

# Package ‘tcpl’

March 3, 2022

**Title** ToxCast Data Analysis Pipeline

**Version** 2.1.0

**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** <https://github.com/USEPA/CompTox-ToxCast-tcpl>

**Depends** R (>= 3.2.0)

**Imports** data.table (>= 1.9.4), DBI, RMariaDB, numDeriv, RColorBrewer, utils, stats, methods, graphics, grDevices, sqldf

**Suggests** roxygen2, knitr, prettydoc, rmarkdown, htmlTable, testthat (>= 2.1.0)

**License** GPL-2

**LazyData** true

**RoxygenNote** 7.1.2

**VignetteBuilder** knitr

**Encoding** UTF-8

**NeedsCompilation** no

**Author** Richard S Judson [cre, ths],  
Dayne L Filer [aut],  
Jason Brown [ctb],  
Todd Zurlinden [ctb],  
Parth Kothiya [ctb],  
Woodrow R Setzer [ctb],  
Matthew T Martin [ctb, ths],  
Katie Paul Friedman [ctb]

**Maintainer** Richard S Judson <Judson.Richard@epa.gov>

**Repository** CRAN

**Date/Publication** 2022-03-03 19:10:02 UTC

**R topics documented:**

.buildAssayQ	3
.convertNames	4
.load6DR	4
.plateHeat	5
.prepField	5
blinShift	6
chdat	6
Configure functions	7
flareFunc	8
Hill model utilites	9
interlaceFunc	10
is.odd	11
Load assay information	11
lu	13
lw	13
mc1	14
mc2	15
MC2_Methods	16
mc3	17
MC3_Methods	18
mc4	20
MC4_Methods	21
mc5	22
MC5_Methods	22
mc6	23
MC6_Methods	24
mcdat	25
Method functions	26
Models	28
Query functions	30
Register/update annotation	31
registerMthd	33
sc1	34
SC1_Methods	35
sc2	36
SC2_Methods	37
scdat	38
sink.reset	39
tcplAddModel	39
tcplAICProb	40
tcplAppend	41
tcplCascade	42
tcplCode2CASN	42
tcplCytoPt	43
tcpldbStats	45
tcplDelete	46

tcplFit . . . . .	46
tcplGetAeid . . . . .	48
tcplListFlds . . . . .	48
tcplLoadChem . . . . .	49
tcplLoadClib . . . . .	50
tcplLoadConcUnit . . . . .	51
tcplLoadData . . . . .	52
tcplLoadUnit . . . . .	53
tcplLvlCount . . . . .	54
tcplMakeAeidMultiPlts . . . . .	55
tcplMakeAeidPlts . . . . .	56
tcplMakeChidMultiPlts . . . . .	57
tcplMultiplot . . . . .	58
tcplPlot . . . . .	59
tcplPlotFitc . . . . .	60
tcplPlotFits . . . . .	61
tcplPlotM4ID . . . . .	62
tcplPlotPlate . . . . .	63
tcplPrepOtppt . . . . .	65
tcplRun . . . . .	66
tcplSubsetChid . . . . .	67
tcplVarMat . . . . .	68
tcplWriteData . . . . .	71
tcplWriteLvl0 . . . . .	71

**Index** **73**

.buildAssayQ *Generate query for assay information*

**Description**

.buildAssayQ generates a query string to load assay information

**Usage**

.buildAssayQ(out, tblo, fld = NULL, val = NULL, add.fld = NULL)

**Arguments**

- out                   Character, the default fields to include
- tblo                  Integer, the order to send the fields to prepOutput
- 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

**Value**

A character containing the query to send to tcplQuery

---

.convertNames	<i>Convert assay names to their abbreviations</i>
---------------	---

---

**Description**

.convertNames converts the assay names as they appear in the tcpl database to their respective abbreviations

**Usage**

```
.convertNames(names)
```

**Arguments**

names	Character, strings to convert
-------	-------------------------------

**Value**

The same character vector given with any name strings converted to the abbreviated version

---

.load6DR	<i>Load data for tcpl6</i>
----------	----------------------------

---

**Description**

.load6DR loads dose-response data for tcpl6.

**Usage**

```
.load6DR(ae)
```

**Arguments**

ae	String acid to query on
----	-------------------------

---

.plateHeat                      *Plot plate heatmap*

---

**Description**

Plot plate heatmap, to be used with tcplPlotPlate

**Usage**

.plateHeat(vals, rowi, coli, wllt, wllq, rown, coln, main, arng)

**Arguments**

vals	Numeric, the well values
rowi	Integer, the row index
coli	Integer, the column index
wllt	Character, the well type
wllq	Logical, the well quality
rown	Integer, the number of rows on the plate
coln	Integer, the number of columns on the plate
main	Character of length 1, the title/main
arng	Numeric of length 2, the minimum and maximum values to constrain the color scale

**Note**

Optimized for an output with height = 20/3, width = 10, and pointsize = 10

---

.prepField                      *Paste appropriate table name to field name*

---

**Description**

Paste appropriate table name to field name

**Usage**

.prepField(fld, tbl, db)

**Arguments**

fld	Character, the table fields
tbl	Character, the possible tables
db	Character, the database containing the tables

**Details**

The function loops through the given tables, and for each field *i* it assigns the last table containing *i* to *i*. ORDER OF FLD MATTERS!!

---

blineShift	<i>Shift the baseline to 0</i>
------------	--------------------------------

---

**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 intended to be used by the user.

**See Also**

[mc3\\_mthds](#), [mc3](#)

---

chdat	<i>Chemical library of tested chemicals in the example datasets with the corresponding sample IDs.</i>
-------	--

---

**Description**

Chemical library of tested chemicals in the example datasets with the corresponding sample IDs.

**Usage**

```
chdat
```

**Format**

A data frame with 6 rows and 6 variables:

**spid** sample ID  
**casn** Chemical Abstract Service(CAS) number  
**chnm** chemical name  
**dsstox\_substance\_id** chemical-specific DTXSID  
**code** CAS number compressed into numeric string  
**chid** unique chemical ID number for tcpl

**Source**

Toxcast database

---

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()
```

```
tcplConfExample()
```

```
tcplConfList(show.pass = FALSE)
```

```
tcplConfLoad(list.new = TRUE)
```

```
tcplConfReset()
```

```
tcplConfSave()
```

**Arguments**

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

## Details

Currently, the tcpl package only supports the "MySQL" and "tcplLite" 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 the file. There is more information about system environment variables in [Startup](#) and [Sys.getenv](#). Briefly, the user needs to modify the '.Renviro' 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, e.g. 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 tcplLite database, i.e. local directory, 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	Numeric, 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
## specific 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	<i>Check for odd numbers</i>
--------	------------------------------

---

**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**

<code>fld</code>	Character, the field(s) to query/subset on
<code>val</code>	List, vectors of values for each field to query/subset on. Must be in the same order as 'fld'.
<code>add.fld</code>	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()
TCPLlite <- file.path(system.file(package = "tcpl"), "example")
tcplConf(db = TCPLlite, user = NA, host = NA, drvr = "tcplLite")

## 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	<i>Abbreviation for</i> length(unique(x))
----	---

---

**Description**

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

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

**Usage**

```
lu(x)
```

```
lu(x)
```

**Arguments**

x                    A logical

**Value**

The unique of the TRUE values in x

The unique of the TRUE values in x

**See Also**

[unique, which](#)

[unique, which](#)

Other tcpl abbreviations: [is.odd\(\)](#), [lw\(\)](#), [sink.reset\(\)](#)

Other tcpl abbreviations: [is.odd\(\)](#), [lw\(\)](#), [sink.reset\(\)](#)

---

lw	<i>Abbreviation for</i> length(which(x))
----	--

---

**Description**

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

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

**Usage**

```
lw(x)
```

```
lw(x)
```

**Arguments**

x                    A logical

**Value**

The length of the TRUE values in x

The length of the TRUE values in x

**See Also**

[length, which](#)

[length, which](#)

Other tcpl abbreviations: [is.odd\(\)](#), [lu\(\)](#), [sink.reset\(\)](#)

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 indicating the success of processing and the second element is a data.table containing the processed data

**See Also**

Other multiple-concentration: [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 indicating 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: [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 indicating 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: [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.medncbyconc.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.pc.pval.cor** Calculate resp as  $\frac{cval-bval}{pval}100$ .

**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  $md = \text{abs}(1\text{st centile} - 50\text{th centile})$  or  $\text{abs}(99\text{th centile} - 50\text{th centile})$ , 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**

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

**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 indicating the success of processing and the second element is a data.table containing the processed data

**See Also**

[tcp1Fit, Models](#)

Other multiple-concentration: [mc1\(\)](#), [mc2\(\)](#), [mc3\(\)](#), [mc5\(\)](#), [mc6\(\)](#)

---

MC4\_Methods

*List of level 4 multiple-concentration methods for calculating bmad*

---

**Description**

mc4\_mthds returns a list of methods to be used during level 4 multiple-concentration processing for calculating bmad

**Usage**

```
mc4_mthds()
```

**Details**

The functions contained in the list returned by mc4\_mthds take 'aids' (a numeric vector of aid values) and returns a list of expressions to be executed in the mc4 (not exported) function environment. The functions are described here for reference purposes, The mc4\_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**

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

More information about the level 4 multiple-concentration processing is available in the package vignette, "Pipeline\_Overview."

**bmad.aeid.lowconc.twells** bmad based on two lowest concentration of treatment wells

**bmad.aeid.lowconc.nwells** bmad based on two lowest concentration of nwells

**Note**

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

**See Also**

[mc4](#), [Method functions](#) to query what methods get applied to each aeid

---

mc5	<i>Perform level 5 multiple-concentration processing</i>
-----	--

---

**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.

**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 indicating 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: [mc1\(\)](#), [mc2\(\)](#), [mc3\(\)](#), [mc4\(\)](#), [mc6\(\)](#)

---

MC5_Methods	<i>Load list of level 5 multiple-concentration cutoff methods</i>
-------------	---

---

**Description**

mc5\_mthds returns a list of additional activity cutoff methods to be used during level 5 multiple-concentration processing.

**Usage**

```
mc5_mthds(ae)
```

**Arguments**

ae Integer of length 1, the assay endpoint id

**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\*bmada.

**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).

**bmada5** Add a cutoff value of 5\*bmada.

**bmada6** Add a cutoff value of 6\*bmada.

**bmada10** Add a cutoff value of 10\*bmada.

**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).

**coff\_2.32** Add a cutoff value of 2.32.

**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 indicating 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: `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  $3 \times \text{bmad}$  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  $3 \times \text{bmad}$  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  $3 \times \text{bmad}$  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  $1.2 \times \text{cut-off}$  or the the activity probability 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  $0.8 \times \text{cut-off}$  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  $\log_2(1.5)$  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 acid

---

mcdat	<i>A subset of ToxCast data showing changes in the activity of the intracellular estrogen receptor.</i>
-------	---

---

### Description

The example dataset is used to illustrate how the user can pipeline multiple-concentration data from chemical screening using tcplite.

### Usage

```
mcdat
```

### Format

A data frame with 14183 rows and 10 variables:

**spid** sample ID  
**apid** assay plate ID  
**rowi** well-plate row number  
**coli** well-plate column number  
**wllt** well type

**wllq** well quality  
**conc** concentration in micromolar  
**rval** raw assay component readout value  
**srcf** source file containing the data  
**acsn** assay component source name

### Source

Toxcast database

---

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

<code>lvl</code>	Integer of length 1, the method level
<code>id</code>	Integer, the assay component or assay endpoint id(s)
<code>mthd_id</code>	Integer, the method id(s)
<code>ordr</code>	Integer, the order in which to execute the analysis methods, must be the same length as <code>mthd_id</code> , does not apply to levels 5 or 6
<code>type</code>	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 assignments
tcplMthdClear(lvl = 2, id = 53:55, type = "mc")

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

**Description**

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

**Usage**

```
tcpl0bjCnst(p, resp)
tcpl0bjGnls(p, lconc, resp)
tcpl0bjHill(p, lconc, resp)
tcpl0bjCnst(p, resp)
tcpl0bjGnls(p, lconc, resp)
tcpl0bjHill(p, lconc, resp)
```

**Arguments**

<code>p</code>	Numeric, the parameter values. See details for more information.
<code>resp</code>	Numeric, the response values
<code>lconc</code>	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-likelihood.

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

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

where  $\sigma$  is the scale term. Then the log-likelihood is:

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

Where  $n$  is the number of observations.

**Value**

The log-likelihood.

**Constant Model (cnst)**

tcpl0bjCnst calculates the likelihood for a constant model at 0. The only parameter passed to tcpl0bjCnst by p is the scale term  $\sigma$ . The constant model value  $\mu_i$  for the  $i^{th}$  observation is given by:

$$\mu_i = 0$$

tcpl0bjCnst calculates the likelihood for a constant model at 0. The only parameter passed to tcpl0bjCnst by p is the scale term  $\sigma$ . The constant model value  $\mu_i$  for the  $i^{th}$  observation is given by:

$$\mu_i = 0$$

**Gain-Loss Model (gnls)**

tcpl0bjGnls calculates the likelihood 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 tcpl0bjGnls 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 ( $\sigma$ ). The gain-loss model value  $\mu_i$  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.

tcpl0bjGnls calculates the likelihood 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 tcpl0bjGnls 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 ( $\sigma$ ). The gain-loss model value  $\mu_i$  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 likelihood 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 ( $\sigma$ ). The hill model value  $\mu_i$  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.

tcplObjHill calculates the likelihood 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 ( $\sigma$ ). The hill model value  $\mu_i$  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"),
  tbl = NULL
)
```

```
tcplSendQuery(
  query,
  db = getOption("TCPL_DB"),
  drvr = getOption("TCPL_DRVR"),
  tbl = NULL,
  delete = F
)
```

**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
tbl	Tables to be read queried
delete	Logical of length 1, execute delete on queried table

## Details

Currently, the tcpl package only supports the "MySQL" and "tcplLite" 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()
TCPLlite <- file.path(system.file(package = "tcpl"), "example")
tcplConf(db = TCPLlite, user = NA, host = NA, drvr = "tcplLite")

tcplQuery("SELECT 'Hello World';")

## When using tcplLite, name of table must be passed into tcplQuery
if (conf_store$TCPL_DVR == 'MySQL') {
  tcplQuery("SELECT * FROM assay;")
} else {
  tcplQuery("SELECT * FROM assay;", tbl='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.

## If using the tcplLite DRVR, must specify table name
if (conf_store$TCPL_DRVR == 'MySQL') {
  tcplSendQuery(paste0("DELETE FROM assay_source WHERE asid = ", i1, ";"))
} else {
  qy <- paste0("SELECT * FROM assay_source WHERE NOT asid = ", i1, ";")
  tcplSendQuery(qy, tbl='assay_source', delete=TRUE)
}

## Reset configuration
options(conf_store)

## End(Not run)

```

---

registerMthd	<i>Add a new analysis method</i>
--------------	----------------------------------

---

## Description

registerMthd registers a new analysis method to the tcpl databases.

## Usage

```
registerMthd(lvl, mthd, desc, naddr = 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
naddr	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. 'naddr' only applies to level 6 methods.

---

sc1	<i>Perform level 1 single-concentration processing</i>
-----	--

---

### 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 indicating 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: [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 wlft equal to "p" or "c," by wlft, 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 wlft equal to "p" or "c," by wlft, 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 wlft equal to "m" or "o," by wlft, 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 indicating 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: [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  $\log_2(1.2)$ .

**log10\_1.2** Add a cutoff value of  $\log_{10}(1.2)$ .

**bmada5** Add a cutoff value of  $5 \cdot \text{bmada}$ .

**bmada6** Add a cutoff value of  $6 \cdot \text{bmada}$ .

**bmada10** Add a cutoff value of  $10 \cdot \text{bmada}$ .

**pc30orbmada3** Add a cutoff value of either 30 or  $3 \cdot \text{bmada}$ , 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

---

smdat

*A subset of ToxCast data showing changes in transcription factor activity for multiple targets.*

---

### Description

The example dataset is used to illustrate how the user can pipeline single-concentration data from chemical screening using tcplite.

### Usage

```
smdat
```

### Format

A data frame with 320 rows and 10 variables:

**spid** sample ID

**apid** assay plate ID

**rowi** well-plate row number (N/A)

**coli** well-plate column number (N/A)

**wllt** well type (N/A)

**wllq** well quality (N/A)

**conc** concentration in micromolar

**rval** raw assay component readout value

**srcf** source file containing the data

**acsn** assay component source name

### Source

Toxcast database

---

sink.reset	<i>Reset all sinks</i>
------------	------------------------

---

### 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	<i>Draw a tcpl Model onto an existing plot</i>
--------------	--

---

### 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 mod1 is NULL, the function checks pars\$mod1 and will return an error if pars\$mod1 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 ad1 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, mod1 = "hill")
points(r2 ~ logc)
tcplAddModel(pars = p2, mod1 = "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, mod1 = "hill", adj = 1/10)
points(r2/log2(5) ~ logc)
tcplAddModel(pars = p2, mod1 = "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)
```

---

<code>tcplAppend</code>	<i>Append rows to a table</i>
-------------------------	-------------------------------

---

**Description**

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

**Usage**

```
tcplAppend(dat, tbl, db, lvl = NULL)
```

**Arguments**

<code>dat</code>	<code>data.table</code> , the data to append to a table
<code>tbl</code>	Character of length 1, the table to append to
<code>db</code>	Character of length 1, the database containing <code>tbl</code>
<code>lvl</code>	Usually Integer to indicate what level to auto-increment

**Note**

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

---

tcp1Cascade	<i>Do a cascading delete on tcp1 screening data</i>
-------------	---

---

**Description**

tcp1Cascade deletes the data for the given id(s) starting at the processing level given. The delete will cascade through all subsequent tables.

**Usage**

```
tcp1Cascade(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 multiple 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 aacid(s).

**Note**

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

---

tcp1Code2CASN	<i>Convert chemical code to CAS Registry Number</i>
---------------	---

---

**Description**

tcp1Code2CASN takes a code and converts it CAS Registry Number.

**Usage**

```
tcp1Code2CASN(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 chemicals 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 example 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 activity 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 chemical to be used in calculating the "global MAD."

default.pt              Numeric of length 1, the default cytotoxicity point value

## Details

tcp1CytoPt 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 `tcp1SubsetChid` 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 tcp1CytoPt 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.

tcp1CytoPt 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 its 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), tcp1CytoPt 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 chemical code.
3. "chnm" – The chemical name.
4. "casn" – The chemical CASRN.
5. "med" – The median of the "burst" endpoint  $\log(\text{AC50})$  ("modl\_ga" in the level 5 output) values.
6. "mad" – The MAD of the "burst" endpoint  $\log(\text{AC50})$  values.
7. "ntst" – The number of "burst" endpoints tested.
8. "nhit" – The number of active "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" –  $10^{\text{cyto\_pt}}$
13. "lower\_bnd\_um" –  $10^{\text{cyto\_pt} - 3\text{global\_mad}}$

## 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()

## Can only calculate the cytotox burst if using the MySQL database and
## TCPL_DRVR == 'MySQL'

if (getOption("TCPL_DRVR") == "MySQL") {

## 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 activity in the
## up direction, and the other is activity 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)

```

---

tcpldbStats

*Get summary statistics for the database*


---

**Description**

tcpldbStats takes a string(*type*) and an optional parameter(*val*) to return the summary statistics on the entire tcplLite database. When *type* = "all" the *val* is ignored. the function returns the number of distinct *spid* and *aeids* in the database at each level. When *type* = "aeid", the *val* parameter has to be a valid *aeid* in the database. The function returns a table consisting of the number of distinct *spids* at each level of processing for the *aeid* given in '*val*'. When *type* = "spid", the *val* parameter has to be a valid *spid* in the database. The function returns a table consisting of the number of distinct *aeids* at each level of processing for the given *spid* in '*val*'.

**Usage**

```
tcpldbStats(type = "all", val = NULL)
```

**Arguments**

type	String either "all", "aeid" or "spid"
val	integer if type = "aeid" , string if type = "spid"

---

tcplDelete	<i>Delete rows from tcpl databases</i>
------------	--

---

**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	<i>Fit the data with the constant, hill, and gain-loss models</i>
---------	---

---

**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,
  bidirectional = FALSE,
  verbose = 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
bidirectional	Boolean If TRUE, bidirectional negative data before fitting (default=FALSE) The original version of the code required the data to start at small values and rise, so that negative curves had to be bidirectionalped outside the function, and TOP was always positive. Setting bidirectional to TRUE allows both rising and falling curves
verbose	Boolean If TRUE print warning messages
...	Any other data to be included in list output.

**Details**

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")
```

---

tcplGetAeid	<i>get Aeid for endpoint name</i>
-------------	-----------------------------------

---

### Description

tcplGetAeid takes a string(name) and finds the assay component endpoint names that match the string and the auids associated with those names. The function performs a regular expression like matching for strings in the assay component endpoint name column in the assay component endpoint table.

### Usage

```
tcplGetAeid(name)
```

### Arguments

name	A string that will be matched to the assay component endpoint name
------	--

### Examples

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

## Search for aenm (assay name) case insensitive
tcplGetAeid("TOX21")
tcplGetAeid("tox21")

## Reset configuration
options(conf_store)
```

---

tcplListFlds	<i>Load the field names for a table</i>
--------------	---

---

### 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	<i>Load sample/chemical information</i>
--------------	---

---

## Description

tcplLoadChem queries the tcpl database and returns the chemical 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.

In the MySQL environment the user should be able to give partial chemical 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()
tcplConfExample()

## 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)

## In addition, the user can retrieve only the registered chemicals from the chemical table
tcplLoadChem(field = 'chem.only')

## Other examples:
tcplLoadChem(field = "chnm", val = "Bisphenol A")
tcplLoadChem(field = "chid", val = 20182)

## Reset configuration
options(conf_store)
```

---

tcplLoadClib

*Load chemical library information*

---

## Description

tcplLoadClib queries the tcpl databases and returns information about the chemical 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 chemical 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 chemical 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()
tcplConfExample()

## 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 = "TOXCAST")
tcplLoadClib(field = "chid", val = 20182)

## Reset configuration
options(conf_store)
```

---

tcplLoadConcUnit      *Load concentration units for assay endpoints*

---

**Description**

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

**Usage**

```
tcplLoadConcUnit(spid)
```

**Arguments**

spid                    Integer, assay endpoint ids

**Value**

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

**See Also**

[tcplQuery](#), [data.table](#)

---

tcplLoadData	<i>Load tcpl data</i>
--------------	-----------------------

---

**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 multiple 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
mc 7 mc4, mc7

```

**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()
tcplConfExample()

## 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)

```

**Description**

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

**Usage**

```
tcpLoadUnit(aeid)
```

**Arguments**

aeid                    Integer, assay endpoint ids

**Value**

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

**See Also**

[tcpQuery](#), [data.table](#)

---

tcpLvlCount	<i>Load tcp level counts</i>
-------------	------------------------------

---

**Description**

tcpLvlCount queries the tcp databases and returns a data frame with count totals for the given levels and data type.

**Usage**

```
tcpLvlCount(lvls = NULL, type = "mc")
```

**Arguments**

lvls                    Integer or list of Integers, The levels of data to load  
 type                    Character of length 1, the data type, "sc" or "mc"

**Details**

The data type can be either 'mc' for mutiple concentration data, or 'sc' for single concentration data.

Leaving lvls NULL will return all data.

**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()
TCPLlite <- file.path(system.file(package = "tcpl"), "example")
tcplConf(db = TCPLlite, user = NA, host = NA, drvr = "tcplLite")

## Get all counts for level 1 for multiple-concentration
tcplLvlCount(lvls = 1)

## Not run:
## Get all counts for levels 4 through 7 for multiple-concentration
tcplLvlCount(lvls = 4:7)

## Get all counts for multiple-concentration data, note 'mc' is the
## default value for type
tcplLvlCount()

## End(Not run)

## Reset configuration
options(conf_store)
```

---

tcplMakeAeidMultiPlts *Create a .pdf with all dose-response plots for a given aeid, 6 per page*

---

**Description**

tcplMakeAeidMultiPlts Create a .pdf with all dose-response plots for a given aeid

**Usage**

```
tcplMakeAeidMultiPlts(
  aeid,
  lvl = 4L,
  fname = NULL,
  odir = getwd(),
  clib = NULL,
  hitc.all = TRUE
)
```

**Arguments**

aeid	Integer of length 1, the assay endpoint id
lvl	Integer of length 1, the data level to use (4-7)
fname	Character, the filename
odir	The directory to save the .pdf file in
clib	Character, the chemical library to subset on, see <a href="#">tcplLoadClib</a> for more information.
hitc.all	If FALSE, only plots with hitc==1 will be displayed

**Details**

tcplMakeAeidMultiPlts provides a wrapper for [tcplMultiplot](#), 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.

---

tcplMakeAeidPlts	<i>Create a .pdf with dose-response plots</i>
------------------	---

---

**Description**

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

**Usage**

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

**Arguments**

aeid	Integer of length 1 or 2, the assay endpoint id
compare	Boolean to for comparison of aeids if length(aeid)>1
lvl	Integer of length 1, the data level to use (4-7). Only level 5-6 valid for compare aeids.
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
cnst	Constant hline to draw on plot <a href="#">tcplLoadClib</a> 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

Note, only level 5 or level 6 is valid for comparing 2 aeids.

### Examples

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

## End(Not run)

## Not run:
## Compare two aeids on same plots
tcplMakeAeidPlts(aeid = c(1,2), compare=T, lvl = 6)

## End(Not run)
```

---

tcplMakeChidMultiPlts *Create a .pdf with all dose-response plots for a given chid, 6 per page*

---

### Description

tcplMakeChidMultiPlts Create a .pdf with all dose-response plots for a given chid

### Usage

```
tcplMakeChidMultiPlts(
  chid,
  lvl = 4L,
  fname = NULL,
  odir = getwd(),
  clib = NULL,
  hitc.all = TRUE
)
```

**Arguments**

chid	Integer of length 1, the chemical id
lvl	Integer of length 1, the data level to use (4-7)
fname	Character, the filename
odir	The directory to save the .pdf file in
clib	Character, the chemical library to subset on, see <a href="#">tcplLoadClib</a> for more information.
hitc.all	If FALSE, only plots with hitc==1 will be displayed

**Details**

tcplMakeChidMultiPlts provides a wrapper for [tcplMultiplot](#), 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.

---

tcplMultiplot	<i>Plot summary fits based on fit and dose-response data</i>
---------------	--

---

**Description**

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

**Usage**

```
tcplMultiplot(dat, agg, flg = NULL, boot = NULL, browse = FALSE, hitc.all)
```

**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.
boot	data.table, level 7 data, see details.
browse	Logical, should browser() be called after every plot?
hitc.all	Logical, if FALSE, only plots with hitc==1 will be displayed

**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.

If dat contains only one aeid, plots will be ordered by chemical name (chm). Otherwise, plots are ordered by assay endpoint name (aenm). ## 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.

---

```
tcplPlot          #----- Generic
                  Plotting Function for tcpl
```

---

## Description

tcplLoadData queries the tcpl databases and returns a plot for the given level and data type.

## Usage

```
tcplPlot(
  lvl = 4,
  fld = NULL,
  val = NULL,
  type = "mc",
  by = NULL,
  output = c("console", "pdf"),
  fileprefix = paste0("tcplPlot_", Sys.Date()),
  multi = FALSE
)
```

## 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"
by	Parameter to divide files into e.g. aeid
output	how should the output be presented
fileprefix	prefix of filename
multi	Boolean, if multi is TRUE output 6 plots per page

## Details

The data type can be either 'mc' for multiple 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.

### Examples

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

tcpIPlot(lvl = 4, fld = "m4id", val = c(18609966))## Create a level 4 plot

## Reset configuration
options(conf_store)
```

---

tcpIPlotFitc	<i>Plot the fit category tree</i>
--------------	-----------------------------------

---

### Description

tcpIPlotFitc makes a plot showing the level 5 fit categories.

### Usage

```
tcpIPlotFitc(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

### Examples

```
## Not run:
## Plot visualization of fit categories for all level 5 data
tcpIPlotFitc(fitc = tcpILoadData(5)$fitc)

## End(Not run)
```

---

`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,  
  boot = NULL,  
  ordr.fitc = FALSE,  
  browse = FALSE,  
  cnst = NULL,  
  orig.aeid = NULL,  
  compare = F  
)
```

## Arguments

<code>dat</code>	data.table, level 4 or level 5 data, see details.
<code>agg</code>	data.table, concentration-response aggregate data, see details.
<code>flg</code>	data.table, level 6 data, see details.
<code>boot</code>	data.table, level 7 data, see details.
<code>ordr.fitc</code>	Logical, should the fits be ordered by fit category?
<code>browse</code>	Logical, should <code>browser()</code> be called after every plot?
<code>cnst</code>	Constant hline to draw on plot
<code>orig.aeid</code>	Original aeid list from <code>tcplMakeAeidPlts</code> to maintain order
<code>compare</code>	boolean to determine if aeids should be compared on same 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 can take level 6 data from mc6 and add the
## flag information to the plots. The following shows how to make level 5
## plots. Adding the 'flg' parameter would result in level 6 plots, and
## loading level 4, rather than level 5 data, would result in level 4 plots.

l5 <- tcplLoadData(lvl = 5, fld = "m4id", val = 18609966)
l4_agg <- tcplLoadData(lvl = "agg", fld = "m4id", val = 18609966)

## Not run:
pdf(file = "tcplPlotFits.pdf", height = 6, width = 10, pointsize = 10)
tcplPlotFits(dat = l5, agg = l4_agg)
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 to 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 following the flag ID. Level 7 plots in addition to all of the level 4, 5, and 6 information, include the AC50 confidence interval and hit percentage information from bootstrapping.

**See Also**

[tcplPlotFits](#), [tcplMakeAeidPlts](#)

**Examples**

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

tcplPlotM4ID(m4id = 18609966, lvl = 4) ## Create a level 4 plot
tcplPlotM4ID(m4id = 18609966, lvl = 5) ## Create a level 5 plot
tcplPlotM4ID(m4id = 18609966, 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 indicate 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)
```



---

`tcplPrep0tpt`*Map assay/chemical ID values to annotation information*

---

## Description

`tcplPrep0tpt` queries the chemical and assay information from the `tcpl` database, and maps the annotation information to the given data.

## Usage

```
tcplPrep0tpt(dat, ids = NULL)
```

## Arguments

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

## Details

`tcplPrep0tpt` 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()
tcplConfExample()

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

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

#' ## Map all annotations
d2 <- tcplPrep0tpt(d1) ##
"chnm" %in% names(d2) ## TRUE
"acnm" %in% names(d2) ## TRUE
```

```
## Map chemical annotation only
d3 <- tcplPrep0tpt(d1, ids = "spid")
"chnm" %in% names(d3) ## TRUE
"acnm" %in% names(d3) ## FALSE

## 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 end-point 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	<i>Subset level 5 data to a single sample per chemical</i>
----------------	--

---

**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, type = "mc")
```

**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
type	Character of length 1, the data type, "sc" or "mc"

**Details**

tcplSubsetChid is intended to work with level 5 data that has chemical and assay information mapped with `tcplPrep0tpt`.

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**

[tcpIPrep0tpt](#)

**Examples**

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

## Load the example level 5 data
d1 <- tcpILoadData(lvl = 5, fld = "aeid", val = 797)
d1 <- tcpIPrep0tpt(d1)

## Subset to an example of a duplicated chid
d2 <- d1[chid == 20182]
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.
tcpISubsetChid(dat = d2, flag = FALSE)[ , list(m4id, modl_ga)]

## Reset configuration
options(conf_store)
```

---

tcpIVarMat

*Create chemical by assay matrices*

---

**Description**

tcpIVarMat creates chemical by assay matrices.

**Usage**

```
tcpIVarMat(
  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 <a href="#">tcplSubsetChid</a>
cyto.pars	List, named list of arguments passed to <a href="#">tcplCytoPt</a>
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 parameters. 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 additional 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' parameter.

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' parameter, 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()
TCPLlite <- file.path(system.file(package = "tcpl"), "example")
tcplConf(db = TCPLlite, user = NA, host = NA, drvr = "tcplLite")
## Not run:
## 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))

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

---

tcplWriteData	<i>Write screening data into the tcpl databases</i>
---------------	---

---

### 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 aoids dat contains from the given and all downstream tables.

The data type can be either 'mc' for multiple 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 intended to be used by the user. The user should only write level 0 data, which is written with [tcplWriteLv10](#).

### See Also

[tcplCascade](#), [tcplAppend](#), [tcplWriteLv10](#)

---

tcplWriteLv10	<i>Write level 0 screening data into the tcpl databases</i>
---------------	---

---

### Description

tcplWriteLv10 takes a data.table with level 0 screening data and writes the data into the level 0 tables in the tcpl databases.

### Usage

```
tcplWriteLv10(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 acids 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 wlt == "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 mutiple 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

- \* **data processing functions**
  - tcplRun, 66
- \* **datasets**
  - chdat, 6
  - mcdat, 25
  - scdat, 38
- \* **multiple-concentration**
  - mc1, 14
  - mc2, 15
  - mc3, 17
  - mc4, 20
  - mc5, 22
  - mc6, 23
- \* **single-concentration**
  - sc1, 34
  - sc2, 36
- \* **tcpl abbreviations**
  - is.odd, 11
  - lu, 13
  - lw, 13
  - sink.reset, 39
- .buildAssayQ, 3
- .convertNames, 4
- .load6DR, 4
- .plateHeat, 5
- .prepField, 5
- AIC, 41
- blinShift, 6
- chdat, 6
- Configure functions, 7
- constrOptim, 28, 47
- data.table, 52–55
- flareFunc, 8
- Hill model utilites, 9
- interlaceFunc, 10
- is.odd, 11, 13, 14, 39
- length, 14
- Load assay information, 11
- lu, 11, 13, 14, 39
- lw, 11, 13, 13, 39
- mc1, 14, 15, 17, 21, 22, 24
- mc2, 15, 15, 17, 21, 22, 24
- MC2\_Methods, 15, 16
- mc2\_mthds (MC2\_Methods), 16
- mc3, 6, 15, 17, 20–22, 24
- MC3\_Methods, 17, 18
- mc3\_mthds, 6
- mc3\_mthds (MC3\_Methods), 18
- mc4, 15, 17, 20, 22, 24
- MC4\_Methods, 21
- mc4\_mthds (MC4\_Methods), 21
- mc5, 15, 17, 21, 22, 23, 24
- MC5\_Methods, 22, 22
- mc5\_mthds (MC5\_Methods), 22
- mc6, 9, 10, 15, 17, 21, 22, 23, 25
- MC6\_Methods, 9, 10, 24, 24
- mc6\_mthds (MC6\_Methods), 24
- mcdat, 25
- mclapply, 67
- Method functions, 9, 10, 15, 17, 20, 22–25, 26, 34, 36–38
- Models, 20, 21, 28, 39, 40
- Query functions, 30
- Register/update annotation, 31
- registerMthd, 33
- sc1, 34, 36, 37
- SC1\_Methods, 34, 35
- sc1\_mthds (SC1\_Methods), 35
- sc2, 34, 36, 38
- SC2\_Methods, 37, 37

- sc2\_mthds (SC2\_Methods), 37
- scdat, 38
- sink, 39
- sink.number, 39
- sink.reset, 11, 13, 14, 39
- Startup, 8
- Sys.getenv, 8
  
- tcplAddModel, 39
- tcplAICProb, 40
- tcplAppend, 41, 71, 72
- tcplCascade, 14, 15, 17, 20, 22, 23, 34, 36, 42, 71, 72
- tcplCode2CASN, 42
- tcplConf (Configure functions), 7
- tcplConfDefault (Configure functions), 7
- tcplConfExample (Configure functions), 7
- tcplConfList (Configure functions), 7
- tcplConfLoad (Configure functions), 7
- tcplConfReset (Configure functions), 7
- tcplConfSave (Configure functions), 7
- tcplCytoPt, 43, 69
- tcpldbStats, 45
- tcplDelete, 46
- tcplFit, 21, 28, 41, 46
- tcplGetAeid, 48
- tcplHillACXX (Hill model utilites), 9
- tcplHillConc (Hill model utilites), 9
- tcplHillVal (Hill model utilites), 9
- tcplListFlds, 48
- tcplLoadAcid (Load assay information), 11
- tcplLoadAeid (Load assay information), 11
- tcplLoadAid (Load assay information), 11
- tcplLoadAsid (Load assay information), 11
- tcplLoadChem, 49
- tcplLoadClib, 50, 56–58
- tcplLoadConcUnit, 51
- tcplLoadData, 52, 58, 61, 65
- tcplLoadUnit, 53
- tcplLvlCount, 54
- tcplMakeAeidMultiPlts, 55
- tcplMakeAeidPlts, 56, 63
- tcplMakeChidMultiPlts, 57
- tcplMthdAssign (Method functions), 26
- tcplMthdClear (Method functions), 26
- tcplMthdList (Method functions), 26
- tcplMthdLoad (Method functions), 26
- tcplMultiplot, 56, 58, 58
- tcplObjCnst, 47
- tcplObjCnst (Models), 28
- tcplObjGnls, 47
- tcplObjGnls (Models), 28
- tcplObjHill, 47
- tcplObjHill (Models), 28
- tcplPlot, 59
- tcplPlotFitc, 60
- tcplPlotFits, 40, 57, 61, 62, 63
- tcplPlotM4ID, 62
- tcplPlotPlate, 63
- tcplPrepOtppt, 65, 67, 68
- tcplQuery, 52–55
- tcplQuery (Query functions), 30
- tcplRegister (Register/update annotation), 31
- tcplRun, 14, 15, 17, 20, 22, 23, 34, 36, 66
- tcplSendQuery, 46
- tcplSendQuery (Query functions), 30
- tcplSubsetChid, 43, 44, 67, 69, 70
- tcplUpdate (Register/update annotation), 31
- tcplVarMat, 68
- tcplWriteData, 71
- tcplWriteLvl0, 71, 71
  
- unique, 13
  
- which, 13, 14