

# Package ‘GladiaTOX’

July 27, 2025

**Type** Package

**biocViews** Software, WorkflowStep, Normalization, Preprocessing,  
QualityControl

**Title** R Package for Processing High Content Screening data

**Version** 1.24.1

**Author** Vincenzo Belcastro [aut, cre],  
Dayne L Filer [aut],  
Stephane Cano [aut]

**Maintainer** PMP S.A. R Support <DL.RSupport@pmi.com>

**Description** GladiaTOX R package is an open-source, flexible solution to high-content screening data processing and reporting in biomedical research. GladiaTOX takes advantage of the tcpl core functionalities and provides a number of extensions: it provides a web-service solution to fetch raw data; it computes severity scores and exports ToxPi formatted files; furthermore it contains a suite of functionalities to generate pdf reports for quality control and data processing.

**License** GPL-2

**Depends** R (>= 3.6.0), data.table (>= 1.9.4)

**Imports** DBI, RMariaDB, RSQLite, numDeriv, RColorBrewer, parallel,  
stats, methods, graphics, grDevices, xtable, tools, brew,  
stringr, RJSONIO, ggplot2, ggrepel, tidyr, utils, RCurl, XML

**LazyData** FALSE

**VignetteBuilder** knitr

**Suggests** roxygen2, knitr, rmarkdown, testthat, BiocStyle

**NeedsCompilation** no

**RoxygenNote** 6.1.1

**URL** <https://github.com/philipmorrisintl/GladiaTOX>

**git\_url** <https://git.bioconductor.org/packages/GladiaTOX>

**git\_branch** RELEASE\_3\_21

**git\_last\_commit** fe7e628

**git\_last\_commit\_date** 2025-07-14

**Repository** Bioconductor 3.21

**Date/Publication** 2025-07-27

## Contents

assignDefaultMthds . . . . .	3
blinShift . . . . .	4
buildAssayTab . . . . .	5
Configure functions . . . . .	6
deleteStudy . . . . .	8
exportResultForToxpiGUI . . . . .	9
exportResultTable . . . . .	10
flareFunc . . . . .	10
glCheckInput . . . . .	11
glComputeToxInd . . . . .	12
glLoadInput . . . . .	13
glPlotPie . . . . .	13
glPlotPieLogo . . . . .	14
glPlotPosCtrl . . . . .	15
glPlotPosCtrlMEC . . . . .	16
glPlotStat . . . . .	17
glPlotToxInd . . . . .	18
gtoxAddModel . . . . .	19
gtoxAICProb . . . . .	20
gtoxAppend . . . . .	21
gtoxCalcVmad . . . . .	22
gtoxCascade . . . . .	23
gtoxCode2CASN . . . . .	23
gtoxDelete . . . . .	24
gtoxFit . . . . .	25
gtoxHillACXX . . . . .	26
gtoxImportThermoDB . . . . .	27
gtoxListFlds . . . . .	28
gtoxLoadApid . . . . .	29
gtoxLoadChem . . . . .	29
gtoxLoadClib . . . . .	31
gtoxLoadData . . . . .	32
gtoxLoadUnit . . . . .	33
gtoxLoadVehicle . . . . .	34
gtoxLoadVmad . . . . .	35
gtoxLoadWaid . . . . .	35
gtoxMakeAeidPlts . . . . .	36
gtoxMthdAssign . . . . .	37
gtoxPlotErrBar . . . . .	39
gtoxPlotFitc . . . . .	40
gtoxPlotFits . . . . .	40

gtoxPlotM4ID . . . . .	42
gtoxPlotPie . . . . .	43
gtoxPlotPieLgnd . . . . .	44
gtoxPlotPlate . . . . .	45
gtoxPlotWin . . . . .	46
gtoxPrepOtp . . . . .	47
gtoxQuery . . . . .	48
gtoxRegister . . . . .	49
gtoxReport . . . . .	51
gtoxRun . . . . .	52
gtoxSetWllq . . . . .	53
gtoxSubsetChid . . . . .	54
gtoxWriteData . . . . .	55
interlaceFunc . . . . .	56
is.odd . . . . .	57
Load assay information . . . . .	58
loadAnnot . . . . .	59
lu . . . . .	60
lw . . . . .	61
mc1 . . . . .	61
mc2 . . . . .	62
MC2_Methods . . . . .	63
mc3 . . . . .	64
MC3_Methods . . . . .	65
mc4 . . . . .	67
mc5 . . . . .	68
MC5_Methods . . . . .	69
mc6 . . . . .	70
MC6_Methods . . . . .	71
Models . . . . .	72
prepareDatForDB . . . . .	74
registerMthd . . . . .	75
sc1 . . . . .	76
SC1_Methods . . . . .	77
sc2 . . . . .	78
SC2_Methods . . . . .	79
sink_reset . . . . .	80

**Index****81**


---

assignDefaultMthds	<i>Assign default processing methods</i>
--------------------	--

---

**Description**

Function to assign default processing method to asid in input

**Usage**

```
assignDefaultMthds(asid, params = NULL)
```

**Arguments**

asid	Integer, the asid value(s) to which assign the default methods
params	Parameters for level 2, 3, and 5 processing

**Details**

This function loads all components and endpoints for the given asid(s) in the database, and assigns a default set of processing methods to them.

This function will overwrite any previously assigned methods.

By default, each assay will receive 'none' at level 2. Level 3 data will receive, in order, 'bval.pmi' (39), 'resp.fc' (9), 'resp.log2' (7), and for endpoints with "down" analysis direction, 'resp.multneg1' (6).

**Value**

None

**Examples**

```
## Prepare for analysis before QC + process data
assignDefaultMthds(asid = 1L)

## Process data
gtoxRun(asid = 1L, slvl = 1, elvl = 6, mc.cores = 2)
```

---

blineShift

*Shift the baseline to 0*


---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

A numeric vector containing the shifted response values

**Note**

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

**See Also**

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

---

buildAssayTab	<i>Prepare assay table mapping info</i>
---------------	---

---

**Description**

This function parses plate annotations and create a mapping between assay endpoints and channels

**Usage**

```
buildAssayTab(plate.mtd, chn.map)
```

**Arguments**

plate.mtd	Legacy study annotation file from biobanking
chn.map	List of endpoints to thermo channels mapping

**Details**

Function used only when processing historical data

**Value**

Table with assay information

**Examples**

```
## Load sample data
load(system.file("extdata", "data_for_vignette.rda", package="GladiaTOX"))

# Build assay table
assay <- buildAssayTab(plate, chnmap)
```

---

Configure functions      *Functions for configuring the gtox package*


---

**Description**

These functions are used to configure the gtox settings.

Load the current configuration file

**Usage**

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

```
gtoxConfDefault()
```

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

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

```
gtoxConfReset()
```

```
gtoxConfSave()
```

**Arguments**

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

**Details**

Currently, the gtox package only supports the "MariaDB" and "SQLite" database drivers.

The settings can be stored in a configuration file to make the using the package more user-friendly. To create the configuration file, the user must first create a system environment variable ('TCPL\_CONF') that points to 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 gtoxConf in the home directory, eg. TCPL\_CONF=~/.gtoxConf. Note, '~' may not indicate the home directory on every operating system. Once the environment variable is added, the user can change the

settings using `gtoxConf`, then save the settings to the file given by the `TCPL_CONF` environment variable running `gtoxConfSave()`.

## Value

None

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

`gtoxConfSave` modifies the configuration file to reflect the current `gtox` settings.

`gtoxConfList` lists the values assigned to the `gtox` global options.

`gtoxConfLoad` updates the `gtox` settings to reflect the current configuration file.

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

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

## Examples

```
gtoxConfList() # List configuration parameters

## Configure database
sqlite <- file.path(system.file(package="GladiaTOX"),
  "sql",
  "gladiatodb.sqlite")
gtoxConf(db=sqlite, user=NA, host=NA, drvr="SQLite")

## Configure database with default parameters
gtoxConfDefault()

## List configuration of database parameters
gtoxConfList()

## Set the environment variable pointing to the configuration file
Sys.setenv(TCPL_CONF=file.path(system.file(package="GladiaTOX"), "gtoxConf"))

## Configure database
gtoxConfLoad()

## Set the environment variable pointing to the configuration file
Sys.setenv(TCPL_CONF=file.path(system.file(package="GladiaTOX"), "gtoxConf"))

## Configure database
gtoxConfReset()
```

```
## Set the environment variable pointing to the configuration file
Sys.setenv(TCPL_CONF=file.path(system.file(package="GladiaTOX"),"gtoxConf"))

## Configure database
gtoxConfSave()
```

---

deleteStudy	<i>Completely remove all data for a study</i>
-------------	---

---

### Description

deleteStudy completely removes all data for a study from the database.

### Usage

```
deleteStudy(asid, db = NULL)
```

### Arguments

asid	The assay source/study ID
db	(optional) the database to delete from, defaults to the current database settings

### Details

Cannot be undone. Please use carefully. Not exported, as this is intended for development and should not be used with real data.

### Value

None

### Examples

```
## Not run:
## Load sample data
load(system.file("extdata", "data_for_vignette.rda", package="GladiaTOX"))

## Build assay table
assay <- buildAssayTab(plate, chnmap)

## Set study parameters
std.nm <- "SampleStudy" # study name
phs.nm <- "PhaseII" # study phase

## Load annotation in gtoxDB
loadAnnot(plate, assay, NULL)
```



```
## Delete previously loaded study data
asid = gtoxLoadAsid(fld=c("asnm", "asph"), val=list(std.nm, phs.nm))$asid
if(length(asid)>0){ deleteStudy(asid=asid) }

## End(Not run)
```

---

exportResultForToxpiGUI

*Create the result table for the asi in input*

---

## Description

This function export results

## Usage

```
exportResultForToxpiGUI(asid, tp, outfile, stat)
```

## Arguments

asid	Assay source id
tp	Time point
outfile	Path to the output file
stat	Character vector of statistic to export

## Details

This function is useful to export results in a table format

## Value

None

## Examples

```
## Export MEC (or AC50) values to be visualized in ToxPiGUI
conf_store <- gtoxConfList()
gtoxConfDefault()

out <- "export_for_toxpiGUI.csv"
exportResultForToxpiGUI(asid=1L, tp="4h", outfile=out, stat=quote(modl_acc))

## Reset configuration
options(conf_store)
```

---

exportResultTable	Create the result table for the asi in input
-------------------	--

---

**Description**

This function export results

**Usage**

```
exportResultTable(asiid, stats, outfile)
```

**Arguments**

asiid	Assay source id
stats	Statistics to export
outfile	Path to the output file

**Details**

This funtion is useful to export results in a table format

**Value**

None

**Examples**

```
outfile <- "export_stats.csv"
exportResultTable(asiid=1L, stats=c("mdl_acc", "mdl_ga"), outfile=outfile)
```

---

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

**Value**

None

**See Also**

[MC6\\_Methods](#), [Method functions](#), [mc6](#)

---

glCheckInput

*Check validity of input file*

---

**Description**

This function check the structure and content of an input file.

**Usage**

```
glCheckInput(file)
```

**Arguments**

file	file URL
------	----------

**Details**

This function is useful to check the structure and content of an input file from the GladiaTOX GUI

**Value**

List of error messages in JSON format

---

`glComputeToxInd`*Create toxicological indicator values for all chemicals in input*

---

## Description

This function computes the toxicological indicator value for the assay source id in input.

## Usage

```
glComputeToxInd(asid, tp = NULL, stat = quote(modl_acc))
```

## Arguments

<code>asid</code>	assay source id
<code>tp</code>	Time point to report
<code>stat</code>	statistic to plot

## Details

This function is useful to compute toxicological indicator values. These values, for each chemical, represent an average impact of the chemical across the list of endpoints tested. The function transform the data to minus log scale. Hence the larger the indicator value, larger is the impact of the chemical.

## Value

A data.table with toxicological severity index for each chemical.

## Examples

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

## Compute toxicological severity index
dat <- glComputeToxInd(asid = 1L)
dat[]
```

---

glLoadInput	<i>Check validity of input file</i>
-------------	-------------------------------------

---

**Description**

This function check the structure and content of an input file.

**Usage**

```
glLoadInput(file = NULL, studyname = "samplestudy",  
            phasename = "samplephase", tab = NULL)
```

**Arguments**

file	file URL
studyname	Name of the study
phasename	Name of the phase
tab	input table is file URL is not provided

**Details**

This function is useful to load an input file in the GladiaTOX GUI

**Value**

List of error messages in JSON format

---

glPlotPie	<i>Pie chart for Minimal Effective Concentrations (MEC) and AC50 plot</i>
-----------	---

---

**Description**

This function plots MEC values

**Usage**

```
glPlotPie(asid, chnms = NULL, acids = NULL, aeids = NULL,  
          expos.time.ordr = NULL, stat = quote(modl_acc))
```

**Arguments**

asid	Assay source id
chnms	Character vector with list of chemical names
acids	Numeric vector with list of acids
auids	Character vector with list of assay endpoints IDs
expos.time.order	Character vector with sorted list of exposure times
stat	Statistic to plot (e.g. MEC:modl_acc or modl_acb, AC50:modl_ga)

**Details**

This function is useful to plot MEC or AC50 values

**Value**

None

**Examples**

```
## Create a pie plot of MEC values for all chemicals tested in the study
glPlotPie(asid=1L)
```

---

glPlotPieLogo

*plot package logo*

---

**Description**

This function plots the GladiaTOX logo.

**Usage**

```
glPlotPieLogo()
```

**Details**

This function is only used to plot the package logo.

**Value**

None

**Examples**

```
glPlotPieLogo()
```

---

glPlotPosCtrl	<i>Box plot for positive control check</i>
---------------	--

---

## Description

This function plots positive controls as well as vehicle and treatments normalized values

## Usage

```
glPlotPosCtrl(asid)
```

## Arguments

asid	Assay source id
------	-----------------

## Details

This function is useful to select plates to mask

## Value

A list of ggplot objects, one per assay X timepoint.

## Examples

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

## Create boxplot for all endpoints and chemicals tested. Useful to save
## plots in a pdf file.
pp <- glPlotPosCtrl(asid = 1L)
pp[[1]]

## Reset configuration
options(conf_store)
```

---

`glPlotPosCtrlMEC`*Box plot for positive control check*

---

**Description**

This function plots positive controls for study id asid as well as boxplot historical positive control MECs

**Usage**

```
glPlotPosCtrlMEC(asid, masked=NULL)
```

**Arguments**

<code>asid</code>	Assay source id
<code>masked</code>	Masking color

**Details**

This funtion is useful to select plates to mask

**Value**

A list of ggplot objects, one per assay X timepoint.

**Note**

PMI-specific

**Examples**

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

## Create boxplot for all endpoints and chemicals tested. Useful to save
## plots in a pdf file.
pp <- glPlotPosCtrlMEC(asid = 1L)
pp[[1]]

## Reset configuration
options(conf_store)
```



---

glPlotStat	<i>Box plot for Minimal Effective Concentrations (MEC) and AC50 plot</i>
------------	--

---

## Description

This function plots MEC values

## Usage

```
glPlotStat(asid, ref.chm = NULL, stat = quote(modl_acc))
```

## Arguments

asid	Assay source id
ref.chm	Chemical to adopt as reference
stat	Character vector of statistic to export

## Details

This function is useful to show the MEC trend over control chemical

## Value

A list of ggplot objects, one per assay X timepoint.

## Examples

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

## Create boxplot of MEC
## plots in a pdf file.
pp <- glPlotStat(asid = 1L)
pp[[1]]

## Reset configuration
options(conf_store)
```

---

glPlotToxInd	<i>Plot toxicological indicator values for all chemicals in input</i>
--------------	---

---

## Description

This function plots the toxicological indicator value for the assay source id in input.

## Usage

```
glPlotToxInd(asid, tp = NULL, stat = quote(modl_acc))
```

## Arguments

asid	assay source id
tp	Time point to report
stat	statistic to plot

## Details

This function is useful to plot toxicological indicator values. These values, for each chemical, represent an average impact of the chemical across the list of endpoints tested. The function transform the data to minus log scale. Hence the larger the indicator value, larger is the impact of the chemical.

## Value

None

## Examples

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

## Compute and plot toxicological severity index
glPlotToxInd(asid=1L)
```

gtoxAddModel

*Draw a gtox Model onto an existing plot***Description**

gtoxAddModel draws a line for one of the gtox Models (see [Models](#) for more information) onto an existing plot.

**Usage**

```
gtoxAddModel(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**

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

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

adj is intended to scale the models, so that models with different response units can be visualized on a single plot. The recommended value for adj is  $1/(3 \times \text{bmad})$  for level 4 data and  $1/\text{coff}$  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.

**Value**

None

**See Also**

[Models](#), [gtoxPlotFits](#)

**Examples**

```
## Create some dummy data to plot
logc <- 1:10
r1 <- sapply(logc, gtoxHillVal, ga = 5, tp = 50, gw = 0.5)
r2 <- log2(sapply(logc, gtoxHillVal, ga = 4, tp = 30, gw = 0.5))
p1 <- gtoxFit(logc = logc, resp = r1, bmad = 10)
p2 <- gtoxFit(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")
gtoxAddModel(pars = p1, modl = "hill")
points(r2 ~ logc)
gtoxAddModel(pars = p2, modl = "hill", lty = "dashed")

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

```

---

gtoxAICProb

---

*Calculate the AIC probabilities*


---

## Description

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

## Usage

```
gtoxAICProb(...)
```

## 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

[gtoxFit](#), [AIC](#) for more information about AIC values.

**Examples**

```
## Returns the probability for each model, given models with AIC values
## ranging from 80 to 100
gtoxAICProb(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
gtoxAICProb(m1, m2)
```

---

gtoxAppend	<i>Append rows to a table</i>
------------	-------------------------------

---

**Description**

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

**Usage**

```
gtoxAppend(dat, tbl, db)
```

**Arguments**

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

**Value**

None

**Note**

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

---

`gtoxCalcVmad`*Calculate and update the assay endpoint cutoff values*

---

### Description

`gtoxCalcVmad` takes the input `aid` values and uses them to calculate the assay endpoint cutoff based on the median absolute deviation of vehicle values across the given assay endpoints.

### Usage

```
gtoxCalcVmad(inputs, aid = NULL, notes = NULL)
```

### Arguments

<code>inputs</code>	integer, the <code>aid(s)</code> used to calculate the cutoff values
<code>aid</code>	integer, the <code>aid(s)</code> to be updated in the database
<code>notes</code>	character of length 1, (optional) comments/justification

### Details

If '`aid`' is `NULL`, the value will be returned with no changes made to the database.

Cutoffs are caluted as the median absolute value of the vehicle values across the assay endpoints given by '`inputs`'.

### Value

None

### Examples

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

## Prepare for analysis before QC + process data
gtoxCalcVmad(inputs = 10L)

## Reset configuration
options(conf_store)
```

---

gtoxCascade	<i>Do a cascading delete on gtox screening data</i>
-------------	---

---

**Description**

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

**Usage**

```
gtoxCascade(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 aeid(s).

**Value**

None

---

gtoxCode2CASN	<i>Convert chemical code to CAS Registry Number</i>
---------------	---

---

**Description**

gtoxCode2CASN takes a code and converts it CAS Registry Number.

**Usage**

```
gtoxCode2CASN(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

```
gtoxCode2CASN("C80057")
gtoxCode2CASN("C09812420") ## Invalid CASRN will give a warning
gtoxCode2CASN("CNOCAS0015") ## The underscore is reinserted for NOCAS codes
```

---

gtoxDelete	<i>Delete rows from gtox databases</i>
------------	--

---

## Description

gtoxDelete deletes rows from the given table and database.

## Usage

```
gtoxDelete(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

## Value

None

## See Also

[gtoxSendQuery](#)



---

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

---

## Description

`gtoxFit` 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

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

## Arguments

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

## Details

By default, `gtoxFit` will only attempt to fit concentration series when at least one median value is greater than  $3 \times \text{bmad}$ .

## Value

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

## See Also

[gtoxObjCnst](#), [gtoxObjHill](#), [gtoxObjGnls](#), [constrOptim](#)

## Examples

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

gtoxHillACXX

*Functions to solve the Hill model***Description**

These functions solve for Hill model parameters.

**Usage**

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

```
gtoxHillConc(val, tp, ga, gw, bt = 0)
```

```
gtoxHillVal(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**

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

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

gtoxHillConc computes the Hill model concentration for a given value.

**Value**

None

**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.
## gtoxHillVal calculates activity value given a concentration. gtoxHillVal
## will return the tp/2 when logc equals ga:
gtoxHillVal(logc = 1, tp = 50, ga = 1, gw = 1, bt = 0)

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

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

## gtoxHillACXX 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 <- gtoxHillVal(logc = 0.25, tp = 50, ga = 1, gw = 1, bt = 0)
c1 <- gtoxHillConc(val = value, tp = 50, ga = 1, gw = 1, bt = 0)
c2 <- gtoxHillACXX(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 gtoxHillACXX
```

---

gtoxImportThermoDB	<i>Import data from ThermoDB by study ID</i>
--------------------	--

---

## Description

This function accesses the ThermoDB webservises and imports data from ThermoDB to the gtox database.

## Usage

```
gtoxImportThermoDB(asid, verbose = TRUE, write = FALSE,
  store = "STORE", type = "mc",
  curlurl = "https://YOUR_THERMODB_SERVER_HOSTNAME/HTTPHCSCConnect")
```

## Arguments

asid	Integer, the assay study/source ID to import data for
verbose	Logical, should the output from the curl be displayed?
write	Logical, should the data be written to the database, or just returned?
store	Character, the name of the store on ThermoDB to query
type	Character, the data type: 'mc' or 'sc'
curlurl	URL of the webservice

## Value

Data table with content fetched from Thermo DB.

**Examples**

```
## Fetches data from ThermoDB to load in GladiaTOX DB prior processing
conf_store <- gtoxConfList()
gtoxConfDefault()

## Not run:
## Fetch data from ThermoDB
dat <- gtoxImportThermoDB(asid=1L)

## End(Not run)

## Reset configuration
options(conf_store)
```

---

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

---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

A string of field names for the given table.

**Examples**

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

---

gtoxLoadApid	<i>Load assay plate information</i>
--------------	-------------------------------------

---

**Description**

gtoxLoadApid queries the gtox database and returns the assay plate information for the given field and values.

**Usage**

```
gtoxLoadApid(fld = NULL, val = NULL)
```

**Arguments**

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'.

**Value**

A data.table with the assay plate information for the given parameters

**Examples**

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

## Prepare for analysis before QC + process data
gtoxLoadApid()

## Reset configuration
options(conf_store)
```

---

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

---

**Description**

gtoxLoadChem queries the gtox database and returns the chemical information for the given field and values.

## Usage

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

## Arguments

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

## Details

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

The functionality of the 'exact' parameter cannot be demonstrated within the SQLite environment. However, in the MariaDB 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 "mercury". More technically, setting 'exact' to FALSE passes the string in 'val' to an RLIKE statement within the MariaDB 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 <- gtoxConfList()  
gtoxConfDefault()  
  
## Passing no parameters gives all of the registered chemicals with their  
## sample IDs  
gtoxLoadChem()  
  
## Or the user can exclude spid and get a unique list of chemicals  
gtoxLoadChem(include.spid = FALSE)  
  
## Other examples:  
gtoxLoadChem(field = "chnm", val = "chromium")  
  
## Reset configuration  
options(conf_store)
```

---

gtoxLoadClib*Load chemical library information*

---

## Description

gtoxLoadClib queries the gtox databases and returns information about the chemical library.

## Usage

```
gtoxLoadClib(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 <- gtoxConfList()
gtoxConfDefault()

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

## Reset configuration
options(conf_store)
```

gtoxLoadData

*Load gtox data***Description**

gtoxLoadData queries the gtox databases and returns a data.table with data for the given level and data type.

**Usage**

```
gtoxLoadData(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 mutiple concentration data, or 'sc' for single concentration data. Multiple concentration data will be loaded into the 'mc' tables, whereas the single concentration will be loaded into the 'sc' tables.

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

Leaving fld NULL will return all data.

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

type	lvl	Queried tables
sc	0	sc0
sc	1	sc0, sc1
sc	agg	sc1, sc2_agg
sc	2	sc2
mc	0	mc0
mc	1	mc0, mc1
mc	2	mc0, mc1, mc2
mc	3	mc0, mc1, mc3
mc	agg	mc3, mc4_agg
mc	4	mc4
mc	5	mc4, mc5
mc	6	mc4, mc6



**Value**

A data.table containing data for the given fields.

**See Also**

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

**Examples**

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

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

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

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

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

## Reset configuration
options(conf_store)
```

---

gtoxLoadUnit

*Load response units for assay endpoints*


---

**Description**

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

**Usage**

```
gtoxLoadUnit(aeid)
```

**Arguments**

`aeid` Integer, assay endpoint ids

**Value**

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

**See Also**

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

---

gtoxLoadVehicle	<i>Load vehicle information</i>
-----------------	---------------------------------

---

**Description**

gtoxLoadVehicle queries the gtox database and returns the vehicle information for the given field and values.

**Usage**

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

**Arguments**

field	Character of length 1, the field to query on
val	Vector of values to subset on

**Value**

A data.table with the list of vehicles and vehicles ids.

**Examples**

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

## Prepare for analysis before QC + process data
gtoxLoadVehicle()

## Reset configuration
options(conf_store)
```

---

gtoxLoadVmad	<i>Load cutoff values for assay endpoints</i>
--------------	---

---

**Description**

gtoxLoadVmad queries the gtox databases and returns a data.table with the cutoff values for the given assay endpoint ids (aeid).

**Usage**

```
gtoxLoadVmad(aeid = NULL)
```

**Arguments**

aeid	Integer, assay endpoint ids
------	-----------------------------

**Value**

A data.table containing cutoff values for the given aeids.

**Examples**

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

## Prepare for analysis before QC + process data
gtoxLoadVmad()

## Reset configuration
options(conf_store)
```

---

gtoxLoadWaid	<i>Load well annotation information</i>
--------------	---

---

**Description**

gtoxLoadWaid queries the gtox database and returns the well annotation information for the given field and values.

**Usage**

```
gtoxLoadWaid(fld = NULL, val = NULL)
```

**Arguments**

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'.

**Value**

A data.table with the well annotation information for the given parameters

**Examples**

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

## Prepare for analysis before QC + process data
gtoxLoadWaid()

## Reset configuration
options(conf_store)
```

---

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

---

**Description**

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

**Usage**

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

**Arguments**

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

## Details

gtoxMakeAeidPlts provides a wrapper for [gtoxPlotFits](#), 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 `gtoxMakeAeidPlts`, but FALSE in `gtoxPlotFits`

## Value

None

## Examples

```
## Save Aeid plot in a pdf file
gtoxMakeAeidPlts(aeid = 10, lvl = 6, ordr.fitc = FALSE)
```

---

gtoxMthdAssign

*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, `gtoxMthdList` lists the available methods.

## Usage

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

```
gtoxMthdClear(lvl, id, mthd_id = NULL, type)
```

```
gtoxMthdList(lvl, type = "mc")
```

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

## Arguments

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

**Details**

`gtoxMthdLoad` loads the assigned methods for the given level and ID(s). Similarly, `gtoxMthdList` 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. `gtoxMthdAssign` assigns methods to the given ID(s), and `gtoxMthdClear` 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.

**Value**

None

**Examples**

```
## Not run:
## Assign level 2 methods (none for all acid values)
gtoxMthdAssign(lvl = 2L, id = 1L, mthd_id = 1, ordr = 1, type = "mc")

## Process data
gtoxRun(asid = 1L, slvl = 1, elvl = 6, mc.cores = 2)

## End(Not run)

## Not run:
## Clear level 2 methods
gtoxMthdClear(lvl = 2L, id = 1L, mthd_id = NULL, type = "mc")

## Assign level 2 methods (none for all acid values)
gtoxMthdAssign(lvl = 2L, id = 1L, mthd_id = 1, ordr = 1, type = "mc")

## Process data
gtoxRun(asid = 1L, slvl = 1, elvl = 6, mc.cores = 2)

## End(Not run)

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

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

## gtoxLoadMthd 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.
gtoxMthdLoad(lvl = 2, id = 1, type = "mc")

## Reset configuration
options(conf_store)
```

---

gtoxPlotErrBar	Create error bar plots
----------------	------------------------

---

## Description

gtoxPlotErrBar creates the error bar plots.

## Usage

```
gtoxPlotErrBar(c1, c2, aeid, ngrp = NULL)
```

## Arguments

c1	Integer of length 1, the chid value for the first chemical
c2	Integer of length 1, the chid value for the first chemical
aeid	Integer, the aeid value(s) to plot
ngrp	Integer, the number of "slots" to draw; overridden if the number of aeid values is greater than 'ngrp'

## Value

None

## Examples

```
## Plot error bar plot
gtoxPlotErrBar(c1=1, c2=3, aeid=17:18)
```

---

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

---

**Description**

gtoxPlotFitc makes a plot showing the level 5 fit categories.

**Usage**

```
gtoxPlotFitc(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

**Value**

None

**Note**

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

**Examples**

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

## Display the fit category tree.
gtoxPlotFitc()
```

---

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

---

**Description**

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

**Usage**

```
gtoxPlotFits(dat, agg, flg = NULL, ordr.fitc = FALSE, bline = "bmad")
```



**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>ordr.fitc</code>	Logical, should the fits be ordered by fit category?
<code>bline</code>	Character of length 1, the value used for drawing the baseline noise

**Details**

The data for 'dat', 'agg', and 'flg' should be loaded using the [gtoxLoadData](#) function with the appropriate 'lvl' parameter. See help page for gtoxLoadData 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.

Any values for 'bline' other than 'coff' will use 3\*bmad.

**Value**

None

**Examples**

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

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

aeid = 2
l5 <- gtoxLoadData(lvl = 5, fld = "aeid", val = aeid)
l4_agg <- gtoxLoadData(lvl = "agg", fld = "aeid", val = aeid)
l6 <- gtoxLoadData(lvl = 6, fld = "aeid", val = aeid)
## Not run:
pdf(file = "gtoxPlotFits.pdf", height = 6, width = 10, pointsize = 10)
gtoxPlotFits(dat = l5, agg = l4_agg, flg = l6)
graphics.off()

## End(Not run)

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

```
## Start by identifying some sample IDs to plot, then call gtoxPlotFits with
## a subset of the data. This browse function is admittedly clunky.
bpa <- gtoxLoadChem(field = "chnm", val = "chromium")[ , spid]
l5_sub <- l5[spid %in% bpa]

gtoxPlotFits(dat = l5_sub, agg = l4_agg[m4id %in% l5_sub$m4id])

## Reset configuration
options(conf_store)
```

---

gtoxPlotM4ID	<i>Plot fit summary plot by m4id</i>
--------------	--------------------------------------

---

## Description

gtoxPlotM4ID creates a summary plots for the given m4id(s) by loading the appropriate data from the gtox databases and sending it to [gtoxPlotFits](#)

## Usage

```
gtoxPlotM4ID(m4id, lvl = 4L, bline = "bmad")
```

## Arguments

m4id	Integer, m4id(s) to plot
lvl	Integer, the level of data to plot
bline	Character of length 1, the value used for drawing the baseline noise

## Details

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

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

Any values for 'bline' other than 'coff' will use 3\*bmad.

## Value

None

## See Also

[gtoxPlotFits](#), [gtoxMakeAeidPlts](#)

**Examples**

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

acnm <- "Cytotoxicity (TIER1)_Cytochrome C release_24h"
pltnm <- "S-000049119"
myaid <- gtoxLoadApid()[u_boxtrack == pltnm, aid]
myaid <- myaid[myaid%in%gtoxLoadAid(fld = "asid", val = 1L)$aid]
apid <- gtoxLoadApid()[u_boxtrack == pltnm & aid == myaid, apid]
acid <- gtoxLoadAcid(fld = c("aid", "acnm"), val = list(myaid, acnm))[, acid]

aeid = gtoxLoadAeid(fld = c("acid", "analysis_direction"),
  val = list(acid, "up"))[,aeid]
spid = gtoxLoadWaid(fld = c("apid", "wllt"),
  val = list(apid, "c"))[,unique(spid)]
m4id = gtoxLoadData(lvl = 4L, fld = c("spid", "aeid"),
  val = list(spid, aeid))[, m4id]

gtoxPlotM4ID(m4id = m4id, lvl = 6, bline = "coff") ## Create a level 4 plot
gtoxPlotM4ID(m4id = m4id, lvl = 5) ## Create a level 5 plot
gtoxPlotM4ID(m4id = m4id, lvl = 6) ## Create a level 6 plot

## Reset configuration
options(conf_store)
```

---

gtoxPlotPie	<i>Create piechart plots</i>
-------------	------------------------------

---

**Description**

gtoxPlotPie creates the piechart plots.

**Usage**

```
gtoxPlotPie(chid, mrks, aeid, col = NULL, lbl = NULL)
```

**Arguments**

chid	Integer of length 1, the chid value
mrks	Numeric, the values for concentration label rings
aeid	Integer, the aeid values to plot
col	Vector of colors
lbl	Vector with pie labels (optional)

**Value**

None

**Examples**

```
## Plot pie for chemical IDs 1 to 5 and multiple endpoints
gtoxPlotPie(chid=1:5, mrks=10^c(1:6), aeid=c(2:10))
```

---

gtoxPlotPieLgnd	<i>Create piechart plot legend</i>
-----------------	------------------------------------

---

**Description**

gtoxPlotPieLgnd creates the piechart plots.

**Usage**

```
gtoxPlotPieLgnd(aenm, ncol = 2, col = NULL, fit.labels = TRUE)
```

**Arguments**

aenm	Character, the assay endpoint names
ncol	Interger, the number of columns for the legend
col	Vector of colors
fit.labels	Boolean, if TRUE, scale the text to fit

**Value**

None

**Examples**

```
## Plot pie legend
gtoxPlotPieLgnd(aenm=c("Endpoint1", "Endpoint2"))
```

---

gtoxPlotPlate	<i>Plot plate heatmap</i>
---------------	---------------------------

---

## Description

gtoxPlotPlate generates a heatmap of assay plate data

## Usage

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

## Arguments

dat	data.table containing gtox 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.

## Value

None

## Note

For the optimal output size, use width = 12, height = 8, pointsize = 12, units = "in"

**Examples**

```
## Define assay component and extract assay component ID
acnm <- "Cytotoxicity (TIER1)_Cytochrome C release_24h"
acid <- gtoxLoadAcid(fld=c("asid", "acnm"), val=list(1L,acnm))[, acid]
## Extract assay plate ID corresponding to plate name S-000049119
apid <- gtoxLoadApid()[u_boxtrack == "S-000049119", apid]
## Load level 2 data (Raw data before normalization)
l2 <- gtoxLoadData(lvl = 2L, fld = "acid", val = acid)
gtoxPlotPlate(dat = l2, apid = apid, id = acid)
```

gtoxPlotWin

*Create winning curve plots***Description**

gtoxPlotWin creates best fit plot.

**Usage**

```
gtoxPlotWin(chid, aeid, bline = "bmad", collapse = TRUE)
```

**Arguments**

chid	Integer of length 1, the chid value
aeid	Integer, the aeid values to plot
bline	Character of length 1, the value used for drawing the baseline noise
collapse	Logical, collapse the data by spid when true

**Details**

When 'collapse' is TRUE the plotted points will be the mean of the values based on spid.

Any values for 'bline' other than 'coff' will use 3\*bmad.

**Value**

None

**Examples**

```
## Not run:
## Load chemical ID
chid <- gtoxLoadChem(field="chnm", val="acrylamide", include.spid=FALSE)$chid

## Load Assay endpoint ID
aeid <- gtoxLoadAeid(fld=c("asid","aenm"),
  val=list(1L, "GSH content_GSH content_4h_dn"), add.fld="asid")$aeid
```

```
## Plot winning model
gtoxPlotWin(chid = chid, aeid = aeid, bline="bmad", collapse=TRUE)

## End(Not run)
```

---

**gtoxPrepOtppt***Map assay/chemical ID values to annotation information*

---

## Description

gtoxPrepOtppt queries the chemical and assay information from the gtox database, and maps the annotation information to the given data.

## Usage

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

## Arguments

dat	data.table, output from <a href="#">gtoxLoadData</a>
ids	Character, (optional) a subset of ID fields to map

## Details

gtoxPrepOtppt 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 <- gtoxConfList()
gtoxConfDefault()

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

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

```
## Map chemical annotation only
d2 <- gtoxPrep0tpt(d1, ids = "spid")
"chnm" %in% names(d2) ## TRUE
"acnm" %in% names(d2) ## FALSE

## Map all annotations
d3 <- gtoxPrep0tpt(d1) ## Also works if function is given d2
"chnm" %in% names(d2) ## TRUE
"acnm" %in% names(d2) ## FALSE

## Reset configuration
options(conf_store)
```

---

gtoxQuery

*Wrappers for sending queries and fetching results*


---

## Description

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

## Usage

```
gtoxQuery(query, db = getOption("TCPL_DB"),
  drvr = getOption("TCPL_DVR"))

gtoxSendQuery(query, db = getOption("TCPL_DB"),
  drvr = getOption("TCPL_DVR"))
```

## Arguments

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

## Details

Currently, the gtox package only supports the "MariaDB" and "SQLite" database drivers.

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

## Value

None



## Examples

```
## Perform query
gtoxSendQuery(paste0("SELECT * FROM assay_source"))

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

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

## Reset configuration
options(conf_store)
```

---

**gtoxRegister***Functions for registering & updating annotation information*

---

## Description

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

## Usage

```
gtoxRegister(what, flds)

gtoxUpdate(what, id, flds)
```

## Arguments

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

## Details

These functions are used to populate the gtox 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 gtox 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
- spid (sample): spid, chid
- chid (chemical): chid, casn
- clib (chemical\_library): chid, clib
- \*vehicle (vehicle): vehicle\_name
- \*waid (assay\_plate\_well): apid, spid, rowi, coli, wllt, vhid, conc
- \*apid (assay\_plate): aid

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. \* indicate PMI-specific fields.

## Value

None

## Examples

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

## Load current ASID information
gtoxLoadAsid()

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

## Show the newly registered ASID
gtoxLoadAsid(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 gtoxUpdate
i1 <- gtoxLoadAsid()[asnm == "example_asid", asid]
gtoxUpdate(what = "asid",
           id = i1,
           flds = list(assay_source_desc = "example asid description"))
gtoxLoadAsid(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.
gtoxSendQuery(paste0("DELETE FROM assay_source WHERE asid = ", i1, ";"))

## Reset configuration
options(conf_store)
```

---

gtoxReport	<i>Generate a report</i>
------------	--------------------------

---

## Description

`gtoxReport` generates a report.

## Usage

```
gtoxReport(type, asid, params = NULL, odir = getwd(), report_author,  
  report_title = "Report", sumfile = NULL,  
  keep.intermediates = FALSE)
```

## Arguments

<code>type</code>	The type of report to generate
<code>asid</code>	The assay source/study ID
<code>params</code>	Named list containing report type-specific parameters, see details
<code>odir</code>	The output directory
<code>report_author</code>	The author for the report
<code>report_title</code>	The title for the report
<code>sumfile</code>	Path to a text file that inserts into the report
<code>keep.intermediates</code>	TRUE/FALSE, keep intermediate files when TRUE

## Details

'type' can have three values, "all," "compare," or "qc." Each report contains slightly different elements, but in general:

- "all" – summarizes the results for all or some compounds
  - "chids" – (optional) a vector of chid values to report, rather than all available compounds
- "compare" – compares the results for two compounds
  - "c1" – (required) the chid for the first compound to compare
  - "c2" – (required) the chid for the second compound to compare
- "qc" – summarizes low-level data for quality control purposes
  - "aids" – (optional) a vector of aid values to report, rather than all available assays

The required list elements vary depending on the type of report, and are described under the report descriptions above.

'sumfile' allows the user to inject a Tex file into the report. The file contents will be inserted into the Study Overview section, immediately after the autogenerated text. Technically, 'sumfile' is brewed, so 'sumfile' can make use of brew and Sweave syntax, and all data loaded for the report.

Value

None

Examples

```
## Generate full analysis report

## Not run:
## Generate report
gtoxReport(type = "all", asid = 1L, report_author = "author",
report_title = "Processing report")

## End(Not run)
```

---

gtoxRun	<i>Perform data processing</i>
---------	--------------------------------

---

Description

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

Usage

```
gtoxRun(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 outfile (optional)

## Details

The gtoxRun 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.

## Examples

```
## Process data for asid 1

## Process data
gtoxRun(asid = 1L, slvl = 1, elvl = 6, mc.cores = 2)
```

---

gtoxSetWllq

---

*Change the well quality for a vector of lvl 0 IDs*


---

## Description

gtoxSetWllq changes the well quality to either 100 or 0 for a given list of 'm0id' or 's0id' values. Changing the well quality initiates a delete cascade for the affected assay components.

## Usage

```
gtoxSetWllq(ids, wllq, type)
```

## Arguments

ids	Integer, the 'm0id' or 's0id' values to change
wllq	Integer of length 1, the new well quality value, 0 or 1
type	Character of length 1, the data type, "sc" or "mc"

## Value

TRUE if successful.

## Examples

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

## Set well quality to zero for specific lvl zero ids.
gtoxSetWllq(ids = 1633, wllq = 0, type = "mc")

## Reset configuration
options(conf_store)
```

---

gtoxSubsetChid	<i>Subset level 5 data to a single sample per chemical</i>
----------------	--

---

## Description

gtoxSubsetChid 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

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

## Arguments

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

## Details

gtoxSubsetChid is intended to work with level 5 data that has chemical and assay information mapped with [gtoxPrep0tpt](#).

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**[gtoxPrep0tpt](#)**Examples**

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

## Load the example level 5 data
d1 <- gtoxLoadData(lvl = 5, fld = "aeid", val = 2)
d1 <- gtoxPrep0tpt(d1)

## Subset to an example of a duplicated chid
d2 <- d1[chid == 10]
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.
gtoxSubsetChid(dat = d2, flag = FALSE)[ , list(m4id, modl_ga)]

## Reset configuration
options(conf_store)
```

gtoxWriteData

*Write screening data into the gtox databases***Description**

`gtoxWriteData` takes a `data.table` with screening data and writes the data into the given level table in the gtox databases.

**Usage**

```
gtoxWriteData(dat, lvl, type)
```

**Arguments**

<code>dat</code>	data.table, the screening data to load
<code>lvl</code>	Integer of length 1, the data processing level
<code>type</code>	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.

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.

**Value**

None

**See Also**

[gtoxCascade](#), [gtoxAppend](#)

**Examples**

```
## Not run:
## Load sample data
load(system.file("extdata", "data_for_vignette.rda", package="GladiaTOX"))

# Build assay table
assay <- buildAssayTab(plate, chnmap)

## Set study parameters
std.nm <- "SampleStudy" # study name
phs.nm <- "PhaseII" # study phase

## Load annotation in gtoxDB
loadAnnot(plate, assay, NULL)

## Get the created study ID
asid = gtoxLoadAsid(fld = c("asnm", "asph"), val = list(std.nm, phs.nm))$asid

## Prepare and load data
dat <- prepareDatForDB(asid, dat)
gtoxWriteData(dat[, list(acid, waid, wllq, rval)], lvl = 0, type = "mc")

## End(Not run)
```

---

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

**Value**

None

**See Also**

[MC6\\_Methods](#), [Method functions](#), [mc6](#)

---

is.odd

---

*Check for odd numbers*


---

**Description**

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

**Usage**

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

**Arguments**

x	An integer
---	------------

**Value**

TRUE for odd integers and FALSE for even integers.

**See Also**

Other gtox abbreviations: [lu](#), [lw](#), [sink\\_reset](#)

---

Load assay information

*Functions for loading assay information*


---

**Description**

These functions query the gtox 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**

```
gtoxLoadAcid(fld = NULL, val = NULL, add.fld = NULL)
gtoxLoadAeid(fld = NULL, val = NULL, add.fld = NULL)
gtoxLoadAid(fld = NULL, val = NULL, add.fld = NULL)
gtoxLoadAsid(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, `gtoxLoadAsid` 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 <- gtoxConfList()
gtoxConfDefault()

## The load assay functions can be used without any parameters to list the
## full list of registered assay elements:

## Assay source ID table
gtoxLoadAsid()
```

```

## Assay ID table
gtoxLoadAid()

## Assay component ID table
gtoxLoadAcid()

## Assay endpoint ID table
gtoxLoadAeid()

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

## Or, the user can look only at a subset:
gtoxLoadAeid(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.
gtoxListFlds("assay")
a1 <- gtoxLoadAeid(fld = "anm", val = "Apo Necro (casp37)_4h")
a2 <- gtoxLoadAeid(fld = "assay_name", val = "Apo Necro (casp37)_4h")
identical(a1, a2)

## Reset configuration
options(conf_store)

```

---

loadAnnot

Register the annotations provided by GUI

---

## Description

This function parses the output from the GUI and registers the appropriate data within the Gladia-TOX database.

## Usage

```
loadAnnot(plate, assay, outFile = "out.json")
```

## Arguments

plate	path to 'plate' JSON file produced by the GUI
assay	path to 'assay' JSON file produced by the GUI
outFile	character of length 1, name of the output file

## Details

If loading legacy data, 'outFile' should be set to NULL and no JSON file will be written.

**Value**

Logical value

**Examples**

```
## Not run:
## Load sample data
load(system.file("extdata", "data_for_vignette.rda", package="GladiaTOX"))

## Build assay table
assay <- buildAssayTab(plate, chnmap)

## Set study parameters
std.nm <- "SampleStudy" # study name
phs.nm <- "PhaseII" # study phase

## Delete previously loaded study data
asid = gttoxLoadAsid(fld=c("asnm", "asph"), val=list(std.nm, phs.nm))$asid
if(length(asid)>0){ deleteStudy(asid=asid) }

## Load annotation in gttoxDB
loadAnnot(plate, assay, NULL)

## End(Not run)
```

---

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

---

**Description**

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

**Usage**

```
lu(x)
```

**Arguments**

x                      A logical

**Value**

The unique of the TRUE values in x

**See Also**

[unique](#), [which](#)  
Other gttox abbreviations: [is.odd](#), [lw](#), [sink\\_reset](#)

**Examples**

```
lu(c(1, 1, 2, 3))
```

---

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

---

**Description**

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

**Usage**

```
lw(x)
```

**Arguments**

x                      A logical

**Value**

The length of the TRUE values in x

**See Also**

[length](#), [which](#)

Other gtox abbreviations: [is.odd](#), [lu](#), [sink\\_reset](#)

**Examples**

```
lw(c(TRUE, FALSE, TRUE))
```

---

mc1	<i>Perform level 1 multiple-concentration processing</i>
-----	--

---

**Description**

mc1 loads level 0 data from the gtox 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 [gtoxCascade](#). See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the [gtoxRun](#) 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 gtox 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 data processing functions: [mc2](#), [mc3](#), [mc4](#), [mc5](#), [mc6](#)

---

mc2

---

*Perform level 2 multiple-concentration processing*


---

**Description**

mc2 loads level 1 data from the gtox 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 [gtocascade](#). See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the [gtocrun](#) 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 gtox 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 data processing functions: [mc1](#), [mc3](#), [mc4](#), [mc5](#), [mc6](#)

---

MC2\_Methods

---

*List of level 2 multiple-concentration correction functions*


---

**Description**

mc2\_mthds returns a list of correction/transformation functions to be used during level 2 multiple-concentration processing.

**Usage**

```
mc2_mthds()
```

**Details**

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

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

**Value**

A list functions

**Available Methods**

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

**log2** Take the logarithm of cval with the base 2.

**log10** Take the logarithm of cval with the base 10.

**rmneg** Remove entries where cval is less than 0.

**rmzero** Remove entries where cval is 0.

**mult25** Multiply cval by 25.

**mult100** Multiply cval by 100.

- negshift** Shift cval by subtracting out the minimum of cval and adding 1, such that the new minimum of cval is 1.
- mult25** Multiply cval by 2.5.
- mult3** Multiply cval by 3.
- mult6** Multiply cval by 6.

**Note**

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

**See Also**

[mc2](#), [Method functions](#) to query what methods get applied to each acid

---

mc3	<i>Perform level 3 multiple-concentration processing</i>
-----	--

---

**Description**

mc3 loads level 2 data from the gtox 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 [gtocascade](#). See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the [gtocrun](#) 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 gtox 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 data processing functions: [mc1](#), [mc2](#), [mc4](#), [mc5](#), [mc6](#)

---

MC3\_Methods

---

*List of level 3 multiple-concentration normalization methods*


---

**Description**

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

**Usage**

```
mc3_mthds()
```

**Details**

The functions contained in the list returned by `mc3_mthds` take 'aids' (a numeric vector of aid 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 aid.

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.

#### **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.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](#), [gtoxMthdLoad](#) to query what methods get applied to each `aeid`

---

mc4

*Perform level 4 multiple-concentration processing*

---

### Description

`mc4` loads level 3 data from the `gtox` 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 [gtoxCascade](#). See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the [gtoxRun](#) 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>gtox</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

[gtoxFit](#), [Models](#)

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

---

mc5

*Perform level 5 multiple-concentration processing*

---

### Description

mc5 loads level 4 data from the gtox 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 [gtoxCascade](#). See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the [gtoxRun](#) wrapper function.

### Usage

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

### Arguments

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

### Details

Level 5 multiple-concentration hit-calling uses the fit parameters and the activity cutoff methods from mc5\_aeid and mc5\_methods to make an activity call and identify the winning model for each fit.

### Value

A boolean of length 1, indicating the success of the processing, or when 'wr' is FALSE, a list where the first element is a boolean 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 data processing functions: [mc1](#), [mc2](#), [mc3](#), [mc4](#), [mc6](#)

---

MC5\_Methods

---

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


---

**Description**

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

**Usage**

```
mc5_mthds()
```

**Value**

A list of functions

**Available Methods**

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

**bmad3** Add a cutoff value of  $3 \times \text{bmad}$ .

**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  $\log_{10}(1.2)$ .

**bmad5** Add a cutoff value of  $5 \times \text{bmad}$ .

**bmad6** Add a cutoff value of  $6 \times \text{bmad}$ .

**bmad10** Add a cutoff value of  $10 \times \text{bmad}$ .

**log2\_2** Add a cutoff value of  $\log_2(2)$ .

**log10\_2** Add a cutoff value of  $\log_{10}(2)$ .

**neglog2\_0.88** Add a cutoff value of  $-1 \times \log_2(0.88)$ .

**vmad3** Add a cutoff value of  $3 \times \text{vmad}$ .

**vmad5** Add a cutoff value of  $5 \times \text{vmad}$ .

**vmad10** Add a cutoff value of  $10 \times \text{vmad}$ .

**See Also**

[mc5](#), [Method functions](#) to query what methods get applied to each acid

---

`mc6`*Perform level 6 multiple-concentration processing*

---

## Description

`mc6` loads level 5 data from the `gtox` 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 [`gtoxCascade`](#). See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the [`gtoxRun`](#) wrapper function.

## Usage

```
mc6(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>gtox</code> 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 data processing functions: [mc1](#), [mc2](#), [mc3](#), [mc4](#), [mc5](#)

**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."

**row.dev.up** The `row.dev.up` flag looks at the individual point data, searching for row effects across an apid. To get flagged the point has to be greater than 3 standard deviations above the mean response for the plate, and the row mean must be greater than 3 standard deviations above the row means for the plate.

**row.dev.dn** The `row.dev.dn` flag is identical to the `row.dev.up` flag, but identifies points falling in rows with decreased signals.

**col.dev.up** The `col.dev.up` flag is identical to the `row.dev.up` flag, but identifies points falling in columns with increased signals.

**col.dev.dn** The `col.dev.dn` flag is identical to the `row.dev.up` flag, but identifies points falling in columns with decreased signals.

**plate.flare** The `plate.flare` flag looks at the individual point data, searching for overly active regions across an apid. Intended for use in fluorometric assays that are read by a plate-reader that measures the plate as a whole, rather than measuring individual wells. For each well the flare value is calculated as a weighted mean a 5 well by 5 well box centered on the well where the weight given to each well in the box is the euclidian distance from the center well. The flag then identifies points with flare values greater than 3 standard deviations above the mean flare values for the plate.

**plate.interlace** The `plate.interlace` flag is specific to one experimental design that plates chemicals from a 386 well chemical plate to a 1536 well assay plate. The flag looks for any chemical-plate affects, by looking for an increased signal in the wells originating from the same chemical plate.

**rep.mismatch** The `rep.mismatch` flag is still in development and is not suggested for use at this time.

- pintool** Deprecated. The pintool flag uses a complicated algorithm to look for signal potentially caused by residual in the pintool used to deliver the chemical to assay plates in some experimental designs. The gnls.lowconc is a faster and simpler way to identify where this problem may be driving the activity or hit-call.
- 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 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. Intended for use with biochemical assays where one might expect at least a 50% change in real responses.

### See Also

[mc6](#), [Method functions](#) to query what methods get applied to each acid

### 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 `gtoxFit` function.



**Usage**

```
gtoxObjCnst(p, resp)

gtoxObjGnls(p, lconc, resp)

gtoxObjHill(p, lconc, resp)
```

**Arguments**

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

**Details**

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

`gtoxObjCnst` calculates the likelihood for a constant model at 0. The only parameter passed to `gtoxObjCnst` 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)**

`gtoxObjGnls` 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 `gtoxObjGnls` 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)

gtoxObjHill calculates the likelihood for a 3 parameter Hill model with the bottom equal to 0. The parameters passed to gtoxObjHill 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.

### Examples

```
## Load level 3 data for an assay endpoint ID
dat <- gtoxLoadData(lvl=3L, type="mc", fld="aeid", val=3L)

## Compute fitting log-likelihood
gtoxObjCnst(1, dat$resp)

## Load level 3 data for an assay endpoint ID
dat <- gtoxLoadData(lvl=3L, type="mc", fld="aeid", val=2L)

## Compute fitting log-likelihood
gtoxObjGnls(p=c(rep(0.5,5),1e-3), lconc=dat$logc, resp=dat$resp)

## Load level 3 data for an assay endpoint ID
dat <- gtoxLoadData(lvl=3L, type="mc", fld="aeid", val=3L)

## Compute fitting log-likelihood
gtoxObjHill(c(rep(0,3), 1e-3), dat$logc, dat$resp)
```

---

```
prepareDatForDB
```

---

```
Assign default processing methods
```

---

### Description

This function is a wrapper to ease the creation of the dataframe containing data and metadata to be loaded in the database

### Usage

```
prepareDatForDB(asid, dat)
```

**Arguments**

asid	Integer, the asid value(s) to assign the default methods to
dat	Data.table containing metadata and data to load in DB

**Details**

This function formats a dat table to be loaded in DB

**Value**

Data table with data and metadata to store in database

**Examples**

```
## Not run:
## Load sample data
load(system.file("extdata", "data_for_vignette.rda", package="GladiaTOX"))

# Build assay table
assay <- buildAssayTab(plate, chnmap)

## Set study parameters
std.nm <- "SampleStudy" # study name
phs.nm <- "PhaseII" # study phase

## Delete previously loaded study data
asid = gtoxLoadAsid(fld=c("asnm", "asph"), val=list(std.nm, phs.nm))$asid
if(length(asid)>0){ deleteStudy(asid=asid) }

## Load annotation in gtoxDB
loadAnnot(plate, assay, NULL)

## Get the created study ID
asid = gtoxLoadAsid(fld = c("asnm", "asph"), val = list(std.nm, phs.nm))$asid

## Prepare and load data
dat <- prepareDatForDB(asid, dat)

## End(Not run)
```

---

registerMthd

---

*Add a new analysis method*


---

**Description**

registerMthd registers a new analysis method to the gtox databases.

Usage

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

Value

None

---

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

---

Description

sc1 loads level 0 data from the gtox 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 [gtxCascade](#). See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the [gtxRun](#) 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 gtox 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 data processing functions: [sc2](#)

---

SC1\_Methods

---

*List of level 1 single-concentration normalization functions*


---

**Description**

sc1\_mthds returns a list of functions to be used during level 1 single-concentration processing.

**Usage**

```
sc1_mthds()
```

**Details**

The functions contained in the list returned by sc1\_mthds return a list of expressions to be executed in the sc2 (not exported) function environment. The functions are described here for reference purposes, The sc1\_mthds function is not exported, nor is it intended for use.

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

**Value**

A list functions

**Available Methods**

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

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

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

**bval Methods:**

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

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

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

**pval Methods:**

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

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

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

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

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

**pval.zero** Define pval as 0.

**resp Methods:**

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

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

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

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

**resp.multneg1** Multiply resp by -1.

**none** Do no normalization; make resp equal to rval.

**Note**

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

**See Also**

[sc1](#), [Method functions](#) to query what methods get applied to each acid

**Description**

sc2 loads level 1 data from the gtox 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 [gtocascade](#). See details for more information.

The individual processing functions are no longer exported, as it is typically more convenient and suggested to use the [gtocrun](#) 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 gtox 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 data processing functions: [sc1](#)

---

SC2_Methods	<i>List of level 2 single-concentration hit-call functions</i>
-------------	--

---

**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."

**bmadd3** Add a cutoff value of  $3 \times \text{bmadd}$ .

**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  $\log_{10}(1.2)$ .

**bmadd5** Add a cutoff value of  $5 \times \text{bmadd}$ .

**bmadd6** Add a cutoff value of  $6 \times \text{bmadd}$ .

**bmadd10** Add a cutoff value of  $10 \times \text{bmadd}$ .

**pc30orbmadd3** Add a cutoff value of either 30 or  $3 \times \text{bmadd}$ , whichever is less.

### Note

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

### See Also

[sc2](#), [Method functions](#) to query what methods get applied to each acid

---

sink\_reset

*Reset all sinks*

---

### Description

sink\_reset resets all sinks and returns all output to the console.

### Usage

```
sink_reset()
```

### Details

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

### Value

None

### See Also

[sink](#), [sink.number](#)

Other gtox abbreviations: [is.odd](#), [lu](#), [lw](#)



# Index

## \* data processing functions

gtoxRun, [52](#)

## \* gtox abbreviations

is.odd, [57](#)

lu, [60](#)

lw, [61](#)

sink\_reset, [80](#)

## \* internal

blinShift, [4](#)

flareFunc, [10](#)

gtoxAppend, [21](#)

gtoxCascade, [23](#)

gtoxDelete, [24](#)

gtoxLoadUnit, [33](#)

interlaceFunc, [56](#)

is.odd, [57](#)

mc1, [61](#)

mc2, [62](#)

MC2\_Methods, [63](#)

mc3, [64](#)

MC3\_Methods, [65](#)

mc4, [67](#)

mc5, [68](#)

MC5\_Methods, [69](#)

mc6, [70](#)

MC6\_Methods, [71](#)

registerMthd, [75](#)

sc1, [76](#)

SC1\_Methods, [77](#)

sc2, [78](#)

SC2\_Methods, [79](#)

sink\_reset, [80](#)

## \* multiple-concentration data

mc1, [61](#)

mc2, [62](#)

mc3, [64](#)

mc4, [67](#)

mc5, [68](#)

mc6, [70](#)

## \* processing functions

mc1, [61](#)

mc2, [62](#)

mc3, [64](#)

mc4, [67](#)

mc5, [68](#)

mc6, [70](#)

sc1, [76](#)

sc2, [78](#)

## \* single-concentration data

sc1, [76](#)

sc2, [78](#)

AIC, [20](#)

assignDefaultMthds, [3](#)

blinShift, [4](#)

buildAssayTab, [5](#)

Configure functions, [6](#)

constrOptim, [25](#), [72](#)

data.table, [33](#), [34](#)

deleteStudy, [8](#)

exportResultForToxpiGUI, [9](#)

exportResultTable, [10](#)

flareFunc, [10](#)

glCheckInput, [11](#)

glComputeToxInd, [12](#)

glLoadInput, [13](#)

glPlotPie, [13](#)

glPlotPieLogo, [14](#)

glPlotPosCtrl, [15](#)

glPlotPosCtrlMEC, [16](#)

glPlotStat, [17](#)

glPlotToxInd, [18](#)

gtoxAddModel, [19](#)

gtoxAICProb, [20](#)

- gtoxAppend, [21](#), [56](#)
- gtoxCalcVmad, [22](#)
- gtoxCascade, [23](#), [56](#), [61](#), [62](#), [64](#), [67](#), [68](#), [70](#), [76](#), [78](#)
- gtoxCode2CASN, [23](#)
- gtoxConf (Configure functions), [6](#)
- gtoxConfDefault (Configure functions), [6](#)
- gtoxConfList (Configure functions), [6](#)
- gtoxConfLoad (Configure functions), [6](#)
- gtoxConfReset (Configure functions), [6](#)
- gtoxConfSave (Configure functions), [6](#)
- gtoxDelete, [24](#)
- gtoxFit, [20](#), [25](#), [68](#), [72](#)
- gtoxHillACXX, [26](#)
- gtoxHillConc (gtoxHillACXX), [26](#)
- gtoxHillVal (gtoxHillACXX), [26](#)
- gtoxImportThermoDB, [27](#)
- gtoxListFlds, [28](#)
- gtoxLoadAcid (Load assay information), [58](#)
- gtoxLoadAeid (Load assay information), [58](#)
- gtoxLoadAid (Load assay information), [58](#)
- gtoxLoadApid, [29](#)
- gtoxLoadAsid (Load assay information), [58](#)
- gtoxLoadChem, [29](#)
- gtoxLoadClib, [31](#), [36](#)
- gtoxLoadData, [32](#), [41](#), [47](#)
- gtoxLoadUnit, [33](#)
- gtoxLoadVehicle, [34](#)
- gtoxLoadVmad, [35](#)
- gtoxLoadWaid, [35](#)
- gtoxMakeAeidPlts, [36](#), [42](#)
- gtoxMthdAssign, [37](#)
- gtoxMthdClear (gtoxMthdAssign), [37](#)
- gtoxMthdList (gtoxMthdAssign), [37](#)
- gtoxMthdLoad, [67](#)
- gtoxMthdLoad (gtoxMthdAssign), [37](#)
- gtoxObjCnst, [25](#)
- gtoxObjCnst (Models), [72](#)
- gtoxObjGnls, [25](#)
- gtoxObjGnls (Models), [72](#)
- gtoxObjHill, [25](#)
- gtoxObjHill (Models), [72](#)
- gtoxPlotErrBar, [39](#)
- gtoxPlotFitc, [40](#)
- gtoxPlotFits, [19](#), [37](#), [40](#), [42](#)
- gtoxPlotM4ID, [42](#)
- gtoxPlotPie, [43](#)
- gtoxPlotPieLgnd, [44](#)
- gtoxPlotPlate, [45](#)
- gtoxPlotWin, [46](#)
- gtoxPrepOtpt, [47](#), [54](#), [55](#)
- gtoxQuery, [33](#), [34](#), [48](#)
- gtoxRegister, [49](#)
- gtoxReport, [51](#)
- gtoxRun, [52](#), [61](#), [62](#), [64](#), [67](#), [68](#), [70](#), [76](#), [78](#)
- gtoxSendQuery, [24](#)
- gtoxSendQuery (gtoxQuery), [48](#)
- gtoxSetWllq, [53](#)
- gtoxSubsetChid, [54](#)
- gtoxUpdate (gtoxRegister), [49](#)
- gtoxWriteData, [55](#)
- Hill model utilites (gtoxHillACXX), [26](#)
- interlaceFunc, [56](#)
- is.odd, [57](#), [60](#), [61](#), [80](#)
- length, [61](#)
- Load assay information, [58](#)
- loadAnnot, [59](#)
- lu, [57](#), [60](#), [61](#), [80](#)
- lw, [57](#), [60](#), [61](#), [80](#)
- mc1, [61](#), [63](#), [65](#), [68–70](#)
- mc2, [62](#), [62](#), [64](#), [65](#), [68–70](#)
- MC2\_Methods, [63](#), [63](#)
- mc2\_mthds (MC2\_Methods), [63](#)
- mc3, [5](#), [62](#), [63](#), [64](#), [67–70](#)
- MC3\_Methods, [65](#), [65](#)
- mc3\_mthds, [5](#)
- mc3\_mthds (MC3\_Methods), [65](#)
- mc4, [62](#), [63](#), [65](#), [67](#), [69](#), [70](#)
- mc5, [62](#), [63](#), [65](#), [68](#), [68](#), [69](#), [70](#)
- MC5\_Methods, [69](#), [69](#)
- mc5\_mthds (MC5\_Methods), [69](#)
- mc6, [11](#), [57](#), [62](#), [63](#), [65](#), [68](#), [69](#), [70](#), [72](#)
- MC6\_Methods, [11](#), [57](#), [70](#), [71](#)
- mc6\_mthds (MC6\_Methods), [71](#)
- mclapply, [53](#)
- Method functions (gtoxMthdAssign), [37](#)
- Models, [19](#), [68](#), [72](#)
- prepareDatForDB, [74](#)
- Query functions (gtoxQuery), [48](#)

Register/update annotation  
    (gtoxRegister), [49](#)  
registerMthd, [75](#)  
  
sc1, [76](#), [78](#), [79](#)  
SC1\_Methods, [77](#), [77](#)  
sc1\_mthds (SC1\_Methods), [77](#)  
sc2, [77](#), [78](#), [80](#)  
SC2\_Methods, [79](#), [79](#)  
sc2\_mthds (SC2\_Methods), [79](#)  
sink, [80](#)  
sink.number, [80](#)  
sink\_reset, [57](#), [60](#), [61](#), [80](#)  
Startup, [6](#)  
Sys.getenv, [6](#)  
  
unique, [60](#)  
  
which, [60](#), [61](#)