value

Lines for a new control stream (invisibly)

See Also

Other Nonmem: NMapplyFilters(), NMextractText(), NMgenText(), NMreadSection(), NMwriteSection()

NMscanData *Automatically find Nonmem input and output tables and organize data*

Description

This is a very general solution to automatically identifying, reading, and merging all output and input data in a Nonmem model. The most important steps are

- Read and combine output tables,
- If wanted, read input data and restore variables that were not output from the Nonmem model
- If wanted, also restore rows from input data that were disregarded in Nonmem (e.g. observations or subjects that are not part of the analysis)

Usage

```
NMscanData(
  file,
  col.row,
  use.input,
  merge.by.row,
  recover.rows,
  file.mod,
  dir.data,
  file.data,
  translate.input = TRUE,
  quiet,
  formats.read,
  args.fread,
  as.fun,
  col.id = "ID",
  modelname,
  col.model,
  col.nmout,
  col.nmrep,
  order.columns = TRUE,
  check.time,
  tz.lst,
  skip.absent = FALSE,
  tab.count,
  use.rds
)
```

Arguments

file	Path to a Nonmem control stream or output file from Nonmem (.mod or .lst)
col.row	A column with a unique value for each row. Such a column is recommended to use if possible. See merge.by.row and details as well. Default ("ROW") can be modified using NMdataConf.
use.input	Should the input data be added to the output data. Only column names that are not found in output data will be retrieved from the input data. Default is TRUE which can be modified using NMdataConf. See merge.by.row too.
merge.by.row	If use.input=TRUE, this argument determines the method by which the input data is added to output data. The default method (merge.by.row=FALSE) is to interpret the Nonmem code to imitate the data filtering (IGNORE and ACCEPT statements), but the recommended method is merge.by.row=TRUE which means that data will be merged by a unique row identifier. The row identifier must be present in input and at least one full length output data table. See argument col.row too.
recover.rows	Include rows from input data files that do not exist in output tables? This will be added to the $row dataset only, and $run, $id, and $occ datasets are created before this is taken into account. A column called nmout will be TRUE when the row was found in output tables, and FALSE when not. Default is FALSE and can be configured using NMdataConf.
file.mod	The input control stream file path. Default is to look for \"file\" with extension changed to .mod (PSN style). You can also supply the path to the file, or you can provide a function that translates the output file path to the input file path. The default behavior can be configured using NMdataConf. See dir.data too.
dir.data	The data directory can only be read from the control stream (.mod) and not from the output file (.lst). So if you only have the output control stream, use dir.data to tell in which directory to find the data file. If dir.data is provided, the .mod file is not used at all.
file.data	Specification of the data file path. When this is used, the control streams are not used at all.
translate.input	Default is TRUE, meaning that input data column names are translated according to $INPUT section in Nonmem listing file.
quiet	The default is to give some information along the way on what data is found. But consider setting this to TRUE for non-interactive use. Default can be configured using NMdataConf.
formats.read	Prioritized input data file formats to look for and use if found. Default is c("rds","csv") which means `rds` will be used if found, and `csv` if not. `fst` is possible too. Default can be modified using `NMdataConf()`.
args.fread	List of arguments passed to when reading _input_ data. Notice that except for "input" and "file", you need to supply all arguments to fread if you use this argument. Default values can be configured using NMdataConf.
as.fun	The default is to return data as a data.frame. Pass a function (say tibble::as_tibble) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.

`col.id` The name of the subject ID variable, default is "ID".

`modelname` The model name to be stored if col.model is not NULL. If not supplied, the name will be taken from the control stream file name by omitting the directory/path and deleting the .lst extension (path/run001.lst becomes run001). This can be a character string or a function which is called on the value of file (file is another argument to NMscanData). The function must take one character argument and return another character string. As example, see NMdataConf()$modelname. The default can be configured using NMdataConf.

`col.model` A column of this name containing the model name will be included in the returned data. The default is to store this in a column called "model". See argument "modelname" as well. Set to NULL if not wanted. Default can be configured using NMdataConf.

`col.nmout` A column of this name will be a logical representing whether row was in output table or not. Default can be modified using NMdataConf.

`col.nmrep` If tables are repeated, include a counter? It does not relate to the order of the $TABLE statements but to cases where a $TABLE statement is run repeatedly. E.g., in combination with the SUBPROBLEMS feature in Nonmem, it is useful to keep track of the table (repetition) number. If col.nmrep is TRUE, this will be carried forward and added as a column called NMREP. This is default behavior when more than one $TABLE repetition is found in data. Set it to a different string to request the column with a different name. The argument is passed to NMscanTables.

`order.columns` If TRUE (default), NMorderColumns is used to reorder the columns before returning the data. NMorderColumns will be called with alpha=FALSE, so columns are not sorted alphabetically. But standard Nonmem columns like ID, TIME, and other will be first. If col.row is used, this will be passed to NMorderColumns too.

`check.time` If TRUE (default) and if input data is used, input control stream and input data are checked to be newer than output control stream and output tables. These are important assumptions for the way information is merged by NMscanData. However, if data has been transferred from another system where Nonmem was run, these checks may not make sense, and you may not want to see these warnings. The default can be configured using NMdataConf. For the output control stream, the time stamp recorded by Nonmem is used if possible, and if the input data is created with NMwriteData, the recorded creation time is used if possible. If not, and for all other files, the file modification times are used.

`tz.lst` If supplied, the timezone to be used when reading the time stamp in the output control stream. Please supply something listed in OlsonNames(). Can be configured using NMdataConf() too.

`skip.absent` Skip missing output table files with a warning? Default is FALSE in which case an error is thrown.

`tab.count` Deprecated. Use `col.tableno`.

`use.rds` Deprecated - use `formats.read` instead. If provided (though not recommended), this will overwrite `formats.read`, and only formats `rds` and `csv` can be used.

Details

This function makes it very easy to collect the data from a Nonmem run.

A useful feature of this function is that it can automatically combine "input" data (the data read by Nonmem in $INPUT or $INFILE) with "output" data (tables written by Nonmem in $TABLE). There are two implemented methods for doing so. One (the default but not recommended) relies on interpretation of filter (IGNORE and ACCEPT) statements in $INPUT. This will work in most cases, and checks for consistency with Nonmem results. However, the recommended method is using a unique row identifier in both input data and at least one output data file (not a FIRSTONLY or LASTONLY table). Supply the name of this column using the col.row argument.

Limitations. A number of Nonmem features are not supported. Most of this can be overcome by using merge.by.row=TRUE. Incomplete list of known limitations:

character TIME If Nonmem is used to translate DAY and a character TIME column, TIME has to be available in an output table. NMscanData does not do the translation to numeric.

RECORDS The RECORDS option to limit the part of the input data being used is not searched for. Using merge.by.row=TRUE will work unaffectedly.

NULL The NULL argument to specify missing value string in input data is not respected. If delimited input data is read (as opposed to rds files), missing values are assumed to be represented by dots (.).

Value

A data set of class 'NMdata'.

See Also

Other DataRead: `NMreadCsv()`, `NMreadTab()`, `NMscanInput()`, `NMscanTables()`

Examples

```
## Not run:
res1 <- NMscanData(system.file("examples/nonmem/xgxr001.lst", package="NMdata"))

## End(Not run)
```

`NMscanInput` *Find and read input data and optionally translate column names according to the $INPUT section*

Description

This function finds and reads the input data based on a control stream file path. It can align the column names to the definitions in $INPUT in the control stream, and it can subset the data based on ACCEPT/IGNORE statements in $DATA. It supports a few other ways to identify the input data file than reading the control stream, and it can also read an rds or fst file instead of the delimited text file used by Nonmem.

Usage

```
NMscanInput(
  file,
  formats.read,
  file.mod,
  dir.data = NULL,
  file.data = NULL,
  apply.filters = FALSE,
  translate = TRUE,
  recover.cols = TRUE,
  details = TRUE,
  col.id = "ID",
  col.row,
  quiet,
  args.fread,
  invert = FALSE,
  as.fun,
  applyFilters,
  use.rds
)
```

Arguments

file	a .lst (output) or a .mod (input) control stream file. The filename does not need to end in .lst. It is recommended to use the output control stream because it reflects the model as it was run rather than how it is planned for next run. However, see file.mod and dir.data.
formats.read	Prioritized input data file formats to look for and use if found. Default is c("rds","csv") which means `rds` will be used if found, and `csv` if not. `fst` is possible too. Default can be modified using `NMdataConf()`.
file.mod	The input control stream file path. Default is to look for \"file\" with extension changed to .mod (PSN style). You can also supply the path to the file, or you can provide a function that translates the output file path to the input file path. If dir.data is missing, the input control stream is needed. This is because the .lst does not contain the path to the data file. The .mod file is only used for finding the data file. How to interpret the datafile is read from the .lst file. The default can be configured using NMdataConf. See dir.data too.
dir.data	The data directory can only be read from the control stream (.mod) and not from the output file (.lst). So if you only have the output file, use dir.data to tell in which directory to find the data file. If dir.data is provided, the .mod file is not used at all.
file.data	Specification of the data file path. When this is used, the control streams are not used at all.
apply.filters	If TRUE (default), IGNORE and ACCEPT statements in the Nonmem control streams are applied before returning the data.

translate	If TRUE (default), data columns are named as interpreted by Nonmem (in $INPUT). If data file contains more columns than mentioned in $INPUT, these will be named as in data file (if data file contains named variables).
recover.cols	recover columns that were not used in the Nonmem control stream? Default is TRUE. Can only be negative when translate=FALSE.
details	If TRUE, metadata is added to output. In this case, you get a list. Typically, this is mostly useful if programming up functions which behavior must depend on properties of the output. See details.
col.id	The name of the subject ID column. Optional and only used to calculate number of subjects in data. Default is modified by NMdataConf.
col.row	The name of the row counter column. Optional and only used to check whether the row counter is in the data.
quiet	Default is to inform a little, but TRUE is useful for non-interactive stuff.
args.fread	List of arguments passed to fread. Notice that except for "input" and "file", you need to supply all arguments to fread if you use this argument. Default values can be configured using NMdataConf.
invert	If TRUE, the data rows that are dismissed by the Nonmem data filters (ACCEPT and IGNORE) and only this will be returned. Only used if apply.filters is TRUE.
as.fun	The default is to return data as a data.frame. Pass a function (say tibble::as_tibble) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.
applyFilters	Deprecated - use apply.filters.
use.rds	Deprecated - use formats.read instead. If provided (though not recommended), this will overwrite formats.read, and only formats rds and csv can be used.

Details

Columns that are dropped (using DROP or SKIP in $INPUT) in the model will be included in the output.

It may not work if a column is dropped, and a new column is renamed to the same name. Say you have DV and CONC as the only two columns (not possible but illustrative), and in Nonmem you do DV=DROP DV. Not sure it will work in Nonmem, and it probably won't work in NMscanInput.

Value

A data set, class defined by 'as.fun'

See Also

Other DataRead: NMreadCsv(), NMreadTab(), NMscanData(), NMscanTables()

NMscanMultiple *Run NMscanData on multiple models and stack results*

Description

Useful function for meta analyses when multiple models are stored in one folder and can be read with NMscanData using the same arguments.

Usage

```
NMscanMultiple(files, dir, file.pattern, as.fun, ...)
```

Arguments

files	File paths to the models (control stream) to edit. See file.pattern too.
dir	The directory in which to find the models. Passed to list.files(). See file.pattern argument too.
file.pattern	The pattern used to match the filenames to read with NMscanData. Passed to list.files(). If dir is supplied and files is not (or is NULL), the default is ".*\.lst" which means all files ending in '.lst'. See dir argument too.
as.fun	The default is to return data as a data.frame. Pass a function (say tibble::as_tibble) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.
...	Additional arguments passed to NMscanData.

Value

All results stacked, class as defined by as.fun

Examples

```
## Not run:
res <- NMscanMultiple(dir=system.file("examples/nonmem", package="NMdata"),
file.pattern="xgxr01.*\\.lst",as.fun="data.table")
res.mean <- res[,.(meanPRED=exp(mean(log(PRED)))),by=.(model,NOMTIME)]
library(ggplot2)
ggplot(res.mean,aes(NOMTIME,meanPRED,colour=model))+geom_line()

## End(Not run)
```

NMscanTables *Find and read all output data tables in Nonmem run*

Description

Find and read all output data tables in Nonmem run

Usage

```
NMscanTables(
  file,
  as.fun,
  quiet,
  col.nmrep = TRUE,
  col.tableno = FALSE,
  col.id = "ID",
  col.row,
  details,
  skip.absent = FALSE,
  meta.only = FALSE
)
```

Arguments

file	the Nonmem file to read (normally .mod or .lst)
as.fun	The default is to return data as a data.frame. Pass a function (say tibble::as_tibble) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.
quiet	The default is to give some information along the way on what data is found. But consider setting this to TRUE for non-interactive use. Default can be configured using NMdataConf.
col.nmrep	col.nmrep If tables are repeated, include a counter? It does not relate to the order of the $TABLE statements but to cases where a $TABLE statement is run repeatedly. E.g., in combination with the SUBPROBLEMS feature in Nonmem, it is useful to keep track of the table (repetition) number. If col.nmrep is TRUE, this will be carried forward and added as a column called NMREP. This is default behavior when more than one $TABLE repetition is found in data. Set it to a different string to request the column with a different name. The argument is passed to NMscanTables.
col.tableno	Nonmem includes a counter of tables in the written data files. These are often not useful. However, if col.tableno is TRUE (not default), this will be carried forward and added as a column called NMREP. Even if NMREP is generated by NMscanTables, it is treated like any other table column in meta (?NMinfo) data.
col.id	name of the subject ID column. Used for calculation of the number of subjects in each table.

col.row	The name of the row counter column. Optional and only used to check whether the row counter is in the data.
details	If TRUE, metadata is added to output. In this case, you get a list. Typically, this is mostly useful if programming up functions which behavior must depend on properties of the output.
skip.absent	Skip missing output table files with a warning? Default is FALSE in which case an error is thrown.
meta.only	If TRUE, tables are not read, only a table is returned showing what tables were found and some available meta information. Notice, not all meta information (e.g., dimensions) are available because the tables need to be read to derive that.

Value

A list of all the tables as data.frames. If details=TRUE, this is in one element, called data, and meta is another element. If not, only the data is returned.

See Also

Other DataRead: NMreadCsv(), NMreadTab(), NMscanData(), NMscanInput()

Examples

```
tabs1 <- NMscanTables(system.file("examples/nonmem/xgxr001.lst", package="NMdata"))
```

NMstamp *stamp a dataset or any other object*

Description

Dataset metadata can be valuable, eg. by tracing an archived dataset back to the code that generated it. The metadata added by NMstamp can be accessed using the function NMinfo.

Usage

```
NMstamp(data, script, time = Sys.time(), ...)
```

Arguments

data	The dataset to stamp.
script	path to the script where the dataset was generated.
time	the time stamp to attach. Default is to use cpu clock.
...	other named metadata elements to add to the dataset. Example: Description="PK data for phase 1 trials in project".

Details

NMstamp modifies the meta data by reference. See example.

Value

data with meta data attached. Class unchanged.

See Also

NMinfo

Other DataCreate: NMorderColumns(), NMwriteData(), addTAPD(), findCovs(), findVars(), flagsAssign(), flagsCount(), mergeCheck(), tmpcol()

Examples

```
x=1
NMstamp(x,script="example.R",description="Example data")
NMinfo(x)
```

NMwriteData *Write dataset for use in Nonmem (and R)*

Description

Instead of trying to remember the arguments to pass to write.csv, use this wrapper. It tells you what to write in $DATA and $INPUT in Nonmem, and it (additionally) exports an rds file as well which is highly preferable for use in R. It never edits the data before writing the datafile. The filenames for csv, rds etc. are derived by replacing the extension to the filename given in the file argument.

Usage

```
NMwriteData(
  data,
  file,
  formats.write = c("csv", "rds"),
  script,
  args.stamp,
  args.fwrite,
  args.rds,
  args.RData,
  args.write_fst,
  quiet,
  args.NMgenText,
  csv.trunc.as.nm = FALSE,
  genText = TRUE,
  save = TRUE,
  write.csv,
  write.rds,
  write.RData,
  nm.drop,
  nmdir.data,
```

```
  col.flagn,
  nm.rename,
  nm.copy,
  nm.capitalize,
  allow.char.TIME
)
```

Arguments

data	The dataset to write to file for use in Nonmem.
file	The file to write to. The extension (everything after and including last ".") is dropped. csv, rds and other standard file name extensions are added.
formats.write	character vector of formats.write. Default is c("csv","rds"). "fst" is possible too. Default can be modified with NMdataConf().
script	If provided, the object will be stamped with this script name before saved to rds or RData. See ?NMstamp.
args.stamp	A list of arguments to be passed to NMstamp.
args.fwrite	List of arguments passed to fwrite. Notice that except for "x" and "file", you need to supply all arguments to fwrite if you use this argument. Default values can be configured using NMdataConf.
args.rds	A list of arguments to be passed to saveRDS.
args.RData	A list of arguments to be passed to save. Please note that writing RData is deprecated.
args.write_fst	An optional list of arguments to be passed to write_fst.
quiet	The default is to give some information along the way on what data is found. But consider setting this to TRUE for non-interactive use. Default can be configured using NMdataConf.
args.NMgenText	List of arguments to pass to NMgenText - the function that generates text suggestion for INPUT and DATA sections in the Nonmem control stream. You can use these arguments to get a text suggestion you an use directly in Nonmem - and NMwriteSection can even update multiple Nonmem control streams based on the result. This will update your control streams to match your new data file with just one command.
csv.trunc.as.nm	If TRUE, csv file will be truncated horizontally (columns will be dropped) to match the $INPUT text generated for Nonmem (genText must be TRUE for this option to be allowed). This can be a great advantage when dealing with large datasets that can create problems in parallellization. Combined with write.rds=TRUE, the full data set will still be written to an rds file, so this can be used when combining output and input data when reading model results. This is done by default by NMscanData. This means writing a lean (narrow) csv file for Nonmem while keeping columns of non-numeric class like character and factor for post-processing.
genText	Run and report results of NMgenText? Default is TRUE. You may want to disable this if data set is not for Nonmem.

save	Save defined files? Default is TRUE. If a variable is used to control whether a script generates outputs (say writeOutputs=TRUE/FALSE), if you use save=writeOutputs to comply with this.
write.csv	Write to csv file? Deprecated, use 'formats.write' instead.
write.rds	write an rds file? Deprecated, use 'formats.write' instead.
write.RData	Deprecated and not recommended - will be removed. RData is not a adequate format for a dataset (but is for environments). Please use write.rds instead.
nm.drop	Deprecated, use args.NMgenText=list(drop=c("column")) instead.
nmdir.data	Deprecated, use args.NMgenText=list(dir.data="your/path") instead.
col.flagn	Name of a numeric column with zero value for rows to include in Nonmem run, non-zero for rows to skip. The argument is only used for generating the proposed $DATA text to paste into the Nonmem control stream. To skip this feature, use col.flagn=NULL.
nm.rename	Deprecated, use args.NMgenText=list(rename=c(newname="existing")) instead.
nm.copy	Deprecated, use args.NMgenText=list(copy=c(newname="existing")) instead.
nm.capitalize	Deprecated, use args.NMgenText=list(capitalize=TRUE) instead.
allow.char.TIME	Deprecated, use args.NMgenText=list(allow.char.TIME=TRUE) instead.

Details

When writing csv files, the file will be comma-separated. Because Nonmem does not support quoted fields, you must avoid commas in character fields. An error is returned if commas are found in strings.

The user is provided with text to use in Nonmem. This lists names of the data columns. Once a column is reached that Nonmem will not be able to read as a numeric and column is not in nm.drop, the list is stopped. Only exception is TIME which is not tested for whether character or not.

Value

Text for inclusion in Nonmem control stream, invisibly.

See Also

Other DataCreate: NMorderColumns(), NMstamp(), addTAPD(), findCovs(), findVars(), flagsAssign(), flagsCount(), mergeCheck(), tmpcol()

NMwriteSection *Replace ($)sections of a Nonmem control stream*

Description

Just give the section name, the new lines and the file path, and the "$section", and the input to Nonmem will be updated.

Usage

```
NMwriteSection(
  files,
  file.pattern,
  dir,
  section,
  newlines,
  list.sections,
  location = "replace",
  newfile,
  backup = TRUE,
  blank.append = TRUE,
  data.file,
  write = TRUE,
  quiet,
  simplify = TRUE
)
```

Arguments

files	File paths to the models (control stream) to edit. See file.pattern too.
file.pattern	Alternatively to files, you can supply a regular expression which will be passed to list.files as the pattern argument. If this is used, use dir argument as well. Also see data.file to only process models that use a specific data file.
dir	If file.pattern is used, dir is the directory to search in.
section	The name of the section to update with or without "$". Example: section="EST" or section="$EST" to edit the sections starting by $EST. Section specification is not case-sensitive. See ?NMreadSection too.
newlines	The new text (including "$SECTION"). Better be broken into lines in a character vector since this is simply past to writeLines.
list.sections	Named list of new sections, each element containing a section. Names must be section names, contents of each element are the new section lines for each section.
location	In combination with ‘section‘, this determines where the new section is inserter. Possible values are "replace" (default), "before", "after", "first", "last".

newfile	path and filename to new run. If missing, the original file (from files or file.pattern) is overwritten (see the backup option below). If NULL, output is returned as a character vector rather than written.
backup	In case you are overwriting the old file, do you want to backup the file (to say, backup_run001.mod)?
blank.append	Append a blank line to output?
data.file	Use this to limit the scope of models to those that use a specific input data data file. The string has to exactly match the one in $DATA or $INFILE in Nonmem.
write	Default is to write to file. If write=FALSE, NMwriteSection returns the resulting input.txt without writing it. to disk? Default is FALSE.
quiet	The default is to give some information along the way on what data is found. But consider setting this to TRUE for non-interactive use. Default can be configured using NMdataConf.
simplify	If TRUE (default) and only one file is edited, the resulting rows are returned directly. If more than one file is edited, the result will always be a list with one element per file.

Details

The new file will be written with unix-style line endings.

Value

The new section text is returned. If write=TRUE, this is done invisibly.

See Also

Other Nonmem: NMapplyFilters(), NMextractText(), NMgenText(), NMreadSection(), NMreplaceDataFile()

Examples

```
newlines <- "$EST POSTHOC INTERACTION METHOD=1 NOABORT PRINT=5 MAXEVAL=9999 SIG=3"
NMwriteSection(files=system.file("examples/nonmem/xgxr001.mod", package = "NMdata"),
section="EST", newlines=newlines,newfile=NULL)
## Not run:
text.nm <- NMwriteData(data)
NMwriteSection(dir="nonmem",
               file.pattern="^run.*\\.mod",
               list.sections=text.nm["INPUT"])

## End(Not run)
```

print.summary_NMdata *print method for NMdata summaries*

Description

print method for NMdata summaries

Usage

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

Arguments

x	The summary object to be printed. See ?summary.NMdata
...	Arguments passed to other print methods.

Value

NULL (invisibly)

renameByContents *Rename columns matching properties of data contents*

Description

For instance, lowercase all columns that Nonmem cannot interpret (as numeric).

Usage

```
renameByContents(data, fun.test, fun.rename, invert.test = FALSE, as.fun)
```

Arguments

data	data.frame in which to rename columns
fun.test	Function that returns TRUE for columns to be renamed.
fun.rename	Function that takes the existing column name and returns the new one.
invert.test	Rename those where FALSE is returned from fun.test.
as.fun	The default is to return data as a data.frame. Pass a function (say tibble::as_tibble) in as.fun to convert to something else. If data.tables are wanted, use as.fun="data.table". The default can be configured using NMdataConf.

Value

data with (some) new column names. Class as defined by as.fun.

Examples

```
pk <- readRDS(file=system.file("examples/data/xgxr2.rds",package="NMdata"))
pk[,trtact:=NULL]
pk <- renameByContents(data=pk,
                       fun.test = NMisNumeric,
                       fun.rename = tolower,
                       invert.test = TRUE)
## Or append a "C" to the same column names
pk <- readRDS(file=system.file("examples/data/xgxr2.rds",package="NMdata"))
pk[,trtact:=NULL]
pk <- renameByContents(data=pk,
                       fun.test = NMisNumeric,
                       fun.rename = function(x)paste0(x,"C"),
                       invert.test = TRUE)
```

---

summary.NMdata *summary method for NMdata objects*

Description

summary method for NMdata objects

Usage

```
## S3 method for class 'NMdata'
summary(object, ...)
```

Arguments

object	An NMdata object (from NMscanData).
...	Only passed to the summary generic if object is missing NMdata meta data (this should not happen anyway).

Details

The subjects are counted conditioned on the nmout column. If only id-level output tables are present, there are no nmout=TRUE rows. This means that in this case it will report that no IDs are found in output. The correct statement is that records are found for zero subjects in output tables.

Value

A list with summary information on the NMdata object.

unNMdata *Remove NMdata class and discard NMdata meta data*

Description

Remove NMdata class and discard NMdata meta data

Usage

```
unNMdata(x)
```

Arguments

x	An 'NMdata' object.

Value

x stripped from the 'NMdata' class

Index