# The ToxCast™ Analysis Pipeline: An R Package for Processing and Modeling Chemical Screening Data

Version 1.4.3

Dayne L. Filer, Parth Kothiya, Woodrow R. Setzer,
Richard S. Judson, Matthew T. Martin

May 20, 2018

# Contents

# Introduction

## Overview

The `tcpl` package was developed to process high-throughput and high-content screening data generated by the U.S. Environmental Protection Agency (EPA) ToxCast™ program.[1] ToxCast is screening thousands of chemicals with hundreds of assays coming from numerous and diverse biochemical and cell-based technology platforms. The diverse data, received in heterogeneous formats from numerous vendors, are transformed to a standard computable format and loaded into the `tcpl` database by vendor-specific R scripts. Once data is loaded into the database, ToxCast utilizes the generalized processing functions provided in this package to process, normalize, model, qualify, flag, inspect, and visualize the data. While developed primarily for ToxCast, we have attempted to make the `tcpl` package generally applicable to chemical-screening community.

The `tcpl` package includes processing functionality for two screening paradigms: (1) single-concentration screening and (2) multiple-concentration screening. Single-concentration screening consists of testing chemicals at one concentration, often for the purpose of identifying potentially active chemicals to test in the multiple-concentration format. Multiple-concentration screening consists of testing chemicals across a concentration range, such that the modeled activity can give an estimate of potency, efficacy, etc.

Prior to the pipeline processing provided in this package, all the data must go through pre-processing (level 0). Level 0 pre-processing utilizes dataset-specific R scripts to process the heterogeneous data into a uniform format and to load the uniform data into the `tcpl` database. Level 0 pre-processing is outside the scope of this package, but can be done for virtually any high-throughput or high-content chemical screening effort, provided the resulting data includes the minimum required information.

In addition to storing the data, the `tcpl` database stores every processing/analysis decision at the assay component or assay endpoint level to facilitate transparency and reproducibility. For the illustrative purposes of this vignette we have included a SQLite version of the `tcpl` database containing a small subset of data from the ToxCast program. Because of differences in database capabilities, not all functionality of the package will work with the SQLite version. To best utilize the package the user should work with a MySQL database and the `RMySQL` package. The package includes a SQL file to initialize the MySQL database on the user's server of choice. Additionally, the MySQL version of the ToxCast database containing all the publicly available ToxCast data is available for download at: `<http://epa.gov/ncct/toxcast/data.html>`.

[1] `<http://www.epa.gov/ncct/toxcast/>`

## Package Settings

First, it is highly recommended for users to utilize the `data.table` package. The `tcpl` package utilizes the `data.table` package for all data frame-like objects.

R Input

```
> library(data.table)
> library(tcpl)
> ## Store the path the tcpl directory for loading data
> pkg_dir <- system.file(package = "tcpl")
```

Every time the package is loaded in a new R session, a message similar to the following will print showing the default package settings:

R Output

```
tcpl (v1.0) loaded with the following settings:
  TCPL_DB:    /usr/local/lib64/R/library/tcpl/sql/xmpl.sqlite
  TCPL_USER:  NA
  TCPL_HOST:  NA
  TCPL_DRVR:  SQLite
Default settings stored in TCPL.conf. See ?tcplConf for
more information.
```

The package consists of five settings: (1) `$TCPL_DB` points to the `tcpl` database (either the SQLite file, as in the given example above, or the name of the MySQL database), (2) `$TCPL_USER` stores the username for accessing the database, (3) `$TCPL_PASS` stores the password for accessing the database, (4) `$TCPL_HOST` points to the MySQL server host, and (5) `$TCPL_DRVR` indicates which database driver to use (either "MySQL" or "SQLite").

Refer to `?tcplConf` for more information. At any time users can check the settings using `tcplConfList()`. An example of database settings would be as follows:

R Input

```
> tcplConf(drvr = "MySQL",
           user = "root",
           pass = "",
           host = "localhost",
           db   = "toxcastdb")
```

Note, `tcplSetOpts` will only make changes to the parameters given. The package is always loaded with the settings stored in the TCPL.config file located within the package directory. The user can edit the file, such that the package loads with the desired settings, rather than having to call the `tcplSetOpts` function every time. The TCPL.config file has to be edited whenever the package is updated or re-installed.

## Assay Structure

The definition of an "assay" is, for the purposes of this package, broken into:

**assay_source** – the vendor/origination of the data

**assay** – the procedure to generate the component data

**assay_component** – the raw data readout(s)

**assay_component_endpoint** – the normalized component data

Each assay element is represented by a separate table in the `tcpl` database. In general, we refer to an "assay_component_endpoint" as an "assay endpoint." As we move down the hierarchy, each additional layer has a one-to-many relationship with the previous layer. For example, an assay component can have multiple assay endpoints, but an assay endpoint can derive only from a single assay component.

All processing occurs by assay component or assay endpoint, depending on the processing type (single-concentration or multiple-concentration) and level. No data are stored at the assay or assay source level. The "assay" and "assay_source" tables store annotations to help in the processing and downstream understanding/analysis of the data. For more information about the assay annotations and the ToxCast assays please refer to `<http://www.epa.gov/ncct/toxcast/>`.

Throughout the package the levels of assay hierarchy are defined and referenced by their primary keys (IDs) in the `tcpl` database: *asid* (assay source ID), *aid* (assay ID), *acid* (assay component ID), and *aeid* (assay endpoint ID). In addition, the package abbreviates the fields for the assay hierarchy names. The abbreviations mirror the abbreviations for the IDs with "nm" in place of "id" in the abbreviations, e.g. assay_component_name is abbreviated *acnm*.

## Register and Upload New Data

This section explains how to register and upload new data into the `tcpl` database using a small subset of ToxCast data showing changes intracellular cortisol hormone. The subset of data comes from an assay measuring steroidogenesis through cellular levels of mutliple steroid hormones.

The `tcpl` package provides three functions for adding new data: (1) `tcplRegister` to register a new assay or chemical ID, (2) `tcplUpdate` to change or add additional information for existing assay or chemical IDs, and (3) `tcplWriteLvl0` for loading data. Before writing any data to the `tcpl` database, the user has to register the assay and chemical information.

The first step in registering new assays is to register the assay source. As discussed in the previous section, the package refers to the levels of the assay hierarchy by their ID names, e.g. *asid* for assay source.The following code shows how to register an assay source, then ensure the assay source was properly registered.

R Input

```
> ## Add a new assay source, call it CTox,
> ## that produced the data
> tcplRegister(what = "asid", flds = list(asnm = "CTox"))
```

R Output

```
[1] TRUE
```

R Input

```
> tcplLoadAsid()
```

R Output

```
   asid asnm
1:    1 CTox
```

The `tcplRegister` function takes the abbreviation for *assay_source_name*, but the function will also take the unabbreviated form. The same is true of the `tcplLoadA-` functions, which load the information for the assay annotations stored in the database. The next steps show how to register, in order, an assay, assay component, and assay endpoints.

R Input

```
> tcplRegister(what = "aid",
               flds = list(asid = 1,
                           anm = "Steroidogenesis",
                           assay_footprint = "96 well"))
```

R Output

```
[1] TRUE
```

When registering an assay (*aid*), the user must give an *asid* to map the assay to the correct assay source. Registering an assay, in addition to an assay_name (*anm*) and *asid*, requires *assay_footprint*. The *assay_footprint* field is used in the assay plate visualization functions (discussed later) to define the appropriate plate size. The *assay_footprint* field can take most string values, but only the numeric value will be extracted, e.g. the text string "hello 384" would indicate to draw a 384-well microtitier plate. Values containing multiple numeric values in *assay_footprint* may cause errors in plotting plate diagrams.

With the assay registered, the next step is to register an assay component. The example data presented here only contains data for one of the many steroids measured and only requires one assay component, but at this step the user could add multiple assay components to the "Steroidogenesis" assay.

R Input

```
> tcplRegister(what = "acid",
               flds = list(aid = 1, acnm = "CTox_CORT"))
```

R Output

```
[1] TRUE
```

R Input

```
> tcplRegister(what = "aeid",
               flds = list(acid = c(1, 1),
                           aenm = c("CTox_CORT_up",
                                    "CTox_CORT_dn"),
                           normalized_data_type =
                           rep("log2_fold_induction", 2),
                           export_ready = c(1, 1),
                           burst_assay = c(0, 0),
                           fit_all = c(0, 0)))
```

R Output

```
[1] TRUE
```

In the example above two assay endpoints were assigned to the assay component. Multiple endpoints allow for different normalization approaches of the data, in this case to detect activity in both the positive and negative directions (up and down). Notice registering an assay endpoint

also requires the *normalized_data_type* field. The *normalized_data_type* field gives some default values for plotting. Currently the package supports three *normalized_data_type* values: (1) "percent_activity," (2) "log2_fold_induction," and (3) "log10_fold_induction." Any other values will be treated as "percent_activity."

The other three additional fields when registering an assay endpoint do not have to be explicitly defined when working in the MySQL environment and will default to the values given above. All three fields represent Boolean values (1 or 0, 1 being `TRUE`). The *export_ready* field indicates (1) the data is done and ready for export or (0) still in progress. The *burst_assay* field is specific to multiple-concentration processing and indicates (1) the assay endpoint is included in the burst distribution calculation or (0) not (Appendix C). The *fit_all* field is specific to multiple-concentration processing and indicates (1) the package should try to fit every concentration series, or (0) only attempt to fit concentration series that show evidence of activity (page 30).

The final piece of assay information needed is the assay component source name (abbreviated *acsn*), stored in the "assay_component_map" table. The assay component source name is intended to simplify level 0 pre-processing by defining unique character strings (concatenating information if necessary) from the source files that identify the specific assay components. The unique character strings (*acsn*) get mapped to *acid*. An example of how to register a new *acsn* will be given later in this section.

With the minimal assay information registered, the next step is to register the necessary chemical and sample information. The "chdat.csv" file included in the package contains the sample and chemical information for the data that will be loaded. The following shows an example of how to load chemical information. Similar to the order in registering assay information, the user must first register chemicals, then register samples that map to chemical.

R Input

```
> ch <- fread(file.path(pkg_dir, "sql", "chdat.csv"))
> head(ch)
```

R Output

```
        spid       casn
1: 01140000A 26172-55-4
2: 01140002A   109-43-3
3: 01140004A   486-56-6
4: 01140006A  2058-94-8
5: 01140008A   732-11-6
6: 01140010A    89-83-8
                                      chnm
1: 5-Chloro-2-methyl-3(2H)-isothiazolone
2:                  Dibutyl decanedioate
3:                              Cotinine
4:              Perfluoroundecanoic acid
5:                               Phosmet
6:                                Thymol
```

R Input

```
> ## Register the unique chemicals
> tcplRegister(what = "chid",
               flds = ch[ ,
                          unique(.SD),
                          .SDcols = c("casn", "chnm")])
```

R Output

```
[1] TRUE
```

The "chdat.csv" file contains a map of sample to chemical information, but chemical and sample information have to be registered separately because a chemical could potentially have multiple samples. Registering chemicals only takes a chemical CAS registry number (*casn*) and name (*chnm*). In the above example only the unique chemicals were loaded. The *casn* and *chnm* fields have unique constraints; trying to register multiple chemicals with the same name or CAS registry number is not possible and will result in an error. With the chemicals loaded the samples can be registered by mapping the sample ID (*spid*) to chemical ID. Note, the user needs to load the chemical information to get the chemical IDs then merge the new chemical IDs with the sample IDs from the original file by chemical name or CASRN.

R Input

```
> cmap <- tcplLoadChem()
> tcplRegister(what = "spid",
               flds = merge(ch[ , list(spid, casn)],
                            cmap[ , list(casn, chid)],
                            by = "casn")[ , list(spid, chid)])
```

R Output

```
[1] TRUE
```

Optionally, the user user can subdivide the chemcial IDs into different groups or libraries. For illustration, the chemical IDs will be arbitrarily divided into two chemical libraries, with the even numbered chemical IDs in group 1 and the odd numbered chemicals IDs in group 2.

R Input

```
> grp1 <- cmap[chid %% 2 == 0, unique(chid)]
> grp2 <- cmap[chid %% 2 == 1, unique(chid)]
> tcplRegister(what = "clib",
               flds = list(clib = "group_1", chid = grp1))
```

R Output

```
[1] TRUE
```

R Input

```
> tcplRegister(what = "clib",
               flds = list(clib = "group_2", chid = grp2))
```

R Output

```
[1] TRUE
```

Chemical IDs can belong to more than one library, and will be listed as seperate entries when loading chemical library information.

R Input

```
> tcplRegister(what = "clib",
               flds = list(clib = "other", chid = 1:2))
```

R Output

```
[1] TRUE
```

R Input

```
> tcplLoadClib(field = "chid", val = 1:2)
```

R Output

```
   chid    clib
1:    1 group_2
2:    1   other
3:    2 group_1
4:    2   other
```

After registering the chemical and assay information the data can be loaded into the `tcpl` database. The package includes two files from the ToxCast program, "scdat.csv" and "mcdat.csv," with a subset of single- and multiple-concentration data, respectively. The single- and multiple-concentration processing require the same level 0 fields; more information about level 0 pre-processing in Appendix B.

R Input

```
> scdat <- fread(file.path(pkg_dir, "sql", "scdat.csv"))
> mcdat <- fread(file.path(pkg_dir, "sql", "mcdat.csv"))
> c(unique(scdat$acsn), unique(mcdat$acsn))
```

R Output

```
[1] "cort" "cort"
```

As discussed above, the final step before loading data is mapping the assay component source name (*acsn*) to the correct *acid*. An assay component can have multiple *acsn* values, but an *acsn* must be unique to one assay component. Assay components can have multiple *acsn* values to minimize the amount of data manipulation required (and therefore potential errors) during the level 0 pre-processing if assay source files change or are inconsistent. The example data presented here only has one *acsn* value, "cort."

R Input

```
> tcplRegister(what = "acsn",
               flds = list(acid = 1, acsn = "cort"))
```

R Output

```
[1] TRUE
```

The data are now ready to be loaded with the `tcplWriteLvl0` function.

R Input

```
> tcplWriteLvl0(dat = scdat, type = "sc")
> tcplWriteLvl0(dat = mcdat, type = "mc")
```

The `type` argument is used throughout the package to distinguish the type of data/processing: "sc" indicates single-concentration; "mc" indicates multiple-concentration. The `tcplLoadData` function can be used to load the data from the database.

R Input

```
> tcplLoadData(lvl = 0, fld = "acid", val = 1, type = "sc")
```

R Output

```
      s0id      spid acid               apid rowi coli wllt
   1:    1 01140000A    1  TP0001059.Plate.8    4    6    t
   2:    2 01140000A    1  TP0001059.Plate.8    5    6    t
   3:    3 01140002A    1 TP0001061.Plate.14    6    9    t
   4:    4 01140002A    1 TP0001061.Plate.14    7    9    t
   5:    5 01140004A    1  TP0001059.Plate.5    4    3    t
  ---
4892: 4892  TX209150    1 TP0001061.Plate.13    5    6    t
4893: 4893  TX210870    1 TP0001061.Plate.14    4    6    t
4894: 4894  TX210870    1 TP0001061.Plate.14    5    6    t
4895: 4895  TX212325    1  TP0000885.Plate.1    2    6    t
4896: 4896  TX212325    1  TP0000885.Plate.1    3    6    t
      wllq  conc      rval                          srcf
   1:    1 100.1 16.130000  TP0001059 Plate 8_CeeTox.csv
   2:    1 100.1 17.270000  TP0001059 Plate 8_CeeTox.csv
   3:    1 100.1 25.870000 TP0001061 Plate 14_CeeTox.csv
   4:    1 100.1 24.160000 TP0001061 Plate 14_CeeTox.csv
   5:    1 100.0  7.670000  TP0001059 Plate 5_CeeTox.csv
  ---
4892:    1  10.0 18.620000 TP0001061 Plate 13_CeeTox.csv
4893:    1 100.0 28.370000 TP0001061 Plate 14_CeeTox.csv
4894:    1 100.0 28.440000 TP0001061 Plate 14_CeeTox.csv
4895:    1  10.0  7.961641       TP0000885 Plate 1A.xlsx
4896:    1  10.0  8.753819       TP0000885 Plate 1A.xlsx
```

Notice in the loaded data the `acsn` is replaced by the correct *acid* and the *s0id* field is added. The "s#" fields, and corresponding "m#" fields in the multiple-concentration data, are the primary keys for each level of data. These primary keys link the various levels of data. All of the keys are auto-generated and will change anytime data are reloaded or processed. Note, the primary keys only change for the levels affected, e.g. if the user reprocesses level 1, the level 0 keys will remain the same.

# Data Processing and the tcplRun Function

This section is intended help the user understand the general aspects of how the data is processed before diving into the specifics of each processing level for both screening paradigms. The details of the two screening paradigms are provided in later sections.

All processing in the `tcpl` package occurs at the assay component or assay endpoint level. There is no capability within either screening paradigm to do any processing which combines data from multiple assay components or assay endpoints. Any combining of data must occur before or after the pipeline processing. For example, a ratio of two values could be processed through the pipeline if the user calculated the ratio during the level 0 pre-processing and uploaded a single "component."

Once data are uploaded in the database, data processing occurs through the `tcplRun` function for both single- and multiple-concentration screening. The `tcplRun` function can either take a single ID (*acid* or *aeid*, depending on the processing type and level) or an *asid*. If given an *asid* the `tcplRun` function will attempt to process all corresponding components/endpoints. When processing by *acid* or *aeid*, the user must know which ID to give for each level (Table 1).

The processing is sequential, and every level of processing requires successful processing at the antecedent level. Any processing changes will cause a "delete cascade," removing any subsequent data affected by the processing change to ensure complete data fidelity at any given time. For example, processing level 3 data will cause the data from levels 4 through 6 to be deleted for the corresponding IDs. Changing any method assignments will also trigger a delete cascade for any corresponding data (more on method assignments below).

The user must give a start and end level when using the `tcplRun` function. If processing more than one assay component or endpoint, the function will not stop if one component or endpoint fails. If a component or endpoint fails while processing multiple levels, the function will not attempt to processes the failed component/endpoint in subsequent levels. When finished processing, the `tcplRun` function returns a list indicating the processing success of each id. For each level processed the list will contain two elements: (1) "l#" a named Boolean vector where `TRUE` indicates successful processing, and (2) "l#_failed" containing the names of any ids that failed processing where "#" is the processing level.

The processing functions print messages to the console indicating the four steps of the processing. First, data for the given assay component ID are loaded, the data are processed, data for the same ID in subsequent levels are deleted, then the processed data is written to the database. The 'outfile' parameter in the `tcplRun` function gives the user the option of printing all of the output text to a file.

The `tcplRun` function will attempt to use multiple processors on Unix-based systems (does not include Windows). Depending on the system environment, or if the user is running into memory constraints, the user may wish to use less processing power and can do so by setting the 'mc.cores' parameter in the `tcplRun` function.

Table 1: Processing checklist

| Type | Level | Input ID | Method ID |
|---|---|---|---|
| SC | Lvl 1 | `acid` | `aeid` |
| SC | Lvl 2 | `aeid` | `aeid` |
| MC | Lvl 1 | `acid` | `N/A` |
| MC | Lvl 2 | `acid` | `acid` |
| MC | Lvl 3 | `acid` | `aeid` |
| MC | Lvl 4 | `aeid` | `N/A` |
| MC | Lvl 5 | `aeid` | `aeid` |
| MC | Lvl 6 | `aeid` | `aeid` |

The Input ID column indicates the ID used for each processing step; Method ID indicates the ID used for assigning methods for data processing, when necessary. SC = single-concentration; MC = multiple-concentration.

The processing requirements vary by screening paradigm and level. Later sections will cover the details, but in general, many of the processing steps require specific methods to accommodate different experimental designs or data processing approaches.

Notice from Table 1 that level 1 single-concentration processing (SC1) requires an *acid* input (Table 1), but the methods are assigned by *aeid*. The same is true for MC3 processing. SC1 and MC3 are the normalization steps and convert *acid* to *aeid*. (Only MC2 has methods assigned by *acid*.) The normalization process is discussed in the following section.

To promote reproducibility, all method assignments must occur through the database. Methods cannot be passed to either the `tcplRun` function or the low-level processing functions called by `tcplRun`.

In general, method data are stored in the "_methods" and "_id" tables that correspond to the data-storing tables. For example, the "sc1" table is accompanied by the "sc1_methods" table which stores the available methods for SC1, and the "sc1_aeid" table which stores the method assignments and execution order.

The `tcpl` package provides three functions for easily modifying and loading the method assignments for the given assay components or endpoints: (1) `tcplMthdAssign` allows the user to assign methods, (2) `tcplMthdClear` clears method assignments, and (3) `tcplMthdLoad` queries the `tcpl` database and returns the method assignments. The package also includes the `tcplMthdList` function that queries the `tcpl` database and returns the list of available methods.

The following code blocks will give some examples of how to use the method-

related functions.

R Input

```
> ## For illustrative purposes, assign level 2 MC methods to
> ## ACIDs 98, 99. First check for available methods.
> mthds <- tcplMthdList(lvl = 2, type = "mc")
> mthds[1:2]
```

R Output

```
   mc2_mthd_id mc2_mthd                    desc
1:           1     none apply no level 2 method
2:           2     log2       log2 all raw data
```

R Input

```
> ## Assign some methods to ACID 97, 98 & 99
> tcplMthdAssign(lvl = 2,
                 id = 97:99,
                 mthd_id = c(3, 4, 2),
                 ordr = 1:3,
                 type = "mc")
```

R Output

```
Completed delete cascade for 0 ids (0.04 secs)
```

R Input

```
> tcplMthdLoad(lvl = 2, id = 97:99, type = "mc")
```

R Output

```
   acid   mthd mthd_id ordr
1:   97  rmneg       3    1
2:   97 rmzero       4    2
3:   97   log2       2    3
4:   98  rmneg       3    1
5:   98 rmzero       4    2
6:   98   log2       2    3
7:   99  rmneg       3    1
8:   99 rmzero       4    2
9:   99   log2       2    3
```

R Input

```
> ## Methods can be cleared one at a time for the given id(s)
> tcplMthdClear(lvl = 2, id = 99, mthd_id = 2, type = "mc")
```

R Output

```
Completed delete cascade for 0 ids (0.06 secs)
```

R Input

```
> tcplMthdLoad(lvl = 2, id = 99, type = "mc")
```

R Output

```
   acid   mthd mthd_id ordr
1:   99  rmneg       3    1
2:   99 rmzero       4    2
```

R Input

```
> ## Or all methods can be cleared for the given id(s)
> tcplMthdClear(lvl = 2, id = 97:98, type = "mc")
```

R Output

```
Completed delete cascade for 0 ids (0.04 secs)
```

R Input

```
> tcplMthdLoad(lvl = 2, id = 97:98, type = "mc")
```

R Output

```
Empty data.table (0 rows) of 4 cols: acid,mthd,mthd_id,ordr
```

# Data Normalization

Data normalization occurs in both single- and multiple-concentration processing at levels 1 and 3, respectively. While the two paradigms use different methods, the normalization approach is the same for both single- and multiple-concentration processing. Data normalization does not have to occur within the package, and normalized data can be loaded into the database at level 0. However, **data must be zero-centered and will only be fit in the positive direction**.

The `tcpl` package supports fold-change and a percent of control approaches to normalization. All data must be zero-centered so all fold-change data must be log-transformed. Normalizing to a control requires three normalization methods: (1) one to define the baseline value, (2) one to define the control value, and (3) one to calculate percent of control ("resp.pc"). Normalizing to fold-change also requires three methods: (1) one to define the baseline value, (2) one to calculate the fold-change, and (3) one to log-transform the fold-change values. Methods defining a baseline value (*bval*) have the "bval" prefix, methods defining the control value (*pval*) have the "pval" prefix, and methods that calculate or modify the final response value have the "resp" prefix. For example, "resp.log2" does a log-transformation of the response value using a base value of 2. The formluae for calculating the percent of control and fold-change response values are listed in equations 1 and 2, respectively.

The percent of control and fold-change values, respectively:

$$resp = \frac{cval - bval}{pval - bval}100 \tag{1}$$

$$resp = cval/bval \tag{2}$$

Order matters when assigning normalization methods. The *bval*, and *pval* if normalizing as a percent of control, need to be calculated prior to calculating the response value. Table 2 shows some possible normalization schemes.

Table 2: Example normalization method assignments.

| | | | |
|---|---|---|---|
| Fold-Change | 1. bval.apid.nwlls.med | 1. bval.apid.lowconc.med | 1. none |
| | 2. resp.fc | 2. resp.fc | 2. resp.log10 |
| | 3. resp.log2 | 3. resp.log2 | 3. resp.blineshift.50.spid |
| | 4. resp.mult.neg1 | 4. | 4. |
| % Control | 1. bval.apid.lowconc.med | 1. bval.spid.lowconc.med | 1. none |
| | 2. pval.apid.pwlls.med | 2. pval.apid.mwlls.med | 2. resp.multneg1 |
| | 3. resp.pc | 3. resp.pc | 3. |
| | 4. resp.multneg1 | 4. | 4. |

If the data does not require any normalization the "none" method must be assigned for normalization. The "none" method simply copies the input data to the response field. Without assigning "none" the response field will not get generated and the processing will not complete.

To reiterate, the package only models response in the positive direction. Therefore, signal in the negative direction must transformed to the positive direction during normalization. Negative direction data are inverted by multiplying the final response values by $-1$ (see the "resp.mult.neg‘" methods in Table 2).

In addition to the required normalization methods, the user can add additional methods to transform the normalized values. For example, the third fold-change example in Table 2 includes "resp.blineshift.50.spid," which corrects for baseline deviations by *spid*. A complete list of available methods, by processing type and level, can be listed with `tcplMthdList`. More information is available in the package documentation, and can be found by running `??tcpl::Methods`.

As discussed in the Assay Structure section (page 5), an assay component can have more than one assay endpoint. Creating multiple endpoints for one component enables multiple normalization approaches. Multiple normalization approaches may become necessary when the assay component detects signal in both positive and negative directions.

# Single-concentration Screening

This section will cover the `tcpl` process for handling single-concentration data[2]. The goal of single-concentration processing is to identify potentially active compounds from a broad screen at a single concentration. After the data is loaded into the `tcpl` database, the single-concentration processing consists of 2 levels (Table 3).

Table 3: Summary of the `tcpl` single-concentration pipeline

| | Description |
|---|---|
| Lvl 0 | Pre-processing: Vendor/dataset-specific pre-processing to organize heterogeneous raw data to the uniform format for processing by the `tcpl` package[†] |
| Lvl 1 | Normalize: Apply assay endpoint-specific normalization listed in the "sc1_aeid" table to the raw data to define response |
| Lvl 2 | Activity Call: Collapse replicates by median response, define the response cutoff based on methods in the "sc2_aeid" table, and determine activity |

[†]Level 0 pre-processing is outside the scope of this package

## Level 1

Level 1 processing converts the assay component to assay endpoint(s) and defines the normalized-response value field (*resp*); logarithm-concentration field (*logc*); and optionally, the baseline value (*bval*) and positive control value (*pval*) fields. The purpose of level 1 is to normalize the raw values to either the percentage of a control or to fold-change from baseline. The normalization process is discussed in greater detail in the Data Normalization section (page 18).

Before beginning the normalization process, all wells with well quality (*wllq*) equal to 0 are removed.

The first step in beginning the processing is to identify which assay endpoints stem from the assay component(s) being processed.

R Input

```
> tcplLoadAeid(fld = "acid", val = 1)
```

[2]This section assumes a working knowledge of the concepts covered in the Data Processing and Data Normalization sections (pages 14 and 18, respectively).

R Output

```
   acid aeid         aenm
1:    1    1 CTox_CORT_up
2:    1    2 CTox_CORT_dn
```

With the corresponding endpoints identified, the appropriate methods can be assigned.

R Input

```
> tcplMthdAssign(lvl = 1,
                 id = 1:2,
                 mthd_id = c(1, 11, 13),
                 ordr = 1:3,
                 type = "sc")
```

R Output

```
Completed delete cascade for 2 ids (0.01 secs)
```

R Input

```
> tcplMthdAssign(lvl = 1,
                 id = 2,
                 mthd_id = 16,
                 ordr = 4,
                 type = "sc")
```

R Output

```
Completed delete cascade for 1 ids (0.01 secs)
```

Above, methods 1, 11, and 13 were assigned for both endpoints. The method assignments instruct the processing to: (1) calculate *bval* for each assay plate ID by taking the median of all data where the well type equals "n;" (2) calculate a fold-change over *bval*; (3) log-transform the fold-change values with base 2. The second method assignment (only for AEID 2) indicates to multiply all response values by $-1$.

For a complete list of normalization methods see `tcplMthdList(lvl = 1, type = "sc")` or `?SC1_Methods`. With the assay endpoints and normalization methods defined, the data are ready for level 1 processing.

R Input

```
> ## Do level 1 processing for acid 1
> sc1_res <- tcplRun(id = 1, slvl = 1, elvl = 1, type = "sc")
```

R Output

```
Writing level 1 complete. (0.07 secs)


Total processing time: 0.01 mins
```

**Notice that level 1 processing takes an assay component ID, not an assay endpoint ID, as the input ID.** As mentioned in previously, the user must assign normalization methods by assay endpoint, then do the processing by assay component. The level 1 processing will attempt to process all endpoints in the database for a given component. If one endpoint fails for any reason (e.g., does not have appropriate methods assigned), the processing for the entire component fails.

## Level 2

Level 2 processing defines the baseline median absolute deviation (*bmad*), collapses any replicates by sample ID, and determines the activity.

Before the data are collapsed by sample ID, the *bmad* is calculated as the median absolute deviation of all wells with well type equal to "t." The calculation to define *bmad* is done once across the entire assay endpoint. **If additional data is added to the database for an assay component, the** *bmad* **values for all associated assay endpoints will change.** Note, this *bmad* definition is different from the *bmad* definition used for multiple-concentration screening.

To collapse the data by sample ID, the median response value is calculated at each concentration. The data are then further collapsed by taking the maximum of those median values (*max_med*).

Once the data are collapsed, such that each assay endpoint-sample pair only has one value, the activity is determined. For a sample to get an active hit-call, the *max_med* must be greater than an efficacy cutoff. The efficacy cutoff is determined by the level 2 methods. The efficacy cutoff value (*coff*) is defined as the maximum of all values given by the assigned level 2 methods. Failing to assign a level 2 method will result in every sample being called active. For a complete list of level 5 methods see `tcplMthdList(lvl = 2, type = "sc")` or `?SC2_Methods`.

R Input

```
> ## Assign a cutoff value of log2(1.2)
> tcplMthdAssign(lvl = 2,
                 id = 1:2,
                 mthd_id = 3,
                 type = "sc")
```

R Output

```
Completed delete cascade for 2 ids (0 secs)
```

For the example data the cutoff value is $log_2(1.2)$. If the maximum median value (*max_med*) is greater than or equal to the efficacy cutoff (*coff*), the sample ID is considered active and the hit-call (*hitc*) is set to 1.

With the methods assigned, the level 2 processing can be completed.

R Input

```
> ## Do level 1 processing for acid 1
> sc2_res <- tcplRun(id = 1:2, slvl = 2, elvl = 2, type = "sc")
```

R Output

```
 processing time: 0.01 mins
```

# Multiple-concentration Screening

This section will cover the `tcpl` process for handling multiple-concentration data[3]. The goal of multiple-concentration processing is to estimate the activity, potency, efficacy, and other parameters for sample-assay pairs. After the data is loaded into the `tcpl` database, the multiple-concentration processing consists of six levels (Table 4).

Table 4: Summary of the `tcpl` multiple-concentration pipeline

| | Description |
|---|---|
| Lvl 0 | Pre-processing: Vendor/dataset-specific pre-processing to organize heterogeneous raw data to the uniform format for processing by the `tcpl` package[†] |
| Lvl 1 | Index: Define the replicate and concentration indices to facilitate all subsequent processing |
| Lvl 2 | Transform: Apply assay component-specific transformations listed in the "mc2_acid" table to the raw data to define the corrected data |
| Lvl 3 | Normalize: Apply assay endpoint-specific normalization listed in the "mc3_aeid" table to the corrected data to define response |
| Lvl 4 | Fit: Model the concentration-response data utilizing three objective functions: (1) constant, (2) hill, and (3) gain-loss |
| Lvl 5 | Model Selection/Acitivty Call: Select the winning model, define the response cutoff based on methods in the "mc5_aeid" table, and determine activity |
| Lvl 6 | Flag: Flag potential false positive and false negative findings based on methods in the "mc6_aeid" table |

[†]Level 0 pre-processing is outside the scope of this package

## Level 1

Level 1 processing defines the replicate and concentration index fields to facilitate downstream processing. Because of cost, availability, physicochemical, and technical constraints screening-level efforts utilize numerous experimental designs and test compound (sample) stock concentrations. The resulting data may contain inconsistent numbers of concentrations, concentration values, and technical replicates. To enable quick and uniform processing, level 1 processing

[3]This section assumes a working knowledge of the concepts covered in the Data Processing and Data Normalization sections (pages 14 and 18, respectively).

explicitly defines concentration and replicate indices, giving integer values $1 \ldots N$ to increasing concentrations and technical replicates, where 1 represents the lowest concentration or first technical replicate.

To assign replicate and concentration indices we assume one of two experimental designs. The first design assumes samples are plated in multiple concentrations on each assay plate, such that the concentration series all falls on a single assay plate. The second design assumes samples are plated in a single concentration on each assay plate, such that the concentration series falls across many assay plates.

For both experimental designs, data are ordered by source file (*srcf*), assay plate ID (*apid*), column index (*coli*), row index (*rowi*), sample ID (*spid*), and concentration (*conc*). Concentration is rounded to three significant figures to correct for potential rounding errors. After ordering the data we create a temporary replicate ID, identifying an individual concentration series. For test compounds in experimental designs with the concentration series on a single plate and all control compounds, the temporary replicate ID consists of the sample ID, well type (*wllt*), source file, assay plate ID, and concentration. The temporary replicate ID for test compounds in experimental designs with concentration series that span multiple assay plates is defined similarly, but does not include assay plate ID.

Once the data are ordered, and the temporary replicate ID is defined, the data are scanned from top to bottom and increment the replicate index (*repi*) every time a replicate ID is duplicated. Then, for each replicate, the concentration index (*cndx*) is defined by ranking the unique concentrations, with the lowest concentration starting at 1.

The following demonstrates how to carry out the level 1 processing and look at the resulting data:

R Input

```
> ## Do level 1 processing for acid 1
> mc1_res <- tcplRun(id = 1, slvl = 1, elvl = 1, type = "mc")
```

R Output

```
Writing level 1 complete. (0.09 secs)


Total processing time: 0.01 mins
```

With the processing complete, the resulting level 1 data can be loaded to check the processing:

R Input

```
> ## Load the level 1 data and look at the cndx and repi values
> m1dat <- tcplLoadData(lvl = 1,
                        fld = "acid",
                        val = 1,
                        type = "mc")
> m1dat <- tcplPrepOtpt(m1dat)
> setkeyv(m1dat, c("repi", "cndx"))
> m1dat[chnm == "3-Phenylphenol", list(chnm, conc, cndx, repi)]
```

R Output

```
              chnm   conc cndx repi
 1: 3-Phenylphenol  0.082    1    1
 2: 3-Phenylphenol  0.247    2    1
 3: 3-Phenylphenol  0.741    3    1
 4: 3-Phenylphenol  2.222    4    1
 5: 3-Phenylphenol  6.667    5    1
 6: 3-Phenylphenol 20.000    6    1
 7: 3-Phenylphenol  0.082    1    2
 8: 3-Phenylphenol  0.247    2    2
 9: 3-Phenylphenol  0.741    3    2
10: 3-Phenylphenol  2.222    4    2
11: 3-Phenylphenol  6.667    5    2
12: 3-Phenylphenol 20.000    6    2
```

3-phenylphenol contains two replicates, each with six distinct concentrations. The package also contains a tool for visualizing the data at the assay plate level. In Figure 1 we see the results of `tcplPlotPlate`. The `tcplPlotPlate` function can be used to visualize the data at levels 1 to 3. The row and column indices are printed along the edge of the plate, with the values in each well represented by color. While the plate does not give sample ID information, the letter/number codes in the wells indicate the well type and concentration index, respectively. The plate display also shows the wells with poor quality (as defined by the well quality, *wllq*, field at level 0) with an "X." Plotting plates in subsequent levels wells with poor quality will appear empty. The title of the plate display lists the assay component/assay endpoint and the assay plate ID (*apid*).

## Level 2

Level 2 processing removes data where the well quality (*wllq*) equals 0 and defines the corrected value (*cval*) field. Level 2 processing allows for any transformation of the raw values at the assay component level. Examples of transformation methods could range from basic logarithm transformations, to

R Input

```
> tcplPlotPlate(dat = m1dat, apid = "09Apr2014.Plate.17")
```

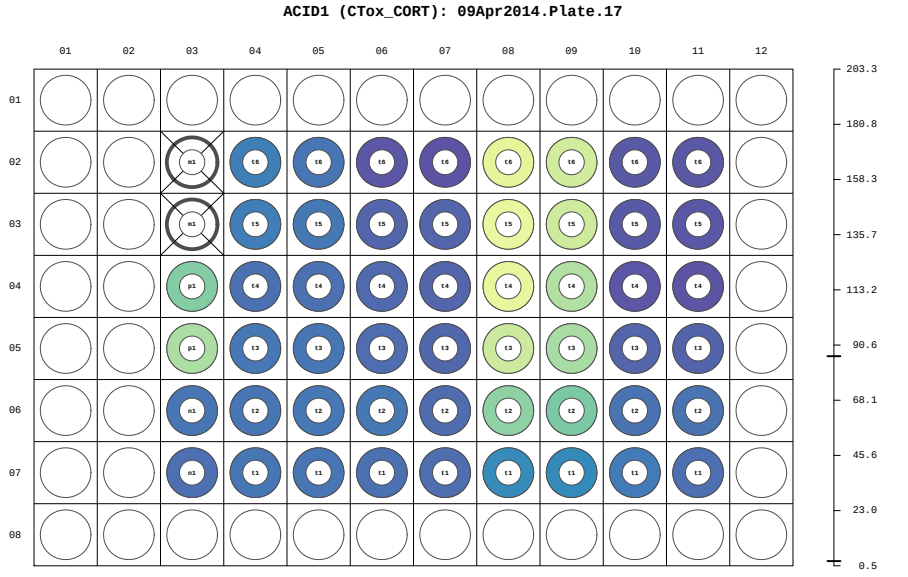



Figure 1: An assay plate diagram. The color indicates the raw values according to the key on the right. The bold lines on the key show the distribution of values for the plate on the scale of values across the entire assay. The text inside each well shows the well type and concentration index. For example, "t4" indicates a test compound at the fourth concentration. The wells with an "X" have a well quality of 0.

complex spacial noise reduction algorithms. Currently the `tcpl` package only consists of basic transformations, but could be expanded in future releases. Level 2 processing does not include normalization methods; normalization should occur during level 3 processing.

For the example data used in this vignette, no transformations are necessary at level 2. To not apply any transformation methods, assign the "none" method:

R Input

```
> tcplMthdAssign(lvl = 2,
                 id = 1,
                 mthd_id = 1,
                 ordr = 1,
                 type = "mc")
```

R Output

```
Completed delete cascade for 2 ids (0.03 secs)
```

Every assay component needs at least one transformation method assigned to complete level 2 processing. With the method assigned, the processing can be completed.

R Input

```
> ## Do level 2 processing for acid 1
> mc2_res <- tcplRun(id = 1, slvl = 2, elvl = 2, type = "mc")
```

R Output

```
Writing level 2 complete. (0.09 secs)


Total processing time: 0 mins
```

For the complete list of level 2 transformation methods currently available, see `tcplMthdList(lvl = 2, type = "mc")` or `?MC2_Methods` for more detail. The coding methodology used to implement the methods is beyond the scope of this vignette, but, in brief, the method names in the database correspond to a function name in the list of functions returned by `mc2_mthds()` (the `mc2_mthds` function is not exported, and not intended for use by the user). Each of the functions in the list given by `mc2_mthds()` only return expression objects that processing function called by `tcplRun` executes in the local function environment to avoid making additional copies of the data in memory. We encourage suggestions for new methods.

## Level 3

Level 3 processing converts the assay component to assay endpoint(s) and defines the normalized-response value field (*resp*); logarithm-concentration field (*logc*); and optionally, the baseline value (*bval*) and positive control value (*pval*) fields. The purpose of level 3 processing is to normalize the corrected values to either

the percentage of a control or to fold-change from baseline. The normalization process is discussed in greater detail in the Data Normalization section (page 18). The processing aspect of level 3 is almost completely analogous to level 2, except the user has to be careful about using assay component versus assay endpoint.

The user first needs to check which assay endpoints stem from the the assay component queued for processing.

R Input

```
> ## Look at the assay endpoints for acid 1
> tcplLoadAeid(fld = "acid", val = 1)
```

R Output

```
   acid aeid         aenm
1:    1    1 CTox_CORT_up
2:    1    2 CTox_CORT_dn
```

With the corresponding assay endpoints listed, the normalization methods can be assigned.

R Input

```
> tcplMthdAssign(lvl = 3,
                 id = 1:2,
                 mthd_id = c(17, 9, 7),
                 ordr = 1:3,
                 type = "mc")
```

R Output

```
Completed delete cascade for 2 ids (0.01 secs)
```

R Input

```
> tcplMthdAssign(lvl = 3,
                 id = 2,
                 mthd_id = 6,
                 ordr = 4,
                 type = "mc")
```

R Output

```
Completed delete cascade for 1 ids (0.01 secs)
```

Above, methods 17, 9, and 7 were assigned for both endpoints. The method assignments instruct the processing to: (1) calculate *bval* for each assay plate ID by taking the median of all data where the well type equals “n” or the well type

equals "t" and the concentration index is 1 or 2; (2) calculate a fold-change over *bval*; (3) log-transform the fold-change values with base 2. The second method assignment (only for AEID 2) tells the processing to multiply all response values by −1.

For a complete list of normalization methods see `tcplMthdList(lvl = 3, type = "mc")` or `?MC3_Methods`. With the assay endpoints and normalization methods defined, the data are ready for level 3 processing.

R Input

```
> ## Do level 3 processing for acid 1
> mc3_res <- tcplRun(id = 1, slvl = 3, elvl = 3, type = "mc")
```

R Output

```
3 complete. (0.1 secs)


Total processing time: 0.02 mins
```

**Notice that level 3 processing takes an assay component ID, not an assay endpoint ID, as the input ID.** As mentioned in previous sections, the user must assign normalization methods by assay endpoint, then do the processing by assay component. The level 3 processing will attempt to process all endpoints in the database for a given component. If one endpoint fails for any reason (e.g., does not have appropriate methods assigned), the processing for the entire component fails.

## Level 4

Level 4 processing splits the data into concentration series by sample and assay endpoint, then models the activity of each concentration series. Activity is modeled only in the positive direction. More information on readouts with both directions is available in the previous section.

The first step in level 4 processing is to remove the well types with only one concentration. To establish the noise-band for the assay endpoint, the baseline median absolute deviation (*bmad*) is calculated as the median absolute deviation of the response values for test compounds where the concentration index equals 1 or 2. The calculation to define *bmad* is done once across the entire assay endpoint. **If additional data is added to the database for an assay component, the *bmad* values for all associated assay endpoints will change.** Note, this *bmad* definition is different from the *bmad* definition used for single-concentration screening.

Before the model parameters are estimated, a set of summary values are calculated for each concentration series: the minimum and maximum response; minimum and maximum log concentration; the number of concentrations,

points, and replicates; the maximum mean and median with the concentration at which they occur; and the number of medians greater than $3bmad$. When referring to the concentration series the "mean" and "median" values are defined as the mean or median of the response values at every concentration. In other words, the maximum median is the maximum of all median values across the concentration series.

Concentration series must have at least four concentrations to enter the fitting algorithm. By default, concentration series must additionally have at least one median value greater than $3bmad$ to enter the fitting algorithm. The median value above $3bmad$ requirement can be ignored by setting *fit_all* to 1 in the assay endpoint annotation.

All models draw from the Student's t-distribution with four degrees of freedom. The wider tails in the t-distribution diminish the influence of outlier values, and produce more robust estimates than do the more commonly used normal distribution. The robust fitting removes the need for any outlier elimination before fitting. The fitting algorithm utilizes maximum likelihood estimates parameters for three models as defined below in equations 3 through 16.

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

$$z_i = \frac{y_i - \mu_i}{\exp(\sigma)}, \tag{3}$$

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

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

where $n$ is the number of observations.

The first model fit in the fitting algorithm is a constant model at 0, abbreviated "cnst." The constant model only has one parameter, the scale term. For the constant model $\mu_i$ is given by

$$\mu_i = 0. \tag{5}$$

The second model in the fitting algorithm is a constrained Hill model (hill), where the bottom asymptote is forced to 0. Including the scale parameter, the Hill model has four parameters. Let $tp$ be the top asymptote, $ga$ be the $AC_{50}$[4] in the gain direction, $gw$ be the Hill coefficient in the gain direction, and $x_i$ be the log concentration at the $i^{th}$ observation. Then $\mu_i$ for the Hill model is given by

$$\mu_i = \frac{tp}{1 + 10^{(ga - x_i)gw}}, \tag{6}$$

[4]The $AC_{50}$ is the activity concentration at 50%, or the concentration where the modeled activity equals 50% of the top asymptote.

with the constraints

$$0 \leq tp \leq 1.2\text{max resp}, \tag{7}$$

$$\min \text{logc} - 2 \leq ga \leq \max \text{logc} + 0.5, \tag{8}$$

and

$$0.3 \leq gw \leq 8. \tag{9}$$

The third model in the fitting algorithm is a constrained gain-loss model (gnls), defined as a product of two Hill models, with a shared top asymptote and both bottom asymptote values equal to 0. Including the scale term, the gain-loss model has six parameters. Let $tp$ be the shared top asymptote, $ga$ be the $AC_{50}$ in the gain direction, $gw$ be the Hill coefficient in the gain direction, $la$ be the $AC_{50}$ in the loss direction, $lw$ be the Hill coefficient in the loss direction, and $x_i$ be the log concentration at the $i^{th}$ observation. Then $\mu_i$ for the gain-loss model is given by

$$\mu_i = tp \left( \frac{1}{1 + 10^{(ga - x_i)gw}} \right) \left( \frac{1}{1 + 10^{(x_i - la)lw}} \right), \tag{10}$$

with the constraints

$$0 \leq tp \leq 1.2\text{max resp}, \tag{11}$$

$$\min \text{logc} - 2 \leq ga \leq \max \text{logc}, \tag{12}$$

$$0.3 \leq gw \leq 8, \tag{13}$$

$$\min \text{logc} - 2 \leq la \leq \max \text{logc} + 2, \tag{14}$$

$$0.3 \leq lw \leq 18, \tag{15}$$

and

$$la - ga > 0.25. \tag{16}$$

Level 4 does not utilize any assay endpoint-specific methods; the user only needs to run the `tcplRun` function. **Level 4 processing and all subsequent processing is done by assay endpoint, not assay component**. The previous section showed how to find the assay endpoints for an assay component using the `tcplLoadAeid` function. The example dataset includes two assay endpoints with aeid values of 1 and 2.

R Input

```
> ## Do level 4 processing for aeid 1 and load the data
> mc4_res <- tcplRun(id = 1:2, slvl = 4, elvl = 4, type = "mc")
```

R Output

```
4 mins
```

The level 4 data include 52 variables, including the ID fields. A complete list of level 4 fields is available in Appendix A. The level 4 data include the fields *cnst*, *hill*, and *gnls* indicating the convergence of the model where a value of 1 means the model converged and a value of 0 means the model did not converge. N/A values indicate the fitting algorithm did not attempt to fit the model. *cnst* will be N/A when the concentration series had less than 4 concentrations; *hill* and *gnls* will be N/A when none of the medians were greater than or equal to $3bmad$. Similarly, the *hcov* and *gcov* fields indicate the success in inverting the Hessian matrix. Where the Hessian matrix did not invert, the parameter standard deviation estimates will be N/A. NaN values in the parameter standard deviation fields indicate the covariance matrix was not positive definite. In Figure 2 the *hill* field is used to find potentially active compounds to visualize with the `tcplPlotL4ID` function.

R Input

```
> ## Load the level 4 data
> m4dat <- tcplLoadData(lvl = 4, type = "mc")
> ## List the first m4ids where the hill model convered
> ## for AEID 1
> m4dat[hill == 1 & aeid == 1, head(m4id)]
```

R Output

```
[1]  3 15 16 19 21 34
```

The model summary values in Figure 2 include Akaike Information Criterion (AIC), probability, and the root mean square error (RMSE). Let $\log(\mathcal{L}(\hat{\theta}, y))$ be the log-likelihood of the model $\hat{\theta}$ given the observed values $y$, and $K$ be the number of parameters in $\hat{\theta}$, then,

$$\text{AIC} = -2\log(\mathcal{L}(\hat{\theta}, y)) + 2K. \tag{17}$$

The probability, $\omega_i$, is defined as the weight of evidence that model $i$ is the best model, given that one of the models must be the best model. Let $\Delta_i$ be the difference $\text{AIC}_i - \text{AIC}_{min}$ for the $i^{th}$ model. If $R$ is the set of models, then $\omega_i$ is given by

$$\omega_i = \frac{\exp\left(-\frac{1}{2}\Delta_i\right)}{\sum_{i=1}^{R}\exp\left(-\frac{1}{2}\Delta_r\right)}. \tag{18}$$

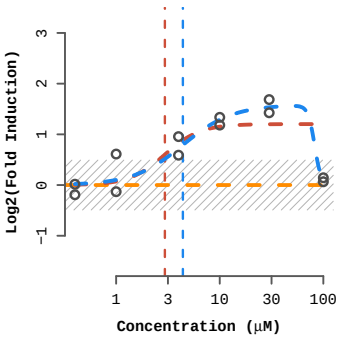
The RMSE is given by

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \mu_i)^2}{N}}, \tag{19}$$

where $N$ is the number of observations, and $\mu_i$ and $y_i$ are the estimated and observed values at the $i^{th}$ observation, respectively.

R Input

```
> ## Plot a fit for m4id 21
> tcplPlotM4ID(m4id = 686, lvl = 4)
```




```
ASSAY:   AEID2 (CTox_CORT_dn)

NAME:    Norgestrel
CHID:    568     CASRN: 797-63-7
SPID(S): 01141142A
M4ID:    686

HILL MODEL (in red):
      tp       ga       gw
val:  1.2      0.47     2.55
sd:   0.267    0.343    3.18

GAIN-LOSS MODEL (in blue):
      tp       ga       gw       la       lw
val:  1.58     0.644    1.81     1.92     13.6
sd:   NaN      NaN      NaN      NaN      NaN

      CNST        HILL        GNLS
AIC:  34.63       25.34       7.07
PROB: 0           0           1
RMSE: 0.9         0.51        0.2

MAX_MEAN: 1.56      MAX_MED: 1.56      BMAD: 0.164
```

Figure 2: An example level 4 plot for a single concentration series. The orange dashed line shows the constant model, the red dashed line shows the Hill model, and the blue dashed line shows the gain-loss model. The gray striped box shows the baseline region, $0 \pm 3bmad$. The summary panel shows assay endpoint and sample information, the parameter values (val) and standard deviations (sd) for the Hill and gain-loss models, and summary values for each model.

## Level 5

Level 5 processing determines the winning model and activity for the concentration series, bins all of the concentration series into categories, and calculates additional point-of-departure estimates based on the activity cutoff.

**The model with lowest AIC value is selected as the winning model (*modl*)**, and is used to determine the activity or hit-call for the concentration series. If two models have equal AIC values, the simpler model (the model with fewer parameters) wins the tie. All of the parameters for the winning model are stored at level 5 with the prefix "modl_" to facilitate easier queries. For a concentration series to get an active hit-call, either the Hill or gain-loss must be selected as the winning model. In addition to selecting the Hill or gain-loss model, the modeled and observed response must meet an efficacy cutoff.

The efficacy cutoff is defined by the level 5 methods. The efficacy cutoff value (*coff*) is defined as the maximum of all values given by the assigned level 5 methods. Failing to assign a level 5 method will result in every

concentration series being called active. For a complete list of level 5 methods see `tcplMthdList(lvl = 5)` or `?MC5_Methods`.

R Input

```
> ## Assign a cutoff value of bmad*6
> tcplMthdAssign(lvl = 5,
                 id = 1:2,
                 mthd_id = 6,
                 type = "mc")
```

R Output

```
Completed delete cascade for 2 ids (0.01 secs)
```

For the example data the cutoff value is $6bmad$. If the Hill or gain-loss model wins, and the estimated top parameter for the winning model (*modl_tp*) and the maximum median value (*max_med*) are both greater than or equal to the efficacy cutoff (*coff*), the concentration series is considered active and the hit-call (*hitc*) is set to 1.

The hit-call can be 1, 0, or -1. A hit-call of 1 or 0 indicates the concentration series is active or inactive, respectively, according to the analysis; a hit-call of -1 indicates the concentration series had less than four concentrations.

For active concentration series, two additional point-of-departure estimates are calculated for the winning model: (1) the activity concentration at baseline (ACB or *modl_acb*) and (2) the activity concentration at cutoff (ACC or *modl_acc*). The ACB and ACC are defined as the concentration where the estimated model value equals $3bmad$ and the cutoff, respectively. The point-of-departure estimates are summarized in Figure 3.

All concentration series fall into a single fit category (*fitc*), defined by the leaves on the tree structure in Figure 4. Concentration series in the same category will have similar characteristics, and often look very similar. Categorizing all of the series enables faster quality control checking and easier identification of potential false results. The first split differentiates series by hit-call. Series with a hit-call of -1 go into fit category 2. The following two paragraphs will outline the logic for the active and inactive branches.

The first split in the active branch differentiates series by the model winner, Hill or gain-loss. For each model, the next split is defined by the efficacy of it's top parameter in relation to the cutoff. The top value is either less than $1.2coff$ or greater than or equal to $1.2coff$. Finally, series on the active branch go into leaves based on the position of the $AC_{50}$ parameter in relation to the tested concentration range. For comparison purposes, the activity concentration at

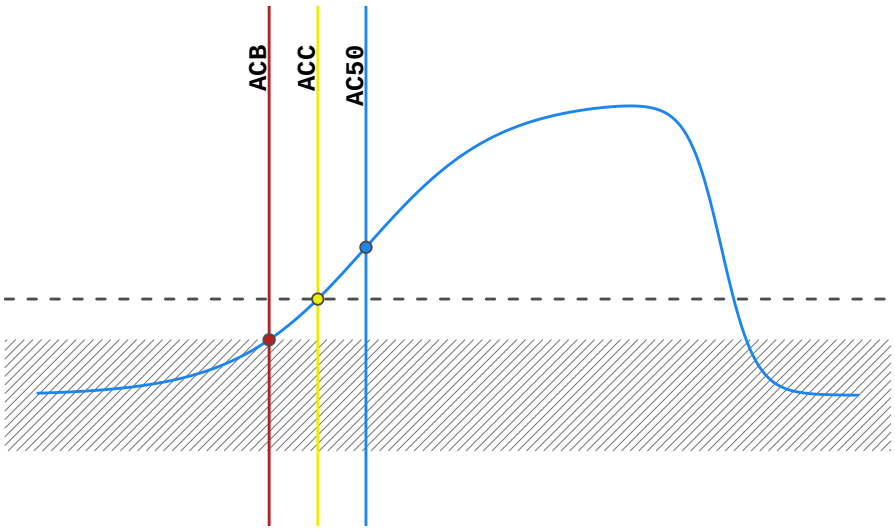



Figure 3: The point-of-departure estimates calculated by the `tcpl` package. The shaded rectangle represents the baseline region, $0 \pm 3bmad$. The dark stripped line represents the efficacy cutoff ($coff$). The vertical lines show where the point-of-departure estimates are defined: the red line shows the ACB, the yellow line shows the ACC, and the blue line shows the $AC_{50}$.

95% (AC95) is calculated, but not stored.[5] Series with $AC_{50}$ values less than the minimum concentration tested ($logc\_min$) go into the "<=" leaves, series with $AC_{50}$ values greater than the minimum tested concentration and AC95 values less than maximum tested concentration ($logc\_max$) go into the "==" leaves, and series with AC95 values greater than the maximum concentration tested go into the ">=" leaves.

The inactive branch is first divided by whether any median values were greater than or equal to $3bmad$. Series with no evidence of activity go into fit category 4. Similar to the active branch, series with evidence for activity are separated by the model winner. The Hill and gain-loss portions of the inactive branch follow the same logic. First, series diverge by the efficacy of their top parameter in relation to the cutoff: $modl\_tp < 0.8\,coff$ or $modl\_tp \geq 0.8\,coff$. Then the same comparison is made on the top values of the losing model. If the losing model did not converge, then the series go into the "DNC" category. If the losing model top value is greater than or equal to $0.8coff$, then the series are split based on whether the losing model top surpassed the cutoff. On the constant model branch, if neither top parameter is greater than or equal to $0.8bmad$, then the series goes into fit category 7. If one of the top parameters is

[5]The `tcplHill-` functions can be used to calculate values, concentrations, and activity concentrations for the Hill model.

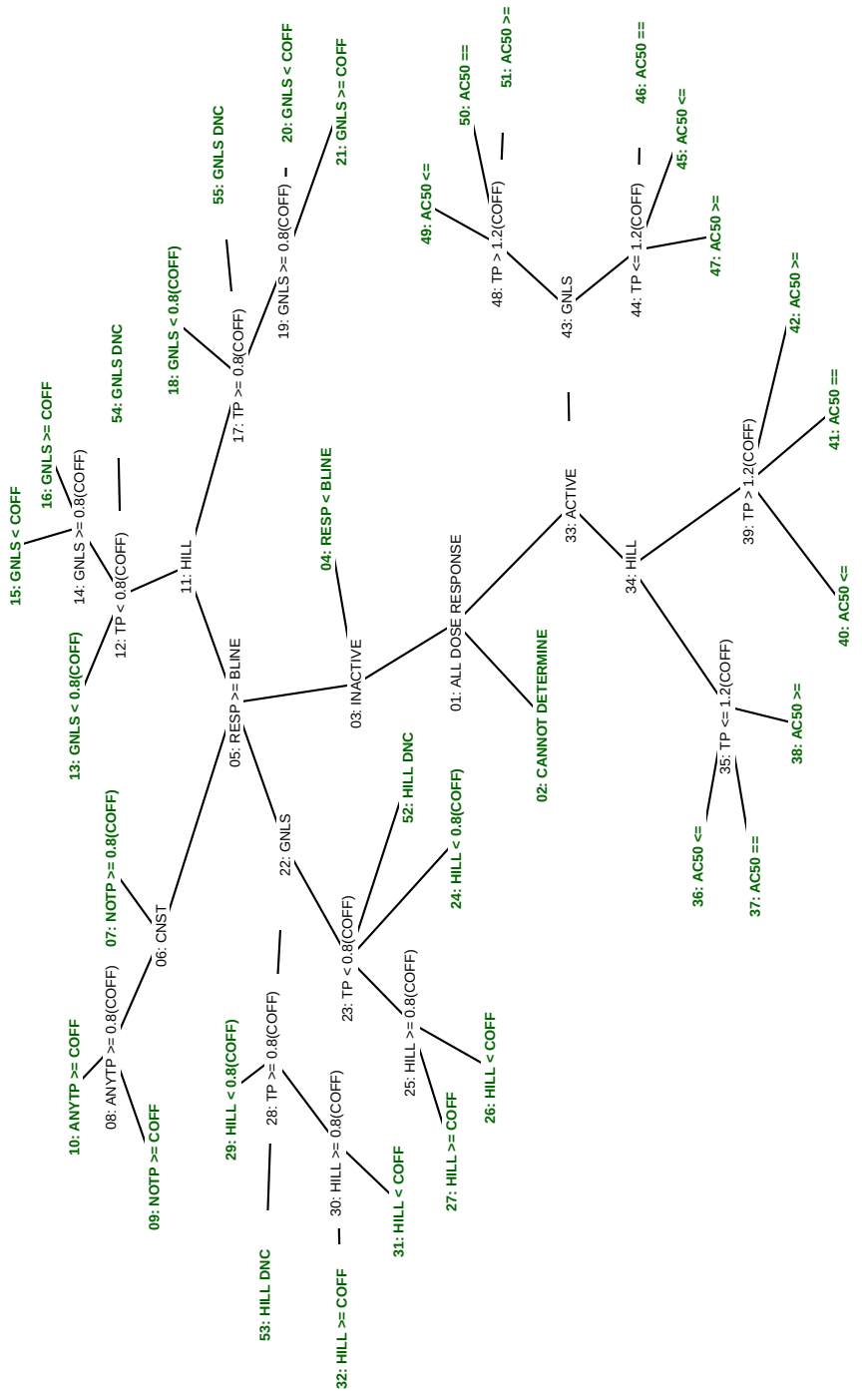



Figure 4: The categories used to bin each fit. Each fit falls into one leaf of the tree. The leaves are indicated by bold green font. (Figure created by calling `tcplPlotFitc()`.)

greater than or equal to $0.8coff$, the series goes into fit category 9 or 10 based on whether one of the top values surpassed the cutoff.

With the level 5 methods assigned, the data are ready for level 5 processing:

R Input

```
> ## Do level 5 processing for aeid 1 and load the data
> mc5_res <- tcplRun(id = 1:2, slvl = 5, elvl = 5, type = "mc")
```

R Output

```
Total processing time: 0 mins
```

R Input

```
> tcplPlotM4ID(m4id = 370, lvl = 5)
```

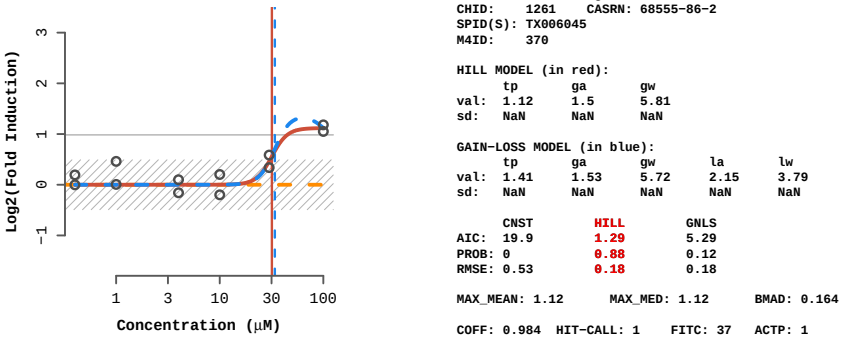



Figure 5: An example level 5 plot for a single concentration series. The solid line and model highlighting indicate the model winner. The horizontal line shows the cutoff value. In addition to the information from the level 4 plots, the summary panel includes the cutoff (*coff*), hit-call (*hitc*), fit category (*fitc*) and activity probability (*actp*) values.

Figure 5 shows an example of a concentration series in fit category 37, indicating the series is active and the Hill model won with a top value less than or equal to $1.2coff$, and an $AC_{50}$ value within the tested concentration range.

The `tcplPlotFitc` function shows the distribution of concentration series across the fit category tree (Figure 6).

R Input

```
> m5dat <- tcplLoadData(lvl = 5, type = "mc")
> tcplPlotFitc(fitc = m5dat$fitc)
```

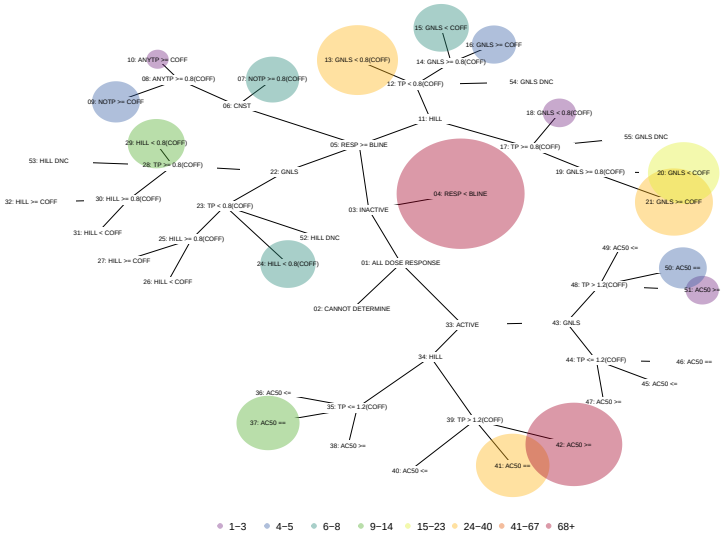



Figure 6: The distribution of concentration series by fit category for the example data. Both the size and color of the circles indicate the number of concentration series. The legend gives the range for number of concentration series by color.

The distribution in Figure 6 shows 24-40 concentration series fell into fit category 21. Following the logic discussed previously, fit category 21 indicates an inactive series where the Hill model was selected, the top asymptote for the Hill model was greater than $0.8coff$, and the gain-loss top asymptote was greater than or equal to the cutoff. The series in fit category 21 can be found easily in the level 5 data.

R Input

```
> head(m5dat[fitc == 21,
             list(m4id, hill_tp, gnls_tp,
                  max_med, coff, hitc)])
```

R Output

```
   m4id   hill_tp  gnls_tp   max_med      coff hitc
1:    3 1.1483868 1.148420 0.9419547 0.9836658    0
2:   21 1.2195154 1.219515 0.9644619 0.9836658    0
3:   45 1.0100353 1.010033 0.7313420 0.9836658    0
4:   46 1.0853209 1.085321 0.8303106 0.9836658    0
5:  125 0.9852517 1.008044 0.8201074 0.9836658    0
6:  174 0.9736302 1.107837 0.8692471 0.9836658    0
```

The plot in Figure 7 shows a concentration series in fit category 21. In the example given by Figure 7, the *hill_tp* and *gnls_tp* parameters are equal and greater than *coff*; however, the maximum median value (*max_med*) is not greater than the cutoff making the series inactive.

R Input

```
> tcplPlotM4ID(m4id = 45, lvl = 5)
```

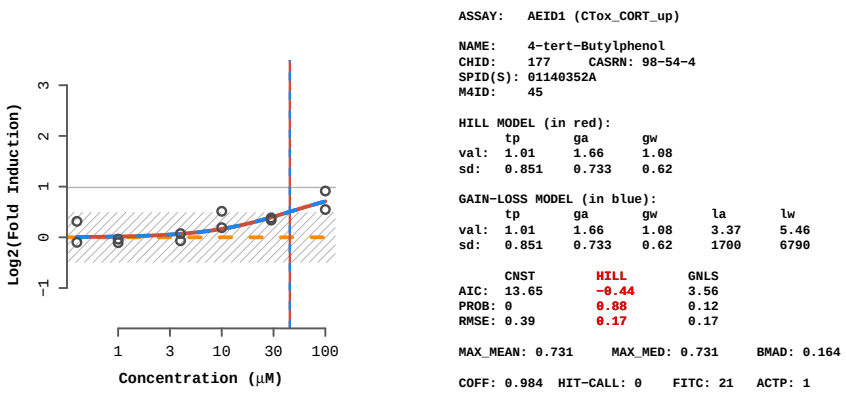



Figure 7: Level 5 plot for m4id 45 showing an example series in fit category 21.

## Level 6

Level 6 processing uses various methods to identify concentration series with etiologies that may suggest false positive/false negative results or explain apparent anomalies in the data. Each flag has is defined by a level 6 method

that has to be assigned to each assay endpoint. Similar to level 5, an assay endpoint does not need any level 6 methods assigned to complete processing.

R Input

```
> tcplMthdAssign(lvl = 6,
                 id = 1:2,
                 mthd_id = c(6:8, 10:12, 15:16),
                 type = "mc")
```

R Output

```
Completed delete cascade for 2 ids (0.01 secs)
```

R Input

```
> tcplMthdLoad(lvl = 6, id = 1, type = "mc")
```

R Output

```
   aeid              mthd mthd_id nddr
1:    1 singlept.hit.high       6    0
2:    1  singlept.hit.mid       7    0
3:    1    multipoint.neg       8    0
4:    1             noise      10    0
5:    1       border.hit      11    0
6:    1      border.miss      12    0
7:    1     gnls.lowconc      15    0
8:    1      overfit.hit      16    0
```

The example above assigns the most common flags. Some of the available flags only apply to specific experimental designs and do not apply to all data. For a complete list of normalization methods see `tcplMthdList(lvl = 6)` or `?MC6_Methods`.

The additional *nddr* field in the "mc6_methods" (and the output from `tcplMthdLoad()`/`tcplMthdList()` for level 6) indicates whether the method requires additional data. Methods with an *nddr* value of 0 only require the modeled/summary information from levels 4 and 5. Methods with an *nddr* value of 1 also require the individual response and concentration values from level 3. Methods requiring data from level 3 can greatly increase the processing time.

R Input

```
> ## Do level 6 processing
> mc6_res <- tcplRun(id = 1:2, slvl = 6, elvl = 6, type = "mc")
```

R Output

```
Total processing time: 0.03 mins
```

R Input

```
> m6dat <- tcplLoadData(lvl = 6, type = "mc")
```

For the two assay endpoints, 268 out of the 1048 concentration series were flagged in the level 6 processing. Series not flagged in the level 6 processing do not get stored at level 6. Each series-flag combination is a separate entry in the level 6 data. Or, in other words, if a series has multiple flags it will show up on multiple rows in the output. For example, consider the following results:

R Input

```
> m6dat[m4id == 46]
```

R Output

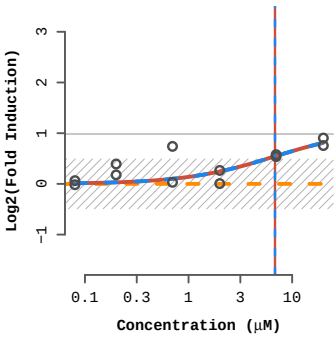
```
   aeid m6id m4id m5id      spid mc6_mthd_id
1:    1    5   46   46 01140354A           8
2:    1  110   46   46 01140354A          12
                                       flag fval fval_unit
1: Multiple points above baseline, inactive   NA        NA
2:                      Borderline inactive   NA        NA
```

The data above lists two flags: “Multiple points above baseline, inactive” and “Borderline inactive.” Without knowing much about the flags one might assume this concentration series had some evidence of activity but was not called a hit, and could potentially be a false negative. In cases of borderline results, plotting the curve is often helpful.

The evidence of true activity shown in Figure 8 could be argued either way. Level 6 processing does not attempt to define truth in the matter of borderline compounds or data anomalies, but rather attempts to identify concentration series for closer consideration.

R Input

```
> tcplPlotM4ID(m4id = 46, lvl = 6)
```




```
ASSAY:   AEID1 (CTox_CORT_up)

NAME:    2-Aminoanthraquinone
CHID:    178     CASRN: 117-79-3
SPID(S): 01140354A
M4ID:    46

HILL MODEL (in red):
      tp       ga       gw
val:  1.09     0.834    1
sd:   0.542    0.476    0.88

GAIN-LOSS MODEL (in blue):
      tp       ga       gw       la       lw
val:  1.09     0.834    1        2.88     4.97
sd:   0.542    0.476    0.88     1700     5330

      CNST        HILL        GNLS
AIC:  19.78       3.68        7.68
PROB: 0           0.88        0.12
RMSE: 0.48        0.23        0.23

MAX_MEAN: 0.83      MAX_MED: 0.83      BMAD: 0.164

COFF: 0.984  HIT-CALL: 0    FITC: 21   ACTP: 1

FLAGS:
8; 12
```

Figure 8: An example level 6 plot for a single concentration series. All level 6 method ID (*l6_mthd_id*) values are concatenated in the flags section. If flags have an associated value (*fval*), the value will be shown in parentheses to the right of the level 6 method ID.

# A Field Explanation/Database Structure

This appendix contains reference tables that describe the structure and table fields found in the `tcpl` database. The first sections of this appendix describe the data-containing tables, followed by a section describing the additional annotation tables.

In general, the single-concentration data and accompanying methods are found in the "sc#" tables, where the number indications the processing level. Likewise, the multiple-concentration data and accompanying methods are found in the "mc#" tables. Each processing level that has accompanying methods will also have a tables with the "_methods" and "_id" naming scheme. For example, the database contains the following tables: "mc5" storing the data from multiple-concentration level 5 processing, "mc5_methods" storing the available level 5 methods, and "mc5_aeid" storing the method assignments for level 5. Note, the table storing the method assignments for level 2 multiple-concentration processing is called "mc2_acid" because MC2 methods are assigned by assay component ID.

There are two additional tables, "sc2_agg" and "mc4_agg," that link the data in tables "sc2" and "mc4" to the data in tables "sc1" and "mc3," respectively. This is necessary because each entry in the database before SC2 and MC4 processing represents a single value; subsequent entries represent summary/modeled values that encompass many values. To know what values were used in calculating the summary/modeled values, the user must use the "_agg" look-up tables.

Each of the methods tables have fields analogous to *mc5_mthd_id*, *mc5_mthd*, and *desc*. These fields represent the unique key for the method, the abbreviated method name (used to call the method from the corresponding `mc5_mthds` function), and a brief description of the method, respectively. The "mc6_methods" table may also includes *nddr* field. More information about *nddr* is available in the discussion of multiple-concentration level 6 processing (page 40).

The method assignment tables will have fields analogous to *mc5_mthd_id* matching the method ID from the methods tables, an assay component or assay endpoint ID, and possibly an *exec_ordr* field indicating the order in which to execute the methods.

The method and method assignment tables will not be listed in the tables below to reduce redundancy.

Many of the tables also include the *created_date*, *modified_date*, and *modified_by* fields that store information helpful for tracking changes to the data. These fields will not be discussed further or included in the tables below.

Many of the tables specific to the assay annotation are not utilized by the `tcpl` package. The full complexity of the assay annotation used by the ToxCast program is beyond the scope of this vignette and the `tcpl` package. More information about the ToxCast assay annotation can be found at: `<http://epa.gov/ncct/toxcast/data.html>`.

## Single-concentration data-containing tables

Table 5: Fields in sc0 table.

| Field | Description |
|---|---|
| s0id | Level 0 ID |
| acid | Assay component ID |
| spid | Sample ID |
| cpid | Chemical plate ID |
| apid | Assay plate ID |
| rowi | Assay plate row index |
| coli | Assay plate column index |
| wllt | Well type[†] |
| wllq | 1 if the well quality was good, else 0[‡] |
| conc | Concentration in micromolar |
| rval | Raw assay component value/readout from vendor |
| srcf | Filename of the source file containing the data |

[†]Information about the different well types is available in Appendix B.

Table 6: Fields in sc1 table.

| Field | Description |
|---|---|
| s1id | Level 1 ID |
| s0id | Level 0 ID |
| acid | Assay component ID |
| aeid | Assay component endpoint ID |
| logc | Log base 10 concentration |
| bval | Baseline value |
| pval | Positive control value |
| resp | Normalized response value |

Table 7: Fields in sc2_agg table.

| Field | Description |
|---|---|
| aeid | Assay component endpoint ID |
| s0id | Level 0 ID |
| s1id | Level 1 ID |
| s2id | Level 2 ID |

Table 8: Fields in sc2 table.

| Field | Description |
|---|---|
| s2id | Level 2 ID |
| aeid | Assay component endpoint ID |
| spid | Sample ID |
| bmad | Baseline median absolute deviation |
| max_med | Maximum median response value |
| hitc | Hit-/activity-call, 1 if active, 0 if inactive |
| coff | Efficacy cutoff value |
| tmpi | Ignore, temporary index used for uploading purposes |

## Multiple-concentration data-containing tables

The "mc0" table, other than containing *m0id* rather than *s0id*, is identical to the "sc0" described in the section above.

Table 9: Fields in mc1 table.

| Field | Description |
|---|---|
| m1id | Level 1 ID |
| m0id | Level 0 ID |
| acid | Assay component ID |
| cndx | Concentration index |
| repi | Replicate index |

Table 10: Fields in mc2 table.

| Field | Description |
|---|---|
| m2id | Level 2 ID |
| m0id | Level 0 ID |
| acid | Assay component ID |
| m1id | Level 1 ID |
| cval | Corrected value |

Table 11: Fields in mc3 table.

| Field | Description |
|---|---|
| m3id | Level 3 ID |
| aeid | Assay endpoint ID |
| m0id | Level 0 ID |
| acid | Assay component ID |
| m1id | Level 1 ID |
| m2id | Level 2 ID |
| bval | Baseline value |
| pval | Positive control value |
| logc | Log base 10 concentration |
| resp | Normalized response value |

Table 12: Fields in mc4_agg table.

| Field | Description |
|---|---|
| aeid | Assay endpoint ID |
| m0id | Level 0 ID |
| m1id | Level 1 ID |
| m2id | Level 2 ID |
| m3id | Level 3 ID |
| m4id | Level 4 ID |

Table 13: Fields in mc4 table (Part 1).

| Field | Description |
|---|---|
| m4id | Level 4 ID |
| aeid | Assay endpoint ID |
| spid | Sample ID |
| bmad | Baseline median absolute deviation |
| resp_max | Maximum response value |
| resp_min | Minimum response value |
| max_mean | Maximum mean response value |
| max_mean_conc | Log concentration at *max_mean* |
| max_med | Maximum median response value |
| max_med_conc | Log concentration at *max_med* |
| logc_max | Maximum log concentration tested |
| logc_min | Minimum log concentration tested |
| cnst | 1 if the constant model converged, 0 if it failed to converge, N/A if series had less than four concentrations |
| hill | 1 if the Hill model converged, 0 if it failed to converge, N/A if series had less than four concentrations or if $max\_med < 3bmad$ |
| hcov | 1 if the Hill model Hessian matrix could be inverted, else 0 |
| gnls | 1 if the gain-loss model converged, 0 if it failed to converge, N/A if series had less than four concentrations or if $max\_med < 3bmad$ |
| gcov | 1 if the gain-loss model Hessian matrix could be inverted, else 0 |
| cnst_er | Scale term for the constant model |
| cnst_aic | AIC for the constant model |
| cnst_rmse | RMSE for the constant model |
| cnst_prob | Probability the constant model is the true model |
| hill_tp | Top asymptote for the Hill model |
| hill_tp_sd | Standard deviation for *hill_tp* |
| hill_ga | $AC_{50}$ for the Hill model |
| hill_ga_sd | Standard deviation for *hill_ga* |

Table 14: Fields in mc4 table (Part 2).

| Field | Description |
|---|---|
| hill_gw | Hill coefficient |
| hill_gw_sd | Standard deviation for *hill_gw* |
| hill_er | Scale term for the Hill model |
| hill_er_sd | Standard deviation for *hill_er* |
| hill_aic | AIC for the Hill model |
| hill_rmse | RMSE for the Hill model |
| hill_prob | Probability the Hill model is the true model |
| gnls_tp | Top asymptote for the gain-loss model |
| gnls_tp_sd | Standard deviation for *gnls_tp* |
| gnls_ga | $AC_{50}$ in the gain direction for the gain-loss model |
| gnls_ga_sd | Standard deviation for *gnls_ga* |
| gnls_gw | Hill coefficient in the gain direction |
| gnls_gw_sd | Standard deviation for *gnls_gw* |
| gnls_la | $AC_{50}$ in the loss direction for the gain-loss model |
| gnls_la_sd | Standard deviation for *gnls_la* |
| gnls_lw | Hill coefficient in the loss direction |
| gnls_lw_sd | Standard deviation for *gnls_lw* |
| gnls_er | Scale term for the gain-loss model |
| gnls_er_sd | Standard deviation for *gnls_er* |
| gnls_aic | AIC for the gain-loss model |
| gnls_rmse | RMSE for the gain-loss model |
| gnls_prob | Probability the gain-loss model is the true model |
| nconc | Number of concentrations tested |
| npts | Number of points in the concentration series |
| nrep | Number of replicates in the concentration series |
| nmed_gtbl | Number of median values greater than $3bmad$ |
| tmpi | Ignore, temporary index used for uploading purposes |

Table 15: Fields in mc5 table.

| Field | Description |
|---|---|
| m5id | Level 5 ID |
| m4id | Level 4 ID |
| aeid | Assay endpoint ID |
| modl | Winning model: “cnst”, “hill”, or “gnls” |
| hitc | Hit-/activity-call, 1 if active, 0 if inactive, -1 if cannot determine |
| fitc | Fit category |
| coff | Efficacy cutoff value |
| actp | Activity probability ($1 - cnst\_prob$) |
| modl_er | Scale term for the winning model |
| modl_tp | Top asymptote for the winning model |
| modl_ga | Gain $AC_{50}$ for the winning model |
| modl_gw | Gain Hill coefficient for the winning model |
| modl_la | Loss $AC_{50}$ for the winning model |
| modl_lw | Loss Hill coefficient for the winning model |
| modl_prob | Probability for the winning model |
| modl_rmse | RMSE for the winning model |
| modl_acc | Activity concentration at cutoff for the winning model |
| modl_acb | Activity concentration at baseline for the winning model |
| modl_ac10 | AC10 for the winning model |

Table 16: Fields in mc6 table.

| Field | Description |
|---|---|
| m6id | Level 6 ID |
| m5id | Level 5 ID |
| m4id | Level 4 ID |
| aeid | Assay endpoint ID |
| m6_mthd_id | Level 6 method ID |
| flag | Text text output for the level 6 method |
| fval | Value from the flag method, if applicable |
| fval_unit | Units for *fval*, if applicable |

## Auxiliary annotation tables

As mentioned in the introduction to this appendix, a full description of the assay annotation is beyond the scope of this vignette. The fields pertinent to the `tcpl` package are listed in the tables below.

Table 17: List of annotation tables.

| Table Name | Description |
|---|---|
| assay | Assay-level annotation |
| assay_component | Assay component-level annotation |
| assay_component_endpoint | Assay endpoint-level annotation |
| assay_component_map | Assay component source names and their corresponding assay component ids |
| assay_reagent* | Assay reagent information |
| assay_reference* | Map of citations to assay |
| assay_source | Assay source-level annotation |
| chemical | List of chemicals and associated identifiers |
| chemical_library | Map of chemicals to different chemical libraries |
| citations* | List of citations |
| gene | Gene* identifiers and descriptions |
| intended_target* | Intended assay target at the assay endpoint level |
| mc5_fit_categories | The level 5 fit categories |
| organism* | Organism identifiers and descriptions |
| sample | Sample ID information and chemical ID mapping |
| technological_target* | Technological assay target at the assay component level |

* indicates tables not currently used by the `tcpl` package

Table 18: Fields in assay.

| Field | Description |
|---|---|
| aid | Assay ID |
| asid | Assay source ID |
| assay_name | Assay name (abbreviated "anm" within the package) |
| assay_desc | Assay description |
| timepoint_hr | Treatment duration in hours |
| assay_footprint | Microtiter plate size† |

† discussed further in the "Register and Upload New Data" section (page 6)

Table 19: Fields in assay_component.

| Field | Description |
|---|---|
| acid | Assay component ID |
| aid | Assay ID |
| assay_component_name | Assay component name (abbreviated "acnm" within the package) |
| assay_component_desc | Assay component description |

Table 20: Fields in assay_source.

| Field | Description |
|---|---|
| asid | Assay source ID |
| assay_source_name | Assay source name (typically an abbreviation of the assay_source_long_name, abbreviated "asnm" within the package) |
| assay_source_long_name | The full assay source name |
| assay_source_description | Assay source description |

Table 21: Fields in assay_component_endpoint.

| Field | Description |
|---|---|
| aeid | Assay component endpoint ID |
| acid | Assay component ID |
| assay_component_endpoint_name | Assay component endpoint name (abbreviated "aenm" within the package) |
| assay_component_endpoint_desc | Assay component endpoint description |
| export_ready | 0 or 1, used to flag data as "done" |
| normalized_data_type | The units of the normalized data† |
| burst_assay | 0 or 1, 1 indicates the assay results should be used in calculating the burst z-score |
| fit_all | 0 or 1, 1 indicates all results should be fit, regardless of whether the $max\_med$ surpasses $3bmad$ |

† discussed further in the "Register and Upload New Data" section (page 6)

Table 22: Fields in assay_component_map table.

| Field | Description |
|---|---|
| acid | Assay component ID |
| acsn | Assay component source name |

Table 23: Fields in chemical.

| Field | Description |
|---|---|
| chid | Chemical ID† |
| casn | CAS Registry Number |
| chnm | Chemical name |

† this is the DSSTox GSID within the ToxCast data, but can be any integer and will be auto-generated (if not explicitly defined) for newly registered chemicals

Table 24: Fields in chemical_library.

| Field | Description |
|---|---|
| chid | Chemical ID |
| clib | Chemical library |

Table 25: Fields in mc5_fit_categories table.

| Field | Description |
|---|---|
| fitc | Fit category |
| parent_fitc | Parent fit category |
| name | Fit category name |
| xloc | x-axis location for plotting purposes |
| yloc | y-axis location for plotting purposes |

Table 26: Fields in sample.

| Field | Description |
|---|---|
| spid | Sample ID |
| chid | Chemical ID |
| stkc | Stock concentration |
| stkc_unit | Stock concentration unit |
| tested_conc_unit | The concentration unit for the concentration values in the data-containing tables |
| spid_legacy | A place-holder for previous sample ID strings |

The stock concentration fields in the "sample" table allow the user to track the original concentration when the neat sample is solubilized in vehicle before any serial dilutions for testing purposes.

# B Level 0 Pre-processing

Level 0 pre-processing can be done on virtually any high-throughput/high-content screening application. In the ToxCast program, level 0 processing is done in R by vendor/dataset-specific scripts. The individual R scripts act as the "laboratory notebook" for the data, with all pre-processing decisions clearly commented and explained.

Level 0 pre-processing has to reformat the raw data into the standard format for the pipeline, and also can make manual changes to the data. All manual changes to the data should be very well documented with justification. Common examples of manual changes include fixing a sample ID typo, or changing well quality value(s) to 0 after finding obvious problems like a plate row/column missing an assay reagent.

Each row in the level 0 pre-processing data represents one well-assay component combination, containing 11 fields (Table 27). The only field in level 0 pre-processing not stored at level 0 is the assay component source name (*acsn*). The assay component source name should be some concatenation of data from the assay source file that identifies the unique assay components. When the data are loaded into the database, the assay component source name is mapped to assay component ID through the assay_component_map table in the `tcpl` database. Assay components can have multiple assay component source names, but each assay component source name can only map to a single assay component.

The well type field is used in the processing to differentiate controls from test compounds in numerous applications, including normalization and definition of the assay noise level. Currently, the `tcpl` package includes the eight well types in Table 28. Package users are encouraged to suggest new well types and methods to better accommodate their data.

The final step in level 0 pre-processing is loading the data into the `tcpl` database. The `tcpl` package includes the `tcplWriteLvl0` function to load data into the database. The `tcplWriteLvl0` function maps the assay component source name to the appropriate assay component ID, checks each field for the correct class, and checks the database for the sample IDs with well type "t." Each test compound sample ID must be included in the `tcpl` database before loading data. The `tcplWriteLvl0` also checks each test compound for concentration values.

Table 27: Required fields in level 0 pre-processing.

| Field | Description | N/A |
|---|---|---|
| acsn | Assay component source name | No |
| spid | Sample ID | No |
| cpid | Chemical plate ID | Yes |
| apid | Assay plate ID | Yes |
| rowi | Assay plate row index, as an integer | Yes |
| coli | Assay plate column index, as an integer | Yes |
| wllt | Well type | No |
| wllq | 1 if the well quality was good, else 0 | No |
| conc | Concentration in micromolar | No[†] |
| rval | Raw assay component value/readout from vendor | Yes[‡] |
| srcf | Filename of the source file containing the data | No |

The N/A column indicates whether the field can be N/A in the pre-processed data.
[†]Concentration can be N/A for control values only tested at a single concentration. Concentration cannot be N/A for any test compound (well type of "t") data.
[‡]If the raw value is N/A, well type has to be 0.

Table 28: Well types

| Well Type | Description |
|---|---|
| t | Test compound |
| c | Gain-of-signal control in multiple concentrations |
| p | Gain-of-signal control in single concentration |
| n | Neutral/negative control |
| m | Loss-of-signal control in multiple concentrations |
| o | Loss-of-signal control in single concentration |
| b | Blank well |
| v | Viability control |

# C Cytotoxicity Distribution

Recognizing the susbtantial impact of cytotoxicity in confounding high-throughput and high-content screening results, the `tcpl` package includes methodology for defining chemical-specific cytotoxicity estimates. Our observations based on ToxCast data suggest a complex, and not-yet fully understood, cellular biology that includes non-specific activation of many targets as cells approach death. For example, a chemical may induce activity in an estrogen-related assay, but if that chemical also causes activity in hundreds of other assays at or around the same concentration as cytotoxicity, should the chemical be called an estrogen agonist? The `tcplCytoPt` function provides an estimate of chemical-specific cytotoxicity points to provide some context to the "burst" phenomenon.

The cytotoxicity point is simply the median $AC_{50}$ for a set of assay endpoints, either given by the user or defined within the `tcpl` database. By default, the `tcplCytoPt` function uses the assay endpoints listed in the *burst_assay* field of the "assay_component_endpoint" table, where 1 indicates to include the assay endpoint in the calculation. The "burst" assay endpoints can be indentified by running `tcplLoadAeid(fld = "burst_assay", val = 1)`.

In addition to the cytotoxicity point, `tcplCytoPt` provides two additional estimates: (1) the MAD of the $AC_{50}$ (*modl_ga*) values used to calculate the cytotoxicity point, and (2) the global MAD. Note, only active assay endpoints (where the hit-call, *hitc*, equals 1) are included in the calculations. Once the burst distribution (cytotoxicity point and MAD) is defined for each chemical, the global burst MAD is defined as the median of the MAD values. Not every chemcial may be tested in every "burst" assay, so the user can determine the minimum number of tested assays as a condition for the MAD value for a particular chemical to be included in the global MAD calculation. By default, if "aeid" is the vector of assay endpoints used in the calculation, `tcplCytoPt` requires the chemical to tested in at least `floor(0.8 * length(aeid))` assay endpoints to be included in the calculation. The user can specify to include all calculated MAD values (note, there must be at least two active assay endpoints to calculate the MAD) by setting 'min.test' to `FALSE`. The 'min.test' parameter also accepts a number, allowing the user to explicitly set the requirement.

The global MAD gives an estimate of overall cytotoxicity window, and allows for a cytotoxicity distrubtion to be determined for chemicals with less than two active "burst" assay endpoints. The cytotoxicity point for chemicals with less two active "burst" endpoints is set to the value given to the 'default.pt' parameter. By default the `tcplCytoPt` assigns 'default.pt' to 3.[6]

[6] $10^3 = 1000$, therefore, when using micromolar units, 3 is equivalent to 1 millimolar. 1 millimolar was chosen as an arbitrary high concentration (outside the testing range for ToxCast data), based on the principle all compounds are toxic if given in high enough concentration.

# D Build Variable Matrices

The `tcplVarMat` function creates chemical-by-assay matrices for the level 4 and level 5 data. When multiple sample-assay series exist for one chemical, a single series is selected by the `tcplSubsetChid` function. See `?tcplSubsetChid` for more information.

1. "modl_ga" – The $\log_{10} AC_{50}$ (in the gain direction) for the winning model.
2. "hitc" – The hit-call for the winning model.
3. "m4id" – The m4id, listing the concentration series selected by `tcplSubsetChid`.
4. "zscore" – The z-score (described below).
5. "tested" – 1 or 0, 1 indicating the chemical/assay pair was tested in either the single- or multiple-concentration format.
6. "tested_sc" – 1 or 0, 1 indicating the chemical/assay pair was tested in the single-concentration format
7. "tested_mc" – 1 or 0, 1 indicating the chemical/assay pair was tested in the multiple-concentration format
8. "ac50" – a modified $AC_{50}$ table (in non-log units) where assay/chemical pairs that were not tested, or tested and had a hitcall of 0 or $-1$ have the value $1e6$.
9. "neglogac50" – $-\log_{10} \frac{AC_{50}}{1e6}$ where assay/chemical pairs that were not tested, or tested and had a hitcall of 0 or $-1$ have the value 0.

The z-score calculation is based on the output from `tcplCytoPt` (Appendix C), and is calculated for each $AC_{50}$ value as follows:

$$zscore = -\frac{modl\_ga - cyto\_pt}{global\_mad}, \tag{20}$$

Note: the burst z-score values are multiplied by -1 to make values that are more potent relative to the burst distribution a higher positive z-score.

In addition the the standard matrices, additional matrices can be defined by the 'add.vars' parameter in the `tcplVarMat` function. The 'add.vars' function will take any level 4 or level 5 field and create the respective matrix.