Package 'NMdata'

August 9, 2021

Type Package

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

Version 0.0.9

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. Helps with trivial but tedious tasks and tries to identify errors to save time on debugging. Implemented in 'data.table', but easily integrated with 'base' and 'tidyverse'.

License MIT + file LICENSE

RoxygenNote 7.1.1

Depends R (>= 3.0.0)

Imports data.table

Suggests testthat, knitr, rmarkdown, ggplot2, tibble, covr

VignetteBuilder knitr

Encoding UTF-8

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

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

NeedsCompilation no

Author Philip Delff [aut, cre]

Repository CRAN

Date/Publication 2021-08-09 20:40:02 UTC

R topics documented:

checkColRow *check that col.row is not edited in Nonmem control stream*

Description

In order to safely merge by a unique row identifier, that row identifier must not be edited from input to output. checkColRow helps checking that based on the control stream.

Usage

```
checkColRow(col.row, file)
```

Arguments

col.row	The name of the unique row identifier (say "ROW").
file	a list file or input control stream file path.

Value

TRUE if no issues found

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(
  ...,
  keepNames = TRUE,
  testEqual = FALSE,
  diff.only = TRUE,
  fun.class = base::class,
  quiet,
  as.fun = NULL
)
```

Arguments

`...`	objects which element names to compare
`keepNames`	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.
`testEqual`	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 compitable enough (say numeric and integer), at 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.
`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 un 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.

Details

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

Value

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

See Also

Other DataWrangling: `dims()`

`dims` *Get dimensions of multiple objects*

Description

Get dimensions of multiple objects

Usage

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

Arguments

...	data sets
list.data	As alternative to ..., you can supply the data sets in a list here.
keepNames	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 un 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

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

See Also

Other DataWrangling: `compareCols()`

---

egdt *Expand grid of data.tables*

Description

Expand grid of data.tables

Usage

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

Arguments

dt1	a data.table.
dt2	another data.table.

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))
```

---

filePathSimple *Clean and standardize file system paths*

Description

Use this to tidy up paths. Combines pieces of a path like file.path(). The function is intended to return a canonical path format, i.e. paths that can be compared by simple string comparisson. Redundant /'s removed. normalizePath is used to possibly shorten path.

Usage

```
filePathSimple(...)
```

Arguments

...	additional arguments passed to file.path().

Value

A (character) file path

findCovs *Extract columns that do not vary within variables in data*

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, 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").
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 configued using NMdataConf.

Value

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

See Also

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

Examples

```
dat <- NMscanData(system.file("examples/nonmem/xgxr001.lst", package = "NMdata"))
### very common use
findCovs(dat,by="ID")
### Without an ID column we get non-varying columns
findCovs(dat)
```

`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, 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").
`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 configued 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()`, `findCovs()`, `flagsAssign()`, `flagsCount()`, `mergeCheck()`, `tmpcol()`

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,
  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. 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.
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(), findCovs(), findVars(), flagsCount(), mergeCheck(), tmpcol()

Examples

```
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,col.flagn="flagn",col.flagc="flagc")
unique(pk[,c("flagn","flagc","BLQ")])
flagsCount(pk[EVID==0],dt.flags,col.flagn="flagn",col.flagc="flagc")
```

`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,
  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.
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 number of subjects in N.discarded mode can be misunderstood. 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(), findCovs(), findVars(), flagsAssign(), mergeCheck(), tmpcol()

Examples

```
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,col.flagn="flagn",col.flagc="flagc")
unique(pk[,c("flagn","flagc","flagn")])
flagsCount(pk[EVID==0],dt.flags,col.flagn="flagn",col.flagc="flagc")
```

---

fnExtension *Change file name extension*

Description

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

Usage

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

Arguments

fn	file name
ext	new file name extension

Value

A text string

Examples

```
fnExtension("file.lst",".mod")
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

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

Description

This function is a useful wrapper for merges where df1 will be extended with columns from df2, i.e. all rows in df1 are retained, and no new rows can be created. For this very common type of (simple) merges, mergeCheck does the merge and ensures that exactly this and nothing else happened. Notice, mergeCheck passes the hard work to merge.data.table, the contributions lies in checking that the results are consistent with the simple merge described above.

Usage

```
mergeCheck(
  df1,
  df2,
  by,
  as.fun = NULL,
  fun.commoncols = base::warning,
  ncols.expect,
  track.msg = FALSE,
  ...
)
```

Arguments

df1	A data.frame with the number of rows must should be obtained from the merge. The resulting data.frame will be ordered like df1.
df2	A data.frame that will be merged onto df1.
by	The column(s) to merge by. Character string (vector). Must be supplied.
as.fun	The default is to return a data.table if df1 is a data.table and return a data.frame in all other cases. Pass a function in as.fun to convert to something else.
fun.commoncols	If common columns are found in df1 and df2, 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 df1. So if ncols.expect=1, the resulting data must have exactly one column more than df1 - 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.
...	additional arguments passed to merge. If all is among them, an error will be returned.

Details

Besides merging and checking rows, mergeCheck makes sure the order in df1 is retained in the resulting data. 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 df1 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.

Value

a data.frame resulting from merging df1 and df2. Class as defined by as.fun.

See Also

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

messageWrap *Pretty wrapping of lines in NMdata vignettes*

Description

Pretty wrapping of lines in NMdata vignettes

Usage

```
messageWrap(
  ...,
  fun.msg = message,
  prefix = "\n",
  initial = "",
  width,
  track.msg = FALSE
)
```

Arguments

`...`	parameters to pass to strwrap
`fun.msg`	The function to pass the text through. Typically, message, warning, or stop. If NULL, nothing will happen, and NULL is invisibly returned.
`prefix`	Passed to strwrap. Default is "\n".
`initial`	Passed to strwrap. Default is an empty string.
`width`	Passed to strwrap. Default is 80.
`track.msg`	If TRUE, the name of the function throwing the message/warning/error is mentioned. This is not default but useful when using function inside other functions.

Value

Nothing.

NMapplyFilters *Translate filters in Nonmem and apply to data*

Description

Translate filters in Nonmem and apply to data

Usage

```
NMapplyFilters(data, file, text, lines, invert = FALSE, as.fun, quiet)
```

Arguments

data	An input data object. Could be read with NMreadCsv or NMscanInput.
file	Path to mod/lst file. Only one of file, text, or lines to be given. See ?NMreadSection for understanding when to use, file, text, or lines.
text	The mod/lst as characters.
lines	The mod/lst as character, line by line.
invert	Invert the filters? This means read what Nonmem would disregard, and disregard what Nonmem would read.
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	Don't report information along the way if no warnings or errors. Default is FALSE.

Details

This is not bulletproof. Nested conditions are not supported altogether.

Value

data with filters applied

See Also

Other Nonmem: NMextractText(), NMreadSection(), NMwriteData(), NMwriteSection()

NMcheckColnames *Compare $DATA to column names in input data*

Description

Misspecification of column names in $DATA are a common source of problems with Nonmem models, and one of the first things to check when seemingly inexplainable 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

---

`NMcode2R` *Translate Nonmem $PK, $PRED sections or other Nonmem code to R code*

Description

Translate Nonmem $PK, $PRED sections or other Nonmem code to R code

Usage

```
NMcode2R(text)
```

Arguments

`text`	the Nonmem code.

Details

You probably want to run this on text obtained by using the NMreadSection function.

Value

R code as text

`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 behaviour. 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(...)
```

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.

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.
- 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).
- 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.
- 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.

- 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 Affects NMscanData and NMscanInput.

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)
```

NMdataConfOptions *Get NMdataConf parameter properties*

Description

Get NMdataConf parameter properties

Usage

```
NMdataConfOptions(name)
```

Arguments

name	Optionally, a single parameter name (say "as.fun").

Value

If name is provided, a list representing one argument, otherwise a list with an element for each argument that can be customized using NMdataConf.

NMdataDecideOption *Determine active parameter value based on argument and NMdataConf setting*

Description

Determine active parameter value based on argument and NMdataConf setting

Usage

```
NMdataDecideOption(name, argument)
```

Arguments

name	The name of the parameter, say "as.fun"
argument	The value to pass. If missing or NULL, the value returned by NMdataConf/NMdataGetOption will typically be used.

Value

Active argument value.

NMdataGetOption *Look up default configuration of an argument*

Description

Look up default configuration of an argument

Usage

```
NMdataGetOption(...)
```

Arguments

...	argument to look up. Only one argument can be looked up.

Value

The value active in configuration

NMdataOperations *Basic arithmetic*

Description

Basic arithmetic

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 clases (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'.

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

Description

Extract the data file used in a control stream

Usage

```
NMextractDataFile(file, dir.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.

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",
  keepEmpty = FALSE,
  keepName = TRUE,
  keepComments = TRUE,
  asOne = TRUE,
  simplify = TRUE,
```

```
  cleanSpaces = FALSE,
  type = "mod",
  linesep = "\n"
)
```

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 seperator 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 denoter as a string compatible with regular expressions. "\$" 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.
keepEmpty	Keep empty lines in output? Default is FALSE.
keepName	Keep the section name in output (say, "$PROBLEM") Default is TRUE. It can only be FALSE, if return"idx".
keepComments	Keep comment lines?
asOne	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.
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.
cleanSpaces	If TRUE, leading and trailing are removed, and multiplied succeeding white spaces are reduced to single white spaces.
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.

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(), NMreadSection(), NMwriteData(), NMwriteSection()

Examples

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

`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)
```

Arguments

x	The vector to check Don't export

Value

TRUE or FALSE

`NMorderColumns` *Order columns in dataset for use in Nonmem.*

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.nomtime,
  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.nomtime`	The name of the column containing nominal time. If given, it will put the column quite far left, just after row counter and ID. Default value is NOMTIME and can be configured 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.ntime - "Nominal time.
- "first - "user-specified first columns
- "not editable - "Standard Nonmem columns: TIME, EVID, CMT, AMT, RATE, 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()`, `findCovs()`, `findVars()`, `flagsAssign()`, `flagsCount()`, `mergeCheck()`, `tmpcol()`

NMreadCsv *read csv data that is written to be read by nonmem*

Description

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

Usage

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

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.

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()

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",
  keepEmpty = FALSE,
  keepName = TRUE,
  keepComments = TRUE,
  asOne = TRUE,
  simplify = TRUE,
  cleanSpaces = FALSE,
  ...
)

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 seperator with the linesep argument (handled by NMextractText). Use only one of file, lines, and text.
section	The name of section to extract. Examples: "INPUT", "PK", "TABLE", etc.
return	If "text", plain text lines are returned. If "idx", matching line numbers are returned. "text" is default.
keepEmpty	Keep empty lines in output? Default is FALSE.
keepName	Keep the section name in output (say, "$PROBLEM") Default is TRUE. It can only be FALSE, if return"idx".
keepComments	Keep comment lines?
asOne	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.

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.
cleanSpaces	If TRUE, leading and trailing are removed, and multiplied succeeding white spaces are reduced to single white spaces.
...	Additional arguments passed to NMextractText

Value

character vector with extracted lines.

Functions

- NMgetSection: Old function name for NMreadSection

See Also

Other Nonmem: NMapplyFilters(), NMextractText(), NMwriteData(), 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, tab.count = TRUE, quiet, as.fun, ...)
```

Arguments

file	path to NONMEM table file
tab.count	Nonmem includes a counter of tables in the written data files. These are often not useful. However, if tab.count is TRUE (default), a counter of tables is included as a column called TABLENO. Just notice, the table numbers in TABLENO are just cumulatively counting the number of tables reported in the file. TABLENO is not true to the actual table number as given by Nonmem.

quiet	logical stating whether or not information is prignted 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()

NMscanData *Automatically find Nonmem 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,
  translate.input = TRUE,
  quiet,
  use.rds,
  args.fread,
```

```
  as.fun,
  col.id = "ID",
  modelname,
  col.model,
  col.nmout,
  tab.count = FALSE,
  order.columns = TRUE,
  check.time
)
```

Arguments

file	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 immitate 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. 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.
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.
use.rds	If an rds file is found with the exact same name (except for .rds instead of say .csv) as the input data file mentioned in the Nonmem control stream, should this

be used instead? The default is yes, and NMwriteData will create this by default too. Default can be configured 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.

`tab.count` Nonmem includes a counter of tables in the written data files. These are often not useful. Especially for NMscanData output it can be meaningless because multiple tables can be combined so this information is not unique across those source tables. However, if tab.count is TRUE (not default), this will be carried forward and added as a column called TABLENO. 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 alphebetically. 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 transfered 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.

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 TIMEIf Nonmem is used to translate DAY and a charater TIME column, TIME has to be available in an output table. NMscanData does not do the translation to numeric.
- RECORDSThe RECORDS option to limit the part of the input data being used is not searched for. Using merge.by.row=TRUE will work unaffectedly.
- NULLThe 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()`

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

Description

Based on a nonmem run (lst and/or mod file), this function finds the input data and reads it. It reads the data like the nonmem run by applying DROP/SKIP arguments and alternative naming of columns in the nonmem run.

Usage

```
NMscanInput(
  file,
  use.rds,
  file.mod,
  dir.data = NULL,
  applyFilters = FALSE,
  translate = TRUE,
  details = TRUE,
  col.id = "ID",
  col.row,
  quiet,
  args.fread,
```

```
  invert = FALSE,
  as.fun
)
```

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.
use.rds	If an rds file is found with the exact same name (except for .rds instead of say .csv) as the text file mentioned in the Nonmem control stream, should this be used instead? The default is yes, and NMwriteData will create this by default too.
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.
applyFilters	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).
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 applyFilters 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.

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()`

NMscanTables *read all output data tables in nonmem run*

Description

read all output data tables in nonmem run

Usage

```
NMscanTables(
  file,
  details = FALSE,
  as.fun,
  quiet,
  tab.count = FALSE,
  col.id = "ID",
  col.row
)
```

Arguments

`file`	the nonmem file to read (normally .mod or .lst)
`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.
`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.

tab.count	Nonmem includes a counter of tables in the written data files. These are often not useful. However, if tab.count is TRUE (not default), this will be carried forward and added as a column called TABLENO.
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.

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()

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 ph1 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(), findCovs(), findVars(), flagsAssign(), flagsCount(), mergeCheck(), tmpcol()

Examples

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

`NMtransInp` *translate the column names according to the $INPUT section of a control stream*

Description

translate the column names according to the $INPUT section of a control stream

Usage

```
NMtransInp(data, file, translate = TRUE)
```

Arguments

`data`	the data to translate
`file`	the list file or control stream
`translate`	logical. Do translation according to Nonmem code or not? If not, an overview of column names in data and in Nonmem code is still returned with the data.

Value

data with column names translated as specified by nonmem control stream. Class same as for 'data' argument.

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 or Rdata 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,
  write.csv = TRUE,
  write.rds = write.csv,
  write.RData = FALSE,
  script,
  args.stamp,
  args.fwrite,
  args.rds,
  args.RData,
  nm.drop,
  nmdir.data,
  col.flag = "FLAG",
  nm.rename,
  nm.copy,
  nm.capitalize = FALSE,
  allow.char.TIME = TRUE,
  quiet
)
```

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.
write.csv	Write to csv file?
write.rds	write an rds file?
write.RData	In case you want to save to .RData object. Not recommended. Use write.rds instead.
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.

`nm.drop` Only used for generation of proposed text for Nonmem control stream. Columns to drop in Nonmem $DATA. This has two implications. One is that the proposed $DATA indicates =DROP after the given column names. The other that in case it is a non-numeric column, succeeding columns can still be included.

`nmdir.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.

`col.flag` 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 text to paste into the Nonmem control stream. To skip this feature, use col.flag=NULL.

`nm.rename` For the $DATA text proposal only. If you plan 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 nm.rename=c(BWBASE="BBW"). The result is different from nm.copy since the nm.copy syntax is only allowed by Nonmem for certain standard column names such as DV.

`nm.copy` For the $DATA text proposal only. If you plan to rename columns in Nonmem $DATA, NMwriteData can adjust the suggested $DATA text. If you plan to use CONC as DV in Nonmem, you can include nm.rename=c(DV="CONC").

`nm.capitalize` For the $DATA text proposal only. If TRUE, the suggested text for Nonmem will only contain capital letters in column names.

`allow.char.TIME`
For the $DATA 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.

`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

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

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(
  file,
  section,
  newlines,
  list.sections,
  newfile,
  backup = TRUE,
  blank.append = TRUE,
  write = TRUE
)
```

Arguments

file	File path to the model (control stream) to edit.
section	The name of the section to update. Example: section="EST" to edit the sections starting by $EST. See NMreadSection
newlines	The new text (including "$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.
newfile	path to new run. If missing, path is used. If NULL, output is returned 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?
write	Default is to write to file. If write=FALSE, NMwriteSection returns the resulting input.txt without writing it. to disk? Default is FALSE.

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(), NMreadSection(), NMwriteData()

Examples

```
newlines <- "$EST POSTHOC INTERACTION METHOD=1 NOABORT PRINT=5 MAXEVAL=9999 SIG=3"
NMwriteSection(file=system.file("examples/nonmem/xgxr001.mod", package = "NMdata"),
section="EST", newlines=newlines,newfile=NULL)
```

---

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 print.

---

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.

---

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.

`tmpcol` *generate a name for a new data column that is not already in use.*

Description

generate a name for a new data column that is not already in use.

Usage

```
tmpcol(
  data,
  names = NULL,
  base = "atmpcol999",
  max.it = 100,
  prefer.plain = TRUE
)
```

Arguments

`data`	The dataset to find a new element name for
`names`	Character vector of names that must not be matched. Only one of data and names can be supplied.
`base`	The base name of the new element. A number will appended to this string that will ensure that the new element name is not already in use.
`max.it`	Maximum number of iterations on element name.
`prefer.plain`	If base isn't in use already, use it without a digit appended?

Value

A character string

See Also

make.names

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

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

writeNMinfo *Do the actual writing of meta data*

Description

Do the actual writing of meta data

Usage

```
writeNMinfo(data, meta, append = FALSE, byRef = TRUE)
```

Arguments

data	A data set
meta	The meta data to attach
append	If FALSE, the existing meta data will be removed. If TRUE, metadata will be appended to existing metadata. However, this will not work recursively.
byRef	Should always be TRUE.

Value

The data with meta data attached

Index