Package 'tcpl'

May 20, 2018

Title ToxCast Data Analysis Pipeline

Version 1.4.3

Description A set of tools for processing and modeling high-throughput and high-content chemical screening data. The package was developed for the the chemical screening data generated by the US EPA ToxCast program, but can be used for diverse chemical screening efforts.

URL http://www.github.com/daynefiler/tcpl/

Depends R (>= 3.2.0), data.table (>= 1.9.4)

Imports DBI, RMySQL, RSQLite, numDeriv, RColorBrewer, utils, stats, methods, graphics, grDevices

Suggests roxygen2

License GPL-2

LazyData true

RoxygenNote 6.0.1

NeedsCompilation no

Author Dayne L Filer [aut, cre],
Parth Kothiya [ctb],
Woodrow R Setzer [ctb],
Richard S Judson [ths],
Matthew T Martin [ctb, ths]

Maintainer Dayne L Filer <dayne.filer@gmail.com>

Repository CRAN

Date/Publication 2018-05-20 14:37:56 UTC

R topics documented:

`blineShift` *Shift the baseline to 0*

Description

`blineShift` Takes in dose-response data and shifts the baseline to 0 based on the window.

Usage

```
blineShift(resp, logc, wndw)
```

Arguments

`resp`	Numeric, the response values
`logc`	Numeric, the log10 concentration values
`wndw`	Numeric, the threshold window

Value

A numeric vector containing the shifted response values

Note

This function is not exported and is not inteded to be used by the user.

See Also

`mc3_mthds`, `mc3`

`Configure functions` *Functions for configuring the tcpl package*

Description

These functions are used to configure the tcpl settings.

Usage

```
tcplConf(drvr = NULL, user = NULL, pass = NULL, host = NULL,
  db = NULL)

tcplConfDefault()

tcplConfList(show.pass = FALSE)

tcplConfLoad(list.new = TRUE)

tcplConfReset()

tcplConfSave()
```

Arguments

drvr	Character of length 1, which database driver to use
user	Character of length 1, the database server username
pass	Character of length 1, the database server password
host	Character of length 1, the database server
db	Character of length 1, the name of the tcpl database
show.pass	Logical, should the password be returned
list.new	Logical of length 1, should the new settings be printed?

Details

Currently, the tcpl package only supports the "MySQL" and "SQLite" database drivers.

The settings can be stored in a configuration file to make the using the package more user-friendly. To create the configuration file, the user must first create a system environment variable ('TCPL_CONF') that points to to the file. There is more information about system environment variables in `Startup` and `Sys.getenv`. Briefly, the user needs to modify the '.Renviron' file in their home directory. If the file does not exist, create it, and add the following line:

TCPL_CONF=path/to/confFile.conf

Here 'path/to/confFile.conf' can be any path to a file. One suggestion would be to include .tcplConf in the home directory, eg. TCPL_CONF=~/.tcplConf. Note, '~' may not indicate the home directory on every operating system. Once the environment variable is added, the user can change the settings using `tcplConf`, then save the settings to the file given by the TCPL_CONF environment variable running `tcplConfSave()`.

`tcplConf` changes `options` to set the tcpl-specific options, most importantly to configure the connection to the tcpl databases. `tcplConf` will only change non-null values, and can be used to change a single value if needed.

`tcplConfSave` modifies the configuration file to reflect the current tcpl settings.

`tcplConfList` lists the values assigned to the tcpl global options.

`tcplConfLoad` updates the tcpl settings to reflect the current configuration file.

`tcplConfDefault` changes the `options` to reflect the default settings for the example SQLite database, but does not alter the configuration file.

`tcplConfReset` is used to generate the initial configuration script, and can be used to reset or regenerate the configuration script by the user.

`flareFunc` *Calculate the weighted mean of a square to detect plate flares*

Description

`flareFunc` calculates the weighted mean of square regions to detect plate flares.

Usage

```
flareFunc(val, coli, rowi, apid, r)
```

Arguments

`val`	Numeric, the well values
`coli`	Integer, the well column index
`rowi`	Integer, the well row index
`apid`	Character, the assay plate id
`r`	Integer, the number of wells from the center well (in one direction) to make the square

See Also

`MC6_Methods`, `Method functions`, `mc6`

`Hill model utilites` *Functions to solve the Hill model*

Description

These functions solve for Hill model parameters.

Usage

```
tcplHillACXX(XX, tp, ga, gw, bt = 0)

tcplHillConc(val, tp, ga, gw, bt = 0)

tcplHillVal(logc, tp, ga, gw, bt = 0)
```

Arguments

XX	Numeric, the activity level (percentage of the top value)
tp	Numeric, the top value from the Hill model
ga	Numeric, the logAC50 value from the Hill model
gw	Numeric, the Hill coefficient from the Hill model
bt	Numierc, the bottom value from the Hill model
val	Numeric, the activity value
logc	Numeric, the log concentration

Details

tcplHillVal computes the value of the Hill model for a given log concentration.

tcplHillACXX computes the activity concentration for a Hill model for a given activity level.

tcplHillConc computes the Hill model concentration for a given value.

Examples

```
## The following code gives examples for a Hill model with a top of 50,
## bottom of 0, AC50 of 1 and Hill coefficient of 1.
## tcplHillVal calculates activity value given a concentration. tcplHillVal
## will return the tp/2 when logc equals ga:
tcplHillVal(logc = 1, tp = 50, ga = 1, gw = 1, bt = 0)

## Here, tcplHillConc returns the concentration where the value equals 20
tcplHillConc(val = 20, tp = 50, ga = 1, gw = 1, bt = 0)

## Note how this differs from tcplHillACXX:
tcplHillACXX(XX = 20, tp = 50, ga = 1, gw = 1, bt = 0)

## tcplHillACXX is based on the top value and allows the user to calculate
## specifc activity concentrations based on a percentage of the top value

## For example, we can calculate the value for the concentration 0.25, then
## use that value to check the other two functions.

value <- tcplHillVal(logc = 0.25, tp = 50, ga = 1, gw = 1, bt = 0)
c1 <- tcplHillConc(val = value, tp = 50, ga = 1, gw = 1, bt = 0)
c2 <- tcplHillACXX(XX = value/50*100, tp = 50, ga = 1, gw = 1, bt = 0)
all.equal(0.25, c1, c2)

## Notice, the value had to be transformed to a percentage of the top value
## when using tcplHillACXX
```

interlaceFunc *Calculate the weighted mean of a square to detect interlace effect*

Description

interlaceFunc calculates the distance weighted mean of square regions from a 384-well plate that is interlaced onto a 1536 well plate to detect non-random signals coming from the source plate

Usage

```
interlaceFunc(val, intq, coli, rowi, apid, r)
```

Arguments

val	Numeric, the well values
intq	Numeric, interlace quadrant
coli	Integer, the well column index
rowi	Integer, the well row index
apid	Character, the assay plate id
r	Integer, the number of wells from the center well (in one direction) to make the square

See Also

MC6_Methods, Method functions, mc6

is.odd *Check for odd numbers*

Description

is.odd takes an integer vector, x, and returns TRUE for odd integers.

Usage

```
is.odd(x)
```

Arguments

x	An integer

Value

TRUE for odd integers and FALSE for even integers.

See Also

Other tcpl abbreviations: lu, lw, sink.reset

`Load assay information`

Functions for loading assay information

Description

These functions query the tcpl databases and returns a data.table with assay ID and name information. More information about the assay hierarchy is available in the overview vignette.

Usage

```
tcplLoadAcid(fld = NULL, val = NULL, add.fld = NULL)

tcplLoadAeid(fld = NULL, val = NULL, add.fld = NULL)

tcplLoadAid(fld = NULL, val = NULL, add.fld = NULL)

tcplLoadAsid(fld = NULL, val = NULL, add.fld = NULL)
```

Arguments

fld	Character, the field(s) to query/subset on
val	List, vectors of values for each field to query/subset on. Must be in the same order as 'fld'.
add.fld	Character, additional field(s) to include, but not query/ subset on

Details

Each element in the assay hierarchy has its own function, loading the ID and name for the given assay element. For example, tcplLoadAsid will return the assay source ID (asid) and assay source name (asnm).

Value

A data.table containing the ID, name, and any additional fields.

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

## The load assay functions can be used without any parameters to list the
```

```
## full list of registered assay elements:
tcplLoadAsid()
tcplLoadAeid()

## Similarly, the user can add fields without doing any element selection:
tcplLoadAeid(add.fld = c("asid", "aid", "acid"))

## Or, the user can look only at a subset:
tcplLoadAeid(fld = "aeid", val = 1, add.fld = "asid")

## The field can be any value in one of the corresponding assay element
## tables, but the functions also recognize the abbreviated version of
## the name fields.
tcplListFlds("assay")
a1 <- tcplLoadAeid(fld = "anm", val = "Steroidogenesis")
a2 <- tcplLoadAeid(fld = "assay_name", val = "Steroidogenesis")
identical(a1, a2)

## Reset configuration
options(conf_store)
```

`lu` *Abbreviation for* `length(unique(x))`

Description

`lu` takes a logical vector, `x`, and returns `length(unique(x))`.

Usage

```
lu(x)
```

Arguments

`x`	A logical

Value

The unique of the `TRUE` values in `x`

See Also

`unique`, `which`

Other tcpl abbreviations: `is.odd`, `lw`, `sink.reset`

lw *Abbreviation for* `length(which(x))`

Description

`lw` takes a logical vector, `x`, and returns `length(which(x))`.

Usage

```
lw(x)
```

Arguments

`x`	A logical

Value

The length of the `TRUE` values in `x`

See Also

`length`, `which`

Other tcpl abbreviations: `is.odd`, `lu`, `sink.reset`

mc1 *Perform level 1 multiple-concentration processing*

Description

`mc1` loads level 0 data from the tcpl database for the given id and performs level 1 multiple-concentration processing. The processed data is then loaded into the mc1 table and all subsequent data is deleted with `tcplCascade`. See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the `tcplRun` wrapper function.

Usage

```
mc1(ac, wr = FALSE)
```

Arguments

`ac`	Integer of length 1, assay component id (acid) for processing.
`wr`	Logical, whether the processed data should be written to the tcpl database

Details

Level 1 processing includes defining the concentration and replicate index, cndx and repi, respectively.

Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean indiciating the success of processing and the second element is a data.table containing the processed data

See Also

Other multiple-concentration data processing functions: `mc2`, `mc3`, `mc4`, `mc5`, `mc6`

`mc2` *Perform level 2 multiple-concentration processing*

Description

`mc2` loads level 1 data from the tcpl database for the given id and performs level 2 multiple-concentration processing. The processed data is then loaded into the mc2 table and all subsequent data is deleted with `tcplCascade`. See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the `tcplRun` wrapper function.

Usage

```
mc2(ac, wr = FALSE)
```

Arguments

`ac`	Integer of length 1, assay component id (acid) for processing.
`wr`	Logical, whether the processed data should be written to the tcpl database

Details

Level 2 multiple-concentration processing includes defining the corrected value, cval, based on the correction methods listed in the mc2_acid and mc2_methods tables.

Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean indiciating the success of processing and the second element is a data.table containing the processed data

See Also

Method functions, MC2_Methods

Other multiple-concentration data processing functions: mc1, mc3, mc4, mc5, mc6

MC2_Methods *List of level 2 multiple-concentration correction functions*

Description

mc2_mthds returns a list of correction/transformation functions to be used during level 2 multiple-concentration processing.

Usage

```
mc2_mthds()
```

Details

The functions contained in the list returned by mc2_mthds return a list of expressions to be executed in the mc2 (not exported) function environment. The functions are described here for reference purposes, The mc2_mthds function is not exported, nor is it intended for use.

All available methods are described in the Available Methods section, listed by the function/method name.

Value

A list functions

Available Methods

More information about the level 2 multiple-concentration processing is available in the package vignette, "Pipeline_Overview."

log2 Take the logarithm of cval with the base 2.

log10 Take the logarithm of cval with the base 10.

rmneg Remove entries where cval is less than 0.

rmzero Remove entries where cval is 0.

mult25 Multiply cval by 25.

mult100 Multiply cval by 100.

negshift Shift cval by subtracting out the minimum of cval and adding 1, such that the new minimum of cval is 1.

mult25 Multiply cval by 2.5.

mult3 Multiply cval by 3.

mult6 Multiply cval by 6.

Note

This function is not exported and is not intended to be used by the user.

See Also

mc2, Method functions to query what methods get applied to each acid

---

mc3 *Perform level 3 multiple-concentration processing*

Description

mc3 loads level 2 data from the tcpl database for the given id and performs level 3 multiple-concentration processing. The processed data is then loaded into the mc3 table and all subsequent data is deleted with tcplCascade. See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the tcplRun wrapper function.

Usage

```
mc3(ac, wr = FALSE)
```

Arguments

ac	Integer of length 1, assay component id (acid) for processing.
wr	Logical, whether the processed data should be written to the tcpl database

Details

Level 3 multiple-concentration processing includes mapping assay component to assay endpoint, duplicating the data when the assay component has multiple assay endpoints, and any normalization of the data. Data normalization based on methods listed in mc3_aeid and mc3_methods tables.

Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean indiciating the success of processing and the second element is a data.table containing the processed data

See Also

Method functions, MC3_Methods

Other multiple-concentration data processing functions: mc1, mc2, mc4, mc5, mc6

MC3_Methods *List of level 3 multiple-concentration normalization methods*

Description

mc3_mthds returns a list of normalization methods to be used during level 3 multiple-concentration processing.

Usage

```
mc3_mthds()
```

Details

The functions contained in the list returned by mc3_mthds take 'aeids' (a numeric vector of aeid values) and returns a list of expressions to be executed in the mc3 (not exported) function environment. The functions are described here for reference purposes, The mc3_mthds function is not exported, nor is it intended for use.

All available methods are described in the Available Methods section, listed by the type of function and the function/method name.

Value

A list of functions

Available Methods

The methods are broken into three types, based on what fields they define. Different methods are used to define "bval" (the baseline value), "pval" (the positive control value), and "resp" (the final response value).

Although it does not say so specifically in each description, all methods are applied by aeid.

More information about the level 3 multiple-concentration processing is available in the package vignette, "Pipeline_Overview."

bval Methods:

bval.apid.nwlls.med Calculate bval as the median of cval for wells with wllt equal to "n," by apid.

bval.apid.lowconc.med Calculate bval as the median of cval for wells with wllt equal to "t" and cndx equal to 1 or 2, by apid.

bval.apid.twlls.med Calculate bval as the median of cval for wells with wllt equal to "t," by apid.

bval.apid.tn.med Calculate bval as the median of cval for wells with wllt equal to "t" or "n," by apid.

bval.apid.nwllslowconc.med Calculate bval as the median of cval for wells with wllt equal to "n" or wells with wllt equal to "t" and cndx equal to 1 or 2, by apid.

bval.spid.lowconc.med Calculate bval as the median of cval for wells with wllt equal to "t" and cndx equal to 1, 2, or 3, by spid.

bval.apid.nwllstcwllslowconc.med Calculate bval as the median of cval for wells with wllt equal to "n" or cndx equal to 1 or 2 and wllt equal to "t" or "c" by apid.

pval Methods:

pval.apid.pwlls.med Calculate pval as the median of cval for wells with wllt equal to "p," by apid.

pval.apid.mwlls.med Calculate pval as the median of cval for wells with wllt equal to "m," by apid.

pval.apid.medpcbyconc.max First calculate the median of cval for wells with wllt equal to "p" or "c," by wllt, conc, and apid. Then calculate pval as the maximum of the calculated medians, by apid.

pval.apid.medpcbyconc.min First calculate the median of cval for wells with wllt equal to "p" or "c," by wllt, conc, and apid. Then calculate pval as the minimum of the calculated medians, by apid.

pval.apid.mednchyconc.min First calculate the median of cval for wells with wllt equal to "m" or "o," by wllt, conc, and apid. Then calculate pval as the minimum of the calculated medians, by apid.

pval.apid.pmv.min First calculate the median of cval for wells with wllt equal to "p," "m," or "v," by wllt, conc, and apid. Then calculate pval as the minimum of the calculated medians, by apid.

pval.apid.pmv.max First calculate the median of cval for wells with wllt equal to "p," "m," or "v," by wllt, conc, and apid. Then calculate pval as the maximum of the calculated medians, by apid.

pval.apid.f.max First calculate the median of cval for wells with wllt equal to "f," by wllt, conc, and apid. Then calculate pval as the maximum of the calculated medians, by apid.

pval.apid.f.min First calculate the median of cval for wells with wllt equal to "f," by wllt, conc, and apid. Then calculate pval as the minimum of the calculated medians, by apid.

pval.apid.p.max First calculate the median of cval for wells with wllt equal to "p," by wllt, conc, and apid. Then calculate pval as the maximum of the calculated medians, by apid.

pval.apid.p.min First calculate the median of cval for wells with wllt equal to "p," by wllt, conc, and apid. Then calculate pval as the minimum of the calculated medians, by apid.

pval.apid.v.min First calculate the median of cval for wells with wllt equal to "v," by wllt, conc, and apid. Then calculate pval as the minimum of the calculated medians, by apid.

pval.zero Define pval as 0.

resp Methods:

resp.pc Calculate resp as $\frac{cval-bval}{pval-bval}100$.

resp.fc Calculate resp as $cval/bval$.

resp.logfc Calculate resp as $cval - bval$.

resp.log2 Take the logarithm of resp with base 2.

resp.mult25 Multiply resp by 25.

resp.scale.mad.log2fc Multiply resp by the scale factor $\frac{\log_2(1.2)}{3bmad}$.

resp.scale.quant.log2fc Determine the maximum response md where md = abs(1st centile - 50th centile) or abs(99th centile - 50th centile), whichever is greater. Scale the response such that 20 percent of md equals $\log_2(1.2)$.

resp.multneg1 Multiply resp by -1.

resp.shiftneg.3bmad Shift all resp values less than -3*bmad to 0.

resp.shiftneg.6bmad Shift all resp values less than -6*bmad to 0.

resp.shiftneg.10bmad Shift all resp values less than -10*bmad to 0.

resp.blineshift.3bmad.repi Shift resp values with the blineShift function by repi, where the window (wndw) is 3*bmad.

resp.blineshift.50.repi Shift resp values with the blineShift function by repi, where the window (wndw) is 50.

resp.blineshift.3bmad.spid Shift resp values with the blineShift function by spid, where the window (wndw) is 3*bmad.

resp.blineshift.50.spid Shift resp values with the blineShift function by spid, where the window (wndw) is 50.

none Do no normalization; make resp equal to cval.

Note

This function is not exported and is not intended to be used by the user.

See Also

`mc3`, `Method functions` to query what methods get applied to each aeid

`mc4` *Perform level 4 multiple-concentration processing*

Description

`mc4` loads level 3 data from the tcpl database for the given id and performs level 4 multiple-concentration processing. The processed data is then loaded into the mc4 table and all subsequent data is deleted with `tcplCascade`. See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the `tcplRun` wrapper function.

Usage

```
mc4(ae, wr = FALSE)
```

Arguments

`ae`	Integer of length 1, assay endpoint id (aeid) for processing.
`wr`	Logical, whether the processed data should be written to the tcpl database

Details

Level 4 multiple-concentration modeling takes the dose-response data for chemical-assay pairs, and fits three models to the data: constant, hill, and gain-loss. For more information about the models see `Models`. When a chemical has more than one sample, the function fits each sample seperately.

Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean indiciating the success of processing and the second element is a data.table containing the processed data

See Also

tcplFit, Models

Other multiple-concentration data processing functions: mc1, mc2, mc3, mc5, mc6

---

mc5 *Perform level 5 multiple-concentration processing*

Description

mc5 loads level 4 data from the tcpl database for the given id and performs level 5 multiple-concentration processing. The processed data is then loaded into the mc5 table and all subsequent data is deleted with tcplCascade. See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the tcplRun wrapper function.

Usage

```
mc5(ae, wr = FALSE)
```

Arguments

ae	Integer of length 1, assay endpoint id (aeid) for processing.
wr	Logical, whether the processed data should be written to the tcpl database

Details

Level 5 multiple-concentration hit-calling uses the fit parameters and the activity cutoff methods from mc5_aeid and mc5_methods to make an activity call and identify the winning model for each fit.

Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean indiciating the success of processing and the second element is a data.table containing the processed data

See Also

Method functions, MC5_Methods

Other multiple-concentration data processing functions: mc1, mc2, mc3, mc4, mc6

MC5_Methods *Load list of level 5 multiple-concentration cutoff methods*

Description

mc5_mthds returns a list of additional activity cutoff methods to be used during level 5 multiple-concentration processing.

Usage

```
mc5_mthds()
```

Value

A list of functions

Available Methods

More information about the level 5 multiple-concentration processing is available in the package vignette, "Pipeline_Overview."

bmad3 Add a cutoff value of 3*bmad.

pc20 Add a cutoff value of 20.

log2_1.2 Add a cutoff value of log2(1.2).

log10_1.2 Add a cutoff value of log10(1.2).

bmad5 Add a cutoff value of 5*bmad.

bmad6 Add a cutoff value of 6*bmad.

bmad10 Add a cutoff value of 10*bmad.

log2_2 Add a cutoff value of log2(2).

log10_2 Add a cutoff value of log10(2).

neglog2_0.88 Add a cutoff value of -1*log2(0.88).

See Also

mc5, Method functions to query what methods get applied to each aeid

mc6 *Perform level 6 multiple-concentration processing*

Description

mc6 loads level 5 data from the tcpl database for the given id and performs level 6 multiple-concentration processing. The processed data is then loaded into the mc6 table and all subsequent data is deleted with tcplCascade. See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the tcplRun wrapper function.

Usage

```
mc6(ae, wr = FALSE)
```

Arguments

ae	Integer of length 1, assay endpoint id (aeid) for processing.
wr	Logical, whether the processed data should be written to the tcpl database

Details

Level 6 multiple-concentration flagging uses both the plate level concentration-response data and the modeled parameters to flag potential false positives and false negative results.

Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean indiciating the success of processing and the second element is a data.table containing the processed data

See Also

Method functions, MC6_Methods

Other multiple-concentration data processing functions: mc1, mc2, mc3, mc4, mc5

`MC6_Methods` *Load list of level 6 multiple-concentration flag methods*

Description

`mc6_mthds` returns a list of flag methods to be used during level 6 multiple-concentration processing.

Usage

```
mc6_mthds()
```

Value

A list functions

Available Methods

More information about the level 6 multiple-concentration processing is available in the package vignette, "Pipeline_Overview."

singlept.hit.high The singlept.hit.high flag identifies concentration series where the median response was greater than 3*bmad only at the highest tested concentration and the series had an active hit-call.

singlept.hit.mid The singlept.hit.mid flag identifies concentration series where the median response was greater than 3*bmad at only one concentration (not the highest tested concentration) and the series had an active hit-call.

multipoint.neg The multipoint.neg flag identifies concentration series with response medians greater than 3*bmad at multiple concentrations and an inactive hit-call.

gnls.lowconc The gnls.lowconc flag identifies concentration series where the gain-loss model won, the gain AC50 is less than the minimum tested concentration, and the loss AC50 is less than the mean tested concentration.

noise The noise flag attempts to identify noisy concentration series by flagging series where the root mean square error for the series is greater than the cutoff for the assay endpoint.

border.hit The border.hit flag identifies active concentration series where the top parameter of the winning model was less than or equal to 1.2*cut-off or the the activity probablity was less than 0.9.

border.miss The border.miss flag identifies inactive concentration series where either the Hill or gain-loss top parameter was greater than or equal to 0.8*cut-off and the activity probability was greater than 0.5.

overfit.hit The overfit.hit flag recalculates the model winner after applying a small sample correction factor to the AIC values. If the hit-call would be changed after applying the small sample correction factor the series is flagged. Series with less than 5 concentrations where the hill model won and series with less than 7 concentrations where the gain-loss model won are automatically flagged.

efficacy.50 The efficacy.50 flag identifies concentration series with efficacy values (either the modeled top parameter for the winning model or the maximum median response) are less than 50 for percent activity data or log2(1.5) for fold induction data

modlga.lowconc The modlga.lowconc flag identifies concentration series with modl_ga (AC50) values less than the minimum tested concentration.

See Also

`mc6`, `Method functions` to query what methods get applied to each aeid

`Method functions` *Functions for managing processing methods*

Description

These functions are used to manage which methods are used to process data. They include methods for assigning, clearing, and loading the assigned methods. Also, `tcplMthdList` lists the available methods.

Usage

```
tcplMthdAssign(lvl, id, mthd_id, ordr = NULL, type)

tcplMthdClear(lvl, id, mthd_id = NULL, type)

tcplMthdList(lvl, type = "mc")

tcplMthdLoad(lvl, id = NULL, type = "mc")
```

Arguments

`lvl`	Integer of length 1, the method level
`id`	Integer, the assay component or assay endpoint id(s)
`mthd_id`	Integer, the method id(s)
`ordr`	Integer, the order in which to execute the analysis methods, must be the same length as mthd_id, does not apply to levels 5 or 6
`type`	Character of length 1, the data type, "sc" or "mc"

Details

`tcplMthdLoad` loads the assigned methods for the given level and ID(s). Similarly, `tcplMthdList` displays the available methods for the given level. These two functions do not make any changes to the database.

Unlike the `-Load` and `-List` functions, the `-Assign` and `-Clear` functions alter the database and trigger a delete cascade. `tcplMthdAssign` assigns methods to the given ID(s), and `tcplMthdClear` removes methods. In addition to the method ID ('mthd_id'), assigning methods at some levels

require an order ('ordr'). The 'ordr' parameter is necessary to allow progression of methods at level one for single-concentration processing, and levels two and three for multiple-concentration processing. More information about method assignments and the delete cascade are available in the package vignette.

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

## tcplListMthd allows the user to display the available methods for
## a given level and data type
head(tcplMthdList(lvl = 2, type = "mc"))

## tcplLoadMthd shows which methods are assigned for the given ID, level,
## and data type. Here we will show how to register, load, and clear methods
## using an acid not in the example database. Note: There is no check for
## whether an ID exists before assigning/clearing methods.
tcplMthdLoad(lvl = 2, id = 55, type = "mc")

## Not run:
## ACID 55 does not have any methods. Assign methods from the list above.
tcplMthdAssign(lvl = 2,
               id = 55,
               mthd_id = c(3, 4, 2),
               ordr = 1:3,
               type = "mc")
## Method assignment can be done for multiple assays, too.
tcplMthdAssign(lvl = 2,
               id = 53:54,
               mthd_id = c(3, 4, 2),
               ordr = 1:3,
               type = "mc")

## Cleanup example method assigments
tcplMthdClear(lvl = 2, id = 53:55, type = "mc")

## End(Not run)
## Reset configuration
options(conf_store)
```

`Models` *Model objective functions*

Description

These functions take in the dose-response data and the model parameters, and return a likelyhood value. They are intended to be optimized using `constrOptim` in the `tcplFit` function.

Usage

```
tcplObjCnst(p, resp)

tcplObjGnls(p, lconc, resp)

tcplObjHill(p, lconc, resp)
```

Arguments

p	Numeric, the parameter values. See details for more information.
resp	Numeric, the response values
lconc	Numeric, the log10 concentration values

Details

These functions produce an estimated value based on the model and given parameters for each observation. Those estimated values are then used with the observed values and a scale term to calculate the log-likelyhood.

Let $t(z, \nu)$ be the Student's t-ditribution with ν degrees of freedom, y_i be the observed response at the i^{th} observation, and μ_i be the estimated response at the i^{th} observation. We calculate z_i as:

$$z_i = \frac{y_i - \mu_i}{e^{\sigma}}$$

where σ is the scale term. Then the log-likelyhood is:

$$\sum_{i=1}^{n}[ln(t(z_i, 4)) - \sigma]$$

Where n is the number of observations.

Value

The log-likelyhood.

Constant Model (cnst)

`tcplObjCnst` calculates the likelyhood for a constant model at 0. The only parameter passed to `tcplObjCnst` by `p` is the scale term σ. The constant model value μ_i for the i^{th} observation is given by:

$$\mu_i = 0$$

Gain-Loss Model (gnls)

`tcplObjGnls` calculates the likelyhood for a 5 parameter model as the product of two Hill models with the same top and both bottoms equal to 0. The parameters passed to `tcplObjGnls` by `p` are (in order) top (tp), gain log AC50 (ga), gain hill coefficient (gw), loss log AC50 la, loss hill coefficient lw, and the scale term (σ). The gain-loss model value μ_i for the i^{th} observation is given by:

$$g_i = \frac{1}{1 + 10^{(ga - x_i)gw}}$$

$$l_i = \frac{1}{1 + 10^{(x_i - la)lw}}$$

$$\mu_i = tp(g_i)(l_i)$$

where x_i is the log concentration for the i^{th} observation.

Hill Model (hill)

`tcplObjHill` calculates the likelyhood for a 3 parameter Hill model with the bottom equal to 0. The parameters passed to `tcplObjHill` by `p` are (in order) top (tp), log AC50 (ga), hill coefficient (gw), and the scale term (σ). The hill model value μ_i for the i^{th} observation is given by:

$$\mu_i = \frac{tp}{1 + 10^{(ga - x_i)gw}}$$

where x_i is the log concentration for the i^{th} observation.

`Query functions` *Wrappers for sending queries and fetching results*

Description

These functions send a query to the given database, and are the access point for all tcpl functions that query or update the tcpl database.

Usage

```
tcplQuery(query, db = getOption("TCPL_DB"), drvr = getOption("TCPL_DRVR"))

tcplSendQuery(query, db = getOption("TCPL_DB"),
  drvr = getOption("TCPL_DRVR"))
```

Arguments

`query`	Character of length 1, the query string
`db`	Character of length 1, the name of the tcpl database
`drvr`	Character of length 1, which database driver to use

Details

Currently, the tcpl package only supports the "MySQL" and "SQLite" database drivers.

`tcplQuery` returns a data.table object with the query results. `tcplSendQuery` sends a query, but does not fetch any results, and returns 'TRUE' or the error message given by the database.

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

tcplQuery("SELECT 'Hello World';")
tcplQuery("SELECT * FROM assay;")

## Reset configuration
options(conf_store)
```

Register/update annotation

Functions for registering & updating annotation information

Description

These functions are used to register and update the chemical and assay annotation information.

Usage

```
tcplRegister(what, flds)

tcplUpdate(what, id, flds)
```

Arguments

what	Character of length 1, the name of the ID to register or update
flds	Named list, the other fields and their values
id	Integer, the ID value(s) to update

Details

These functions are used to populate the tcpl database with the necessary annotation information to complete the processing. As shown in the package vignette, the package requires some information about the samples and assays before data can be loaded into the tcpl database.

Depending on what is being registered, different information is required. The following table lists the fields that can be registered/updated by these functions, and the minimal fields required for registering a new ID. (The database table affected is in parentheses.)

- asid (assay_source): assay_source_name
- aid (assay): asid, assay_name, assay_footprint
- acid (assay_component): aid, assay_component_name

- aeid (assay_component_endpoint): acid, assay_component_endpoint_name, normalized_data_type
- acsn (assay_component_map): acid, acsn
- spid (sample): spid, chid
- chid (chemical): chid, casn
- clib (chemical_library): chid, clib

Note: The functions accept the abbreviated forms of the names, ie. "aenm" rather than the full "assay_component_endpoint_name." More information about the registration process and all of the fields is available in the vignette.

Examples

```
## Not run:
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

## Load current ASID information
tcplLoadAsid()

## Register a new assay source
tcplRegister(what = "asid", flds = list(asnm = "example_asid"))

## Show the newly registered ASID
tcplLoadAsid(add.fld = "assay_source_desc")

## Notice that the newly created ASID does not have an assay_source_desc.
## The field could have been defined during the registration process, but
## can also be updated using tcplUpdate
i1 <- tcplLoadAsid()[asnm == "example_asid", asid]
tcplUpdate(what = "asid",
           id = i1,
           flds = list(assay_source_desc = "example asid description"))
tcplLoadAsid(add.fld = "assay_source_desc")

## Remove the created ASID. Note: Manually deleting primary keys can cause
## serious database problems and should not generally be done.
tcplSendQuery(paste0("DELETE FROM assay_source WHERE asid = ", i1, ";"))

## Reset configuration
options(conf_store)

## End(Not run)
```

registerMthd *Add a new analysis method*

Description

registerMthd registers a new analysis method to the tcpl databases.

Usage

```
registerMthd(lvl, mthd, desc, nddr = 0L, type)
```

Arguments

lvl	Integer of length 1, the level for the analysis method
mthd	Character, the name of the method
desc	Character, same length as mthd, the method description
nddr	Integer, 0 or 1, 1 if the method requires loading the dose- response data
type	Character of length 1, the data type, "sc" or "mc"

Details

'mthd' must match a corresponding function name in the functions that load the methods, ie. mc2_mthds. 'nddr' only applies to level 6 methods.

sc1 *Perform level 1 single-concentration processing*

Description

sc1 loads level 0 data from the tcpl database for the given id and performs level 1 single-concentration processing. The processed data is then loaded into the sc1 table and all subsequent data is deleted with tcplCascade. See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the tcplRun wrapper function.

Usage

```
sc1(ac, wr = FALSE)
```

Arguments

ac	Integer of length 1, assay component id (acid) for processing.
wr	Logical, whether the processed data should be written to the tcpl database

Details

Level 1 single-concentration processing includes mapping assay component to assay endpoint, duplicating the data when the assay component has multiple assay endpoints, and any normalization of the data. Data normalization based on methods listed in sc1_aeid and sc1_methods tables.

Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean indiciating the success of processing and the second element is a data.table containing the processed data

See Also

`Method functions`, `SC1_Methods`

Other single-concentration data processing functions: `sc2`

`SC1_Methods` *List of level 1 single-concentration normalization functions*

Description

`sc1_mthds` returns a list of functions to be used during level 1 single-concentration processing.

Usage

```
sc1_mthds()
```

Details

The functions contained in the list returned by `sc1_mthds` return a list of expressions to be executed in the `sc2` (not exported) function environment. The functions are described here for reference purposes, The `sc1_mthds` function is not exported, nor is it intended for use.

All available methods are described in the Available Methods section, listed by the function/method name.

Value

A list functions

Available Methods

The methods are broken into three types, based on what fields they define. Different methods are used to define "bval" (the baseline value), "pval" (the positive control value), and "resp" (the final response value).

Although it does not say so specifically in each description, all methods are applied by acid.

More information about the level 3 single-concentration processing is available in the package vignette, "Pipeline_Overview."

bval Methods:

bval.apid.nwlls.med Calculate bval as the median of rval for wells with wllt equal to "n," by apid.

bval.apid.twlls.med Calculate bval as the median of rval for wells with wllt equal to "t," by apid.

bval.apid.tn.med Calculate bval as the median of rval for wells with wllt equal to "t" or "n," by apid.

pval Methods:

pval.apid.pwlls.med Calculate pval as the median of rval for wells with wllt equal to "p," by apid.

pval.apid.mwlls.med Calculate pval as the median of rval for wells with wllt equal to "m," by apid.

pval.apid.medpcbyconc.max First calculate the median of rval for wells with wllt equal to "p" or "c," by wllt, conc, and apid. Then calculate pval as the maximum of the calculated medians, by apid.

pval.apid.medpcbyconc.min First calculate the median of rval for wells with wllt equal to "p" or "c," by wllt, conc, and apid. Then calculate pval as the minimum of the calculated medians, by apid.

pval.apid.medncbyconc.min First calculate the median of rval for wells with wllt equal to "m" or "o," by wllt, conc, and apid. Then calculate pval as the minimum of the calculated medians, by apid.

pval.zero Define pval as 0.

resp Methods:

resp.pc Calculate resp as $\frac{rval-bval}{pval-bval}100$.

resp.fc Calculate resp as $rval/bval$.

resp.logfc Calculate resp as $rval - bval$.

resp.log2 Take the logarithm of resp with base 2.

resp.multneg1 Multiply resp by -1.

none Do no normalization; make resp equal to rval.

Note

This function is not exported and is not intended to be used by the user.

See Also

sc1, Method functions to query what methods get applied to each acid

sc2 *Perform level 2 single-concentration processing*

Description

sc2 loads level 1 data from the tcpl database for the given id and performs level 2 single-concentration processing. The processed data is then loaded into the sc2 table and all subsequent data is deleted with tcplCascade. See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the tcplRun wrapper function.

Usage

```
sc2(ae, wr = FALSE)
```

Arguments

ae	Integer of length 1, assay endpoint id (aeid) for processing.
wr	Logical, whether the processed data should be written to the tcpl database

Details

Level 2 single-concentration processing defines the bmad value, and uses the activity cutoff methods from sc2_aeid and sc2_methods to make an activity call.

Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean indiciating the success of processing and the second element is a data.table containing the processed data

See Also

Method functions, SC2_Methods

Other single-concentration data processing functions: sc1

SC2_Methods *List of level 2 single-concentration hit-call functions*

Description

sc2_mthds returns a list of functions to be used during level 2 single-concentration processing.

Usage

```
sc2_mthds()
```

Details

The functions contained in the list returned by sc2_mthds return a list of expressions to be executed in the sc2 (not exported) function environment. The functions are described here for reference purposes, The sc2_mthds function is not exported, nor is it intended for use.

All available methods are described in the Available Methods section, listed by the function/method name.

Value

A list functions

Available Methods

More information about the level 2 single-concentration processing is available in the package vignette, "Pipeline_Overview."

bmad3 Add a cutoff value of 3*bmad.

pc20 Add a cutoff value of 20.

log2_1.2 Add a cutoff value of log2(1.2).

log10_1.2 Add a cutoff value of log10(1.2).

bmad5 Add a cutoff value of 5*bmad.

bmad6 Add a cutoff value of 6*bmad.

bmad10 Add a cutoff value of 10*bmad.

pc30orbmad3 Add a cutoff value of either 30 or 3*bmad, whichever is less.

Note

This function is not exported and is not intended to be used by the user.

See Also

sc2, Method functions to query what methods get applied to each acid

sink.reset *Reset all sinks*

Description

`sink.reset` resets all sinks and returns all output to the console.

Usage

```
sink.reset()
```

Details

`sink.reset` identifies all sinks with `sink.number` then returns all output and messages back to the console.

See Also

`sink`, `sink.number`

Other tcpl abbreviations: `is.odd`, `lu`, `lw`

tcplAddModel *Draw a tcpl Model onto an exisiting plot*

Description

`tcplAddModel` draws a a line for one of the tcpl Models (see `Models` for more information) onto an existing plot.

Usage

```
tcplAddModel(pars, modl = NULL, adj = NULL, ...)
```

Arguments

`pars`	List of parameters from level 4 or 5 output
`modl`	Character of length 1, the model to plot: 'cnst,' 'hill,' or 'gnls'
`adj`	Numeric of length 1, an adjustment factor, see details for more information
`...`	Additional arguments passed to `curve`

Details

tcplAddModel draws the model line assuming the x-axis represents log base 10 concentration.

If modl is NULL, the function checks pars$modl and will return an error if pars$modl is also NULL.

adj is intended to scale the models, so that models with different response units can be visualized on a single plot. The recommended value for adl is 1/(3*bmad) for level 4 data and 1/coff for level 5 data. If adj is NULL the function will check pars$adj and set adj to 1 if pars$adj is also NULL.

See Also

Models, tcplPlotFits

Examples

```
## Create some dummy data to plot
logc <- 1:10
r1 <- sapply(logc, tcplHillVal, ga = 5, tp = 50, gw = 0.5)
r2 <- log2(sapply(logc, tcplHillVal, ga = 4, tp = 30, gw = 0.5))
p1 <- tcplFit(logc = logc, resp = r1, bmad = 10)
p2 <- tcplFit(logc = logc, resp = r2, bmad = log2(1.5))

## In the dummy data above, the two plots are on very different scales
plot(r1 ~ logc, pch = 16, ylab = "raw response")
tcplAddModel(pars = p1, modl = "hill")
points(r2 ~ logc)
tcplAddModel(pars = p2, modl = "hill", lty = "dashed")

## To visualize the two curves on the same plot for comparison, we can
## scale the values to the bmad, such that a scaled response of 1 will equal
## the bmad for each curve.
plot(r1/10 ~ logc, pch = 16, ylab = "scaled response")
tcplAddModel(pars = p1, modl = "hill", adj = 1/10)
points(r2/log2(5) ~ logc)
tcplAddModel(pars = p2, modl = "hill", adj = 1/log2(5), lty = "dashed")
```

---

tcplAICProb *Calculate the AIC probabilities*

Description

tcplAICProb Calculates the probability that the model best represents the data based on the AIC value for each model.

Usage

```
tcplAICProb(...)
```

Arguments

...	Numeric vectors of AIC values

Details

The function takes vectors of AIC values. Each vector represents the model AIC values for multiple observation sets. Each vector must contain the same number and order of observation sets. The calculation assumes every possible model is accounted for, and the results should be interpreted accordingly.

Value

A vector of probability values for each model given, as a list.

See Also

tcplFit, AIC for more information about AIC values.

Examples

```
## Returns the probability for each model, given models with AIC values
## ranging from 80 to 100
tcplAICProb(80, 85, 90, 95, 100)

## Also works for vectors
m1 <- c(95, 195, 300) ## model 1 for three different observations
m2 <- c(100, 200, 295) ## model 2 for three different observations
tcplAICProb(m1, m2)
```

---

tcplAppend *Append rows to a table*

Description

tcplAppend takes a data.table (dat) and appends the data.table into a database table.

Usage

```
tcplAppend(dat, tbl, db)
```

Arguments

dat	data.table, the data to append to a table
tbl	Character of length 1, the table to append to
db	Character of length 1, the database containing tbl

Note

This function is not exported and not intended to be used by the user.

`tcplCascade` *Do a cascading delete on tcpl screening data*

Description

`tcplCascade` deletes the data for the given id(s) starting at the processing level given. The delete will cascade through all subsequent tables.

Usage

```
tcplCascade(lvl, type, id)
```

Arguments

`lvl`	Integer of length 1, the first level to delete from
`type`	Character of length 1, the data type, "sc" or "mc"
`id`	Integer, the id(s) to delete. See details for more information.

Details

The data type can be either 'mc' for mutliple concentration data, or 'sc' for single concentration data. Multiple concentration data will be loaded into the level tables, whereas the single concentration will be loaded into the single tables.

If lvl is less than 3, id is interpreted as acid(s) and if lvl is greater than or equal to 3, id is interpreted as aeid(s).

Note

This function is not exported and not intended to be used by the user.

`tcplCode2CASN` *Convert chemical code to CAS Registry Number*

Description

`tcplCode2CASN` takes a code and converts it CAS Registry Number.

Usage

```
tcplCode2CASN(code)
```

Arguments

code	Character of length 1, a chemical code

Details

The function checks for the validity of the CAS Registry Number. Also, the ToxCast data includes chemcials for which there is no CASRN. The convention for these chemicals is to give them a CASRN as NOCAS_chid; the code for these compounds is CNOCASchid. The function handles the NOCAS compounds as they are stored in the database, as shown in the exmaple below.

Value

A CAS Registry Number.

Examples

```
tcplCode2CASN("C80057")
tcplCode2CASN("C09812420") ## Invalid CASRN will give a warning
tcplCode2CASN("CNOCAS0015") ## The underscore is reinserted for NOCAS codes
```

---

tcplCytoPt *Calculate the cytotoxicity point based on the "burst" endpoints*

Description

tcplCytoPt calculates the cytotoxicity point and average cytotoxicity distribution based on the acitivty in the "burst" assay endpoints.

Usage

```
tcplCytoPt(chid = NULL, aeid = NULL, flag = TRUE, min.test = TRUE,
  default.pt = 3)
```

Arguments

chid	Integer, chemical ID values to subset on
aeid	Integer, assay endpoint ID values to override the "burst assay" definitions
flag	Integer, mc6_mthd_id values to be passed to tcplSubsetChid
min.test	Integer or Boolean, the number of tested assay endpoints required for a cheimcal to be used in calculating the "global MAD."
default.pt	Numeric of length 1, the default cytotoxicity point value

Details

tcplCytoPt provides estimates for chemical-specific cytotoxicity distributions (more information available in the vignette.) Before calculating the cytotoxicity distributions, the level 5 data is subsetted by the tcplSubsetChid function.

The 'chid' parameter specifies a subset of chemicals to use in the calculations, given by chemical ID (chid). The 'aeid' parameter specifies which assays to use in calculating the cytotoxicity point and distribution. By default tcplCytoPt will use all available chemicals and the assay endpoints defined by the 'burst_assay' field in the "assay_component_endpoint" table. The examples show how to identify the "burst" endpoints.

tcplCytoPt returns the cytotoxicity point (the AC50 values of the active "burst" endpoints), the corresponding MAD, and the global MAD (median of the calculated MAD values). Not every chemical must be tested in every "burst" endpoint. The 'min.test' parameter allows the user to specify a minimum number of tested assay endpoints as a requirement for MAD values to be included in the global MAD calculation. For example, suppose the user supplies 10 "burst" assays. The user can choose to require a chemical to be tested in at least 5 of those assays for it's MAD value to be included in the global MAD calculation. Having chemicals with many less "burst" endpoints tested may inflate or deflate the global MAD calculation. By default (values of TRUE or NULL), tcplCytoPt requires a chemical to be tested in at least 80% of the given "burst" assays. The user can also provide 'min.test' values of FALSE (indicating to include all MAD values), or a number (indicating a specific number of endpoints).

Chemicals without at least 2 active "burst" assays do not have a MAD value, and the cytotoxicity point is defined by the 'default.pt' parameter. The default value for 'default.pt' is 3.

The resulting data.table has the following fields:

1. "chid" – The chemical ID.
2. "code" – The chemcial code.
3. "chnm" – The chemical name.
4. "casn" – The chemical CASRN.
5. "med" – The median of the "burst" endpoint log(AC50) ("modl_ga" in the level 5 output) values.
6. "mad" – The MAD of the "burst" endpoint log(AC50) values.
7. "ntst" – The number of "burst" endpoints tested.
8. "nhit" – The number of acive "burst" endpoints.
9. "use_global_mad" – TRUE/FALSE, whether the mad value was used in the global MAD calculation.
10. "global_mad" – The median of the "mad" values where "use_global_mad" is TRUE.
11. "cyto_pt" – The cytotoxicity point, or the value in "med" when "nhit" is at least 2.
12. "cyto_pt_um" – 10^{cyto_pt}
13. "lower_bnd_um" – $10^{cyto\_pt-3global\_mad}$

Value

A data.table with the cytotoxicity distribution for each chemical. The definition of the field names are listed under "details."

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

## Load the "burst" endpoints -- none are defined in the example dataset
tcplLoadAeid(fld = "burst_assay", val = 1)

## Calculate the cytotoxicity distributions using both example endpoints
tcplCytoPt(aeid = 1:2)

## The above example does not calculate a global MAD, because no chemical
## hit both endpoints. (This makes sense, because both endpoints are
## derived from one component, where one endpoint is acitivity in the
## up direction, and the other is acitivty in the down direction.)
## Note, the cyto_pt is also 3 for all chemicals, because the function
## requires at least two endpoints to calculate a cytotoxicity point. If
## the user wishes to use one assay, this function is not necessary.

## Changing 'default.pt' will change cyto_pt in the resulting data.table
tcplCytoPt(aeid = 1:2, default.pt = 6)

## Reset configuration
options(conf_store)
```

---

tcplDelete *Delete rows from tcpl databases*

Description

`tcplDelete` deletes rows from the given table and database.

Usage

```
tcplDelete(tbl, fld, val, db)
```

Arguments

`tbl`	Character, length 1, the table to delete from
`fld`	Character, the field(s) to query on
`val`	List, vectors of values for each field to query on. Must be in the same order as 'fld'.
`db`	Character, the database containing the table

Note

This function is not exported and not intended to be used by the user.

See Also

tcplSendQuery

---

tcplFit *Fit the data with the constant, hill, and gain-loss models*

Description

tcplFit fits the constant, hill, and gain-loss models to the given data and returns some summary statistics and the fit parameters in a list.

Usage

```
tcplFit(logc, resp, bmad, force.fit = FALSE, ...)
```

Arguments

logc	Numeric, log concentration values
resp	Numeric, normalized response values
bmad	Numeric, the baseline median absolute deviation for the entire assay
force.fit	Logical, TRUE indicates to attempt fitting every concentration series
...	Any other data to be included in list output.

Details

By default, tcplFit will only attempt to fit concentration series when at least one median value is greater than 3*bmad.

Value

List of summary values and fit parameters for the given data.

See Also

tcplObjCnst, tcplObjHill, tcplObjGnls, constrOptim

Examples

```
logc <- 1:10
resp <- sapply(1:10, tcplHillVal, ga = 5, tp = 50, gw = 0.5)
params <- tcplFit(logc = logc, resp = resp, bmad = 10)
plot(resp ~ logc)
tcplAddModel(pars = params, modl = "hill")
```

tcplListFlds *Load the field names for a table*

Description

tcplListFlds loads the column names for the given table and database.

Usage

```
tcplListFlds(tbl, db = getOption("TCPL_DB"))
```

Arguments

tbl	Character of length 1, the tcpl database table
db	Character of length 1, the tcpl database

Details

This function can be particularly useful in defining the 'fld' param in the tcplLoad- functions.

Value

A string of field names for the given table.

Examples

```
## Gives the fields in the mc1 table
tcplListFlds("mc1")
```

tcplLoadChem *Load sample/chemical information*

Description

tcplLoadChem queries the tcpl database and returns the chemcial information for the given field and values.

Usage

```
tcplLoadChem(field = NULL, val = NULL, exact = TRUE,
  include.spid = TRUE)
```

Arguments

field	Character of length 1, the field to query on
val	Vector of values to subset on
exact	Logical, should chemical names be considered exact?
include.spid	Logical, should spid be included?

Details

The 'field' parameter is named differently from the 'fld' parameter seen in other functions because it only takes one input.

The functionality of the 'exact' parameter cannot be demonstrated within the SQLite environment. However, in the MySQL environment the user should be able to give parital chemcial name strings, to find chemicals with similar names. For example, setting 'val' to "phenol" when 'field' is "chnm" and 'exact' is FALSE might pull up the chemicals "Bisphenol A" and "4-Butylphenol". More technically, setting 'exact' to FALSE passes the string in 'val' to an RLIKE statement within the MySQL query.

Value

A data.table with the chemical information for the given parameters

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

## Passing no parameters gives all of the registered chemicals with their
## sample IDs
tcplLoadChem()

## Or the user can exclude spid and get a unique list of chemicals
tcplLoadChem(include.spid = FALSE)

## Other examples:
tcplLoadChem(field = "chnm", val = "Bisphenol A")
tcplLoadChem(field = "chid", val = 1:5)

## Reset configuration
options(conf_store)
```

`tcplLoadClib` *Load chemical library information*

Description

`tcplLoadClib` queries the tcpl databases and returns information about the chemial library.

Usage

```
tcplLoadClib(field = NULL, val = NULL)
```

Arguments

`field`	Character of length 1, `'chid'` or `'clib'`, whether to search by chemical id (chid), or chemical library (clib)
`val`	The values to query on

Details

Chemicals are stored in different libraries by chemcial ID. Therefore, it is not possible to delineate samples with the same chemical ID into two distinct chemical libraries. However, it is possible for a chemcial ID to belong to more than one (or no) chemical libraries.

When chemicals belong to more than one library, the chemical is listed multiple times (one for each distinct library).

Value

A data.table with the chemical library information for the given parameters.

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

## Passing no parameters gives all of the chemical ISs that have a chemical
## library registered
clib <- tcplLoadClib()

## Notice there are more rows in tcplLoadClib than in tcplLoadChem,
## indicating some chemicals must belong to more than library.
chem <- tcplLoadChem(include.spid = FALSE)
nrow(chem)
nrow(clib)

## It is possible that some chemicals do not have a chemical library
## registered, although this is not the case in the example data.
```

```
all(chem$chid %in% clib$chid)

## Show the unique chemical libraries
clib[ , unique(clib)]

## Specifying a chemical library will not show what other libraries a
## chemical might belong to.
tcplLoadClib(field = "clib", val = "other")
tcplLoadClib(field = "chid", val = 1:2)

## Reset configuration
options(conf_store)
```

`tcplLoadData` *Load tcpl data*

Description

`tcplLoadData` queries the tcpl databases and returns a data.table with data for the given level and data type.

Usage

```
tcplLoadData(lvl, fld = NULL, val = NULL, type = "mc")
```

Arguments

`lvl`	Integer of length 1, the level of data to load
`fld`	Character, the field(s) to query on
`val`	List, vectors of values for each field to query on. Must be in the same order as 'fld'.
`type`	Character of length 1, the data type, "sc" or "mc"

Details

The data type can be either 'mc' for mutliple concentration data, or 'sc' for single concentration data. Multiple concentration data will be loaded into the 'mc' tables, whereas the single concentration will be loaded into the 'sc' tables.

Setting 'lvl' to "agg" will return an aggregate table containing the m4id with the concentration-response data and m3id to map back to well-level information.

Leaving `fld` NULL will return all data.

Valid `fld` inputs are based on the data level and type:

type	lvl	Queried tables
sc	0	sc0
sc	1	sc0, sc1

sc	agg	sc1, sc2_agg
sc	2	sc2
mc	0	mc0
mc	1	mc0, mc1
mc	2	mc0, mc1, mc2
mc	3	mc0, mc1, mc3
mc	agg	mc3, mc4_agg
mc	4	mc4
mc	5	mc4, mc5
mc	6	mc4, mc6

Value

A data.table containing data for the given fields.

See Also

tcplQuery, data.table

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

## Load all of level 0 for multiple-concentration data, note 'mc' is the
## default value for type
tcplLoadData(lvl = 0)

## Load all of level 1 for single-concentration
tcplLoadData(lvl = 1, type = "sc")

## List the fields available for level 1, coming from tables mc0 and mc1
tcplListFlds(tbl = "mc0")
tcplListFlds(tbl = "mc1")

## Load level 0 data where the well type is "t" and the concentration
## index is 3 or 4
tcplLoadData(lvl = 1, fld = c("wllt", "cndx"), val = list("t", c(3:4)))

## Reset configuration
options(conf_store)
```

---

tcplLoadUnit *Load response units for assay endpoints*

Description

tcplLoadUnit queries the tcpl databases and returns a data.table with the response units for the given assay endpoint ids (aeid).

Usage

```
tcplLoadUnit(aeid)
```

Arguments

aeid	Integer, assay endpoint ids

Value

A data.table containing level 3 correction methods for the given aeids.

See Also

tcplQuery, data.table

---

tcplMakeAeidPlts *Create a .pdf with dose-response plots*

Description

tcplMakeAeidPlts creates a .pdf file with the dose-response plots for the given aeid.

Usage

```
tcplMakeAeidPlts(aeid, lvl = 4L, fname = NULL, odir = getwd(),
  ordr.fitc = TRUE, clib = NULL)
```

Arguments

aeid	Integer of length 1, the assay endpoint id
lvl	Integer of lengh 1, the data level to use (4-6)
fname	Character, the filename
odir	The directory to save the .pdf file in
ordr.fitc	Logical, should the fits be ordered by fit category?
clib	Character, the chemical library to subset on, see tcplLoadClib for more information.

Details

tcplMakeAeidPlts provides a wrapper for tcplPlotFits, allowing the user to produce PDFs with the curve plots without having to separately load all of the data and establish the PDF device.

If 'fname' is NULL, a default name is given by concatenating together assay information.

Note, the default value for ordr.fitc is TRUE in tcplMakeAeidPlts, but FALSE in tcplPlotFits

Examples

```
## Not run:
## Will produce the same result as the example for tcplPlotFits
tcplMakeAeidPlts(aeid = 1, lvl = 6, ordr.fitc = FALSE)

## End(Not run)
```

---

tcplPlotFitc *Plot the fit category tree*

Description

tcplPlotFitc makes a plot showing the level 5 fit categories.

Usage

```
tcplPlotFitc(fitc = NULL, main = NULL, fitc_sub = NULL)
```

Arguments

fitc	Integer, the fit categories
main	Character of length 1, the title (optional)
fitc_sub,	Integer, a subset of fit categories to plot

Note

Suggested device size (inches): width = 10, height = 7.5, pointsize = 9

tcplPlotFits *Plot summary fits based on fit and dose-response data*

Description

tcplPlotFits takes the dose-response and fit data and produces summary plot figures.

Usage

```
tcplPlotFits(dat, agg, flg = NULL, ordr.fitc = FALSE, browse = FALSE)
```

Arguments

dat	data.table, level 4 or level 5 data, see details.
agg	data.table, concentration-response aggregate data, see details.
flg	data.table, level 6 data, see details.
ordr.fitc	Logical, should the fits be ordered by fit category?
browse	Logical, should browser() be called after every plot?

Details

The data for 'dat', 'agg', and 'flg' should be loaded using the tcplLoadData function with the appropriate 'lvl' parameter. See help page for tcplLoadData for more information.

Supplying level 4 data for the 'dat' parameter will result in level 4 plots. Similarly, supp

If fits are not ordered by fit category, they will be ordered by chemical ID. Inputs with multiple assay endpoints will first be ordered by assay endpoint ID.

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

## tcplPlotFits needs data.tables supplying the concentration/response
## data stored in mc4_agg, as well as the fit information from mc4 or mc5.
## Additionally, tcplPlotFits will take level 6 data from mc6 and add the
## flag information to the plots. The following shows how to make level 6
## plots. Omitting the 'flg' parameter would result in level 5 plots, and
## loading level 4, rather than level 5 data, would result in level 4 plots.

l5 <- tcplLoadData(lvl = 5, fld = "aeid", val = 1)
l4_agg <- tcplLoadData(lvl = "agg", fld = "aeid", val = 1)
l6 <- tcplLoadData(lvl = 6, fld = "aeid", val = 1)
## Not run:
pdf(file = "tcplPlotFits.pdf", height = 6, width = 10, pointsize = 10)
```

```
tcplPlotFits(dat = l5, agg = l4_agg, flg = l6)
graphics.off()

## End(Not run)

## While it is most likely the user will want to just save all of the plots
## to view in a PDF, the 'browse' parameter can be used to quickly view
## some plots.

## Start by identifying some sample IDs to plot, then call tcplPlotFits with
## a subset of the data. This browse function is admittedly clunky.
bpa <- tcplLoadChem(field = "chnm", val = "Bisphenol A")[ , spid]
l5_sub <- l5[spid %in% bpa]
## Not run:
tcplPlotFits(dat = l5_sub,
             agg = l4_agg[m4id %in% l5_sub$m4id],
             browse = TRUE)

## End(Not run)

## Reset configuration
options(conf_store)
```

`tcplPlotM4ID` *Plot fit summary plot by m4id*

Description

`tcplPlotM4ID` creates a summary plots for the given m4id(s) by loading the appropriate data from the tcpl databases and sending it to `tcplPlotFits`

Usage

```
tcplPlotM4ID(m4id, lvl = 4L)
```

Arguments

`m4id`	Integer, m4id(s) to plot
`lvl`	Integer, the level of data to plot

Details

A level 4 plot ('lvl' = 4) will plot the concentration series and the applicable curves, without an indication of the activity call or the winning model. Level 4 plots can be created without having done subsequent processing.

Level 5 plots include the level 4 information with the activity call and model selection. The winning model will be highlighted red in the side panel containing the summary statistics. Level 6 plots, in addition the all of the level 4 and 5 information, include the positive flag IDs. If the flag has an associated value, the value will be in parentheses follwing the flag ID.

See Also

tcplPlotFits, tcplMakeAeidPlts

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

tcplPlotM4ID(m4id = 686, lvl = 4) ## Create a level 4 plot
tcplPlotM4ID(m4id = 370, lvl = 5) ## Create a level 5 plot
tcplPlotM4ID(m4id = 322, lvl = 6) ## Create a level 6 plot

## Reset configuration
options(conf_store)
```

---

tcplPlotPlate *Plot plate heatmap*

Description

tcplPlotPlate generates a heatmap of assay plate data

Usage

```
tcplPlotPlate(dat, apid, id = NULL, quant = c(0.001, 0.999))
```

Arguments

dat	data.table containing tcpl data
apid	Character of length 1, the apid to plot
id	Integer of length 1, the assay component id (acid) or assay endpoint id (aeid), depending on level. Only need to specify for multiplexed assays when more than one acid/aeid share an apid.
quant	Numeric vector, the range of data to include in the legend

Details

The legend represents the range of the data supplied to dat, for the applicable ID. The additional horizontal lines on the legend indcate the range of the plotted plate, to show the relation of the plate to the assay as a whole. A plot with a legend specific for the given apid can be created by only supplying the data for the apid of interest to 'dat'.

The quant parameter, by default including 99.8 allows for extreme outliers without losing resolution. Outliers in either direction will be highlighted with a dark ring, as seen in the example. A NULL value for 'quant' will not restrict the data at all, and will use the full range for the legend.

Wells with a well quality of 0 (only applicable for level 1 plots), will have an "X" through their center.

Note

For the optimal output size, use width = 10, height = 10*(2/3), pointsize = 10, units = "in"

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

d1 <- tcplLoadData(lvl = 1, fld = "acid", val = 1)
## Not run:
tcplPlotPlate(dat = d1, apid = "09Apr2014.Plate.17")

## End(Not run)

## Reset configuration
options(conf_store)
```

---

tcplPrepOtpt *Map assay/chemcial ID values to annotation information*

Description

tcplPrepOtpt queries the chemical and assay information from the tcpl database, and maps the annotation information to the given data.

Usage

```
tcplPrepOtpt(dat, ids = NULL)
```

Arguments

dat	data.table, output from tcplLoadData
ids	Character, (optional) a subset of ID fields to map

Details

tcplPrepOtpt is used to map chemical and assay identifiers to their respective names and annotation information to create a human-readable table that is more suitable for an export/output.

By default the function will map sample ID (spid), assay component id (acid), and assay endpoint ID (aeid) values. However, if 'ids' is not null, the function will only attempt to map the ID fields given by 'ids.'

Value

The given data.table with chemical and assay information mapped

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

## Load some example data
d1 <- tcplLoadData(1)

## Check for chemical name in 'dat'
"chnm" %in% names(d1) ## FALSE

## Map chemical annotation only
d2 <- tcplPrepOtpt(d1, ids = "spid")
"chnm" %in% names(d2) ## TRUE
"acnm" %in% names(d2) ## FALSE

## Map all annotations
d3 <- tcplPrepOtpt(d1) ## Also works if function is given d2
"chnm" %in% names(d2) ## TRUE
"acnm" %in% names(d2) ## TRUE

## Reset configuration
options(conf_store)
```

---

tcplRun *Perform data processing*

Description

tcplRun is the function for performing the data processing, for both single-concentration and multiple-concentration formats.

Usage

```
tcplRun(asid = NULL, slvl, elvl, id = NULL, type = "mc",
  mc.cores = NULL, outfile = NULL, runname = NULL)
```

Arguments

asid	Integer, assay source id
slvl	Integer of length 1, the starting level to process
elvl	Integer of length 1, the ending level to process

id	Integer, rather than assay source id, the specific assay component or assay endpoint id(s) (optional)
type	Character of length 1, the data type, "sc" or "mc"
mc.cores	Integer of length 1, the number of cores to use, set to 1 when using Windows operating system
outfile	Character of length 1, the name of the log file (optional)
runname	Character of length 1, the name of the run to be used in the oufile (optional)

Details

The tcplRun function is the core processing function within the package. The function acts as a wrapper for individual processing functions, (ie. mc1, sc1, etc.) that are not exported. If possible, the processing is done in parallel by 'id' by utilizing the mclapply function within the parallel package.

If slvl is less than 4, 'id' is interpreted as acid and if slvl is 4 or greater 'id' is interpreted as aeid. Must give either 'asid' or 'id'. If an id fails no results get loaded into the database, and the id does not get placed into the cue for subsequent level processing.

The 'type' parameter specifies what type of processing to complete: "mc" for multiple-concentration processing, and "sc" for single-concentration processing.

Value

A list containing the results from each level of processing. Each level processed will return a named logical vector, indicating the success of the processing for the id.

---

tcplSubsetChid *Subset level 5 data to a single sample per chemical*

Description

tcplSubsetChid subsets level 5 data to a single tested sample per chemical. In other words, if a chemical is tested more than once (a chid has more than one spid) for a given assay endpoint, the function uses a series of logic to select a single "representative" sample.

Usage

```
tcplSubsetChid(dat, flag = TRUE)
```

Arguments

dat	data.table, a data.table with level 5 data
flag	Integer, the mc6_mthd_id values to go into the flag count, see details for more information

Details

tcplSubsetChid is intended to work with level 5 data that has chemical and assay information mapped with tcplPrepOtpt.

To select a single sample, first a "consensus hit-call" is made by majority rule, with ties defaulting to active. After the chemical-wise hit call is made, the samples corresponding to to chemical-wise hit call are logically ordered using the fit category, the number of the flags, and the modl_ga, then the first sample for every chemical is selected.

The flag param can be used to specify a subset of flags to be used in the flag count. Leaving flag TRUE utilize all the available flags. Setting flag to FALSE will do the subsetting without considering any flags.

Value

A data.table with a single sample for every given chemical-assay pair.

See Also

tcplPrepOtpt

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

## Load the example level 5 data
d1 <- tcplLoadData(lvl = 5, fld = "aeid", val = 2)
d1 <- tcplPrepOtpt(d1)

## Subset to an example of a duplicated chid
d2 <- d1[chid == 559]
d2[ , list(m4id, hitc, fitc, modl_ga)]

## Here the consensus hit-call is 1 (active), and the fit categories are
## all equal. Therefore, if the flags are ignored, the selected sample will
## be the sample with the lowest modl_ga.
tcplSubsetChid(dat = d2, flag = FALSE)[ , list(m4id, modl_ga)]

## Reset configuration
options(conf_store)
```

---

tcplVarMat *Create chemical by assay matrices*

Description

tcplVarMat creates chemical by assay matrices.

Usage

```
tcplVarMat(chid = NULL, aeid = NULL, add.vars = NULL, row.id = "code",
  flag = TRUE, cyto.pars = list(), include.na.chid = FALSE, odir = NULL,
  file.prefix = NULL)
```

Arguments

chid	Integer, chemical ID values to subset on
aeid	Integer, assay endpoint ID values to subset on
add.vars	Character, mc4 or mc5 field(s) not included in the standard list to add additional matrices
row.id	Character, the chemical identifier to use in the output
flag	Integer or Logical of length 1, passed to tcplSubsetChid
cyto.pars	List, named list of arguments passed to tcplCytoPt
include.na.chid	Logical of length 1, whether to include the chemicals not listed in the tcpl databases (ie. controls)
odir	Directory to write comma separated file(s)
file.prefix	Character of length 1, prefix to the file name when odir is not NULL

Details

The tcplVarMat function is used to create chemical by assay matrices for different paramters. The standard list of matrices returned includes:

1. "modl_ga" – The logAC50 (in the gain direction) for the winning model.
2. "hitc" – The hit-call for the winning model.
3. "m4id" – The m4id, listing the concentration series selected by tcplSubsetChid.
4. "zscore" – The z-score based on the output from tcplCytoPt. The formula used for calculating the z-score is $-(modl\_ga - cyto\_pt)/global\_mad$
5. "tested" – 1 or 0, 1 indicating the chemical/assay pair was tested in either the single- or multiple-concentration format
6. "tested_sc" – 1 or 0, 1 indicating the chemical/assay pair was tested in the single-concentration format
7. "tested_mc" – 1 or 0, 1 indicating the chemical/assay pair was tested in the multiple-concentration format
8. "ac50" – a modified AC50 table (in non-log units) where assay/chemical pairs that were not tested, or tested and had a hitcall of 0 or -1 have the value 1e6.
9. "neglogac50" – -log(AC50/1e6) where assay/chemical pairs that were not tested, or tested and had a hitcall of 0 or -1 have the value 0.

To add addtitional matrices, the 'add.vars' parameter can be used to specify the fields from the mc4 or mc5 tables to create matrices for.

When more than one sample is included for a chemical/assay pair, tcplVarMat aggregates multiple samples to a chemical level call utilizing tcplSubsetChid.

By setting odir the function will write out a csv with, naming the file with the convention: "var_Matrix_date.csv" where 'var' is the name of the matrix. A prefix can be added to the output files using the 'file.prefix' paramter.

When a concentration series has a sample id not listed in the tcpl database, and 'include.na.chid' is TRUE, the rowname for that series will be the concatenation of "SPID_" and the spid. Note, if the user gives a subset of chid values to the 'chid' parameter, 'include.na.chid' will be set to FALSE with a warning.

The tcplVarMat function calls both tcplSubsetChid and tcplCytoPt (which separately calls tcplSubsetChid). The input for the tcplVarMat 'flag' parameter is passed to the tcplSubsetChid call used to parse down the data to create the matrices. The tcplSubsetChid called within tcplCytoPt (to parse down the cytotoxicity data used to define the "zscore" matrix) can be modified by passing a separate 'flag' element in the list defined by the 'cyto.pars' parameter.

Value

A list of chemical by assay matrices where the rownames are given by the 'row.id' paramter, and the colnames are given by assay endpoint name (aenm).

See Also

tcplSubsetChid

Examples

```
## Store the current config settings, so they can be reloaded at the end
## of the examples
conf_store <- tcplConfList()
tcplConfDefault()

## Demonstrate the returned values. Note with no "burst" assays defined in
## the example database, the user must provide which aeid values to use
## in calculating the cytotoxicity distributions for the 'zscore' matrix.
tcplVarMat(chid = 1:5, cyto.pars = list(aeid = 1:2))

## Other changes can be made
tcplVarMat(chid = 1:5, row.id = "chnm", cyto.pars = list(aeid = 1:2))
tcplVarMat(chid = 1:5, add.vars = "max_med", cyto.pars = list(aeid = 1:2))

## Reset configuration
options(conf_store)
```

tcplWriteData *Write screening data into the tcpl databases*

Description

tcplWriteData takes a data.table with screening data and writes the data into the given level table in the tcpl databases.

Usage

```
tcplWriteData(dat, lvl, type)
```

Arguments

dat	data.table, the screening data to load
lvl	Integer of length 1, the data processing level
type	Character of length 1, the data type, "sc" or "mc"

Details

This function appends data onto the existing table. It also deletes all the data for any acids or aeids dat contains from the given and all downstream tables.

The data type can be either 'mc' for mutliple concentration data, or 'sc' for single concentration data. Multiple concentration data will be loaded into the level tables, whereas the single concentration will be loaded into the single tables.

Note

This function is not exported and is not inteded to be used by the user. The user should only write level 0 data, which is written with tcplWriteLvl0.

See Also

tcplCascade, tcplAppend, tcplWriteLvl0

tcplWriteLvl0 *Write level 0 screening data into the tcpl databases*

Description

tcplWriteLvl0 takes a data.table with level 0 screening data and writes the data into the level 0 tables in the tcpl databases.

Usage

```
tcplWriteLvl0(dat, type)
```

Arguments

dat	data.table, the screening data to load
type	Character of length 1, the data type, "sc" or "mc"

Details

This function appends data onto the existing table. It also deletes all the data for any acids or aeids dat contains from the given and all downstream tables.

Before writing any data the function maps the assay component source name(s) (acsn) to assay component id (acid), ensures the proper class on each field and checks for every test compound sample id (spid where wllt == "t") in the tcpl chemical database. If field types get changed a warning is given listing the affected fields and they type they were coerced to. If the acsn(s) or spid(s) do not map to the tcpl databases the function will return an error and the data will not be written.

The data type can be either 'mc' for mutliple concentration data, or 'sc' for single concentration data. Multiple concentration data will be loaded into the level tables, whereas the single concentration will be loaded into the single tables.

Note

This function should only be used to load level 0 data.

See Also

tcplCascade, tcplAppend

Index