cxcalc command line tool

This manual gives a walk-through on how to use cxcalc command line tool:

Introduction

cxcalc is the command line version of ChemAxon's Calculator Plugins.

General Usage

Invoking cxcalc

cxcalc performs plugin calculations in a uniform way: it processes general input, output parameters and SDF file tag names and also plugin specific parameters that are different for each plugin. It can also be used to train some of the calculators.

The command has the following syntax:

```
cxcalc [general options] [input file(s)/string(s)] <plugin> [plugin option(s)] [input file(s)/string(s)]
```

or

```
cxcalc [training options] [input file (the training set)]
```

Two or more plugins can also be invoked with its own parameters within one command. In this case calculations are run in the order of invocation.

Input and output of cxcalc

cxcalc takes molecules from text files or from SMILES strings. Most molecular file formats are accepted (e.g. SMILES, SDF). If no input file name or SMILES string is given in the command line, input molecules are read from the standard input.

cxcalc writes calculation results in a format that is based on the specified tags. If the result refers to the entire molecule, it is written as a single number. If the calculation gives a separate number for each atom in the molecule, it is written as a list of numbers separated by semicolons. The order of the results corresponds to the order of the atoms determined by their atom indices. Other output formats may be available for certain plugins, see the plugin specific options for the plugin. By default, results are written without the input molecule in a table form, but they can also written in an SDF file as an SDF tag by adding *--sdf-output*.

Options of cxcalc

General options

The following general (that can be used for every calculation function) options are available:

```
cxcalc -h, --help                 this help message,
                                    list of available calculations
  cxcalc <plugin> -h, --help      plugin specific help message
  -o, --output <filepath>         output file path (default:
standard output)
  -t, --tag <tag name>            name of the SDFile tag to store the
                                  calculation results, tag name
prefix
                                  to default tag names in case of
multiple
                                  plugins (default: see plugin help)
  -i, --id <tag name|format>      the name of the existing SDFile
tag that
                                  stores the molecule ID; or create
```

```
                                    molecule ID by converting the input
                                    molecule into the specified format;
                                    (default: molecule index is used
as ID)
  -N, --do-not-display <type>       [i|h|ih]
                                    do not display molecule ID and/or
                                    table header (in table output form)
         i                          no molecule ID
         h                          no table header
         ih                         neither molecule ID nor table
header
  -S, --sdf-output                  SDF output with results in SDF tags
  -M, --mrv-output                  result molecule output in MRV
format
                                    (if neither -S nor -M is
specified, then
                                    plugin results are written in
table form)
  -g, --ignore-error                continue with next molecule on
error
  -v, --verbose                     print calculation warnings to the
console
      --log <filepath>              write log messages to file
                                    (default: write log to system
error)
      --log-level <level>           [error|warning|off]
                                    set log level (default: error)
         error                      log error level information
         warning                    log warning and error level
information
         off                        no log information
      --log-options <options>       list of logger options, separated
by ','
         time                       log calculation execution time;
calculation
                                    will run on ONE CPU in this case
         timelimit=<time in ms>     only execution times above the
specified
                                    limit will be logged
         format=<molecule format>   log file format; default is SDF
when
                                    logging to file and SMILES when
logging to
                                    system error
```

You can also pass some JVM options to the Java Virtual Machine as command line arguments.

Input files can be given both on the general option side and on the plugin specific option side, in both cases these input files/strings give the input molecules for the calculations. If more plugins are given, then all calculations are performed for the input molecules.

Plugin IDs are not case-sensitive, you can alter upper- and lower case letters if you like, e. g.

- `cxcalc logp in.mol = cxcalc logP in.mol`
- `cxcalc totalchargedensity in.mol = cxcalc totalChargeDensity in.mol`

The syntax of commands can be different under various command line shells, e.g. bash, tcsh, zsh.

Some notes on the command line parameters:

- The command line parameter `--tag` specifies the SDF file tag name to be used when storing the calculation results in an SDF file.
- If the `--do-not-display` parameter is specified, then no molecule ID and/or table header is displayed. This option has no effect in `--sdf-output` and `--mrv-output` output modes.
- If the `--sdf-output` parameter is given, then the input molecules are written in SDF format and the calculation results are added in SDF file tags.
- If the `--mrv-output` parameter is given, then the decorated result molecules are written in MRV format. Atomic results are written into atom labels, while molecular results are displayed as molecule properties. Not every calculation has this decorated molecule output.
- If neither `--sdf-output` nor `--mrv-output` are specified, then the calculation results are shown in text table form. The display of table header and molecule ID column can be altered in the `--do-not-display` option.
- The parameter `--id` parameter specifies the input SDF file tag that stores the molecule ID to be written in the output table. This parameter is only used if the output is in text table form (neither `--sdf-output` nor `--mrv-output` is specified). By default the input file index of the molecule is used as molecule ID. Molecule formats can be specified with output options, for the available formats and options see the File Formats in Marvin manual. Probably the most frequently used format replacing a molecule ID is SMILES, possibly with the *a-H* (aromatize, remove explicit hydrogens) option (*smiles:a-H*) to generate canonical strings.

Plugin-specific options

The plugin specific help message is printed if the user types:

```
cxcalc <plugin> -h
```

Here the second option is the plugin key from the configuration file, e.g. typing

```
cxcalc logp -h
```

gives the following output:

```
Calculator plugin: logp.

logP calculation:
for type logPTrue: logP of uncharged species, or,
in the case of zwitterions, logD at pI;
for type logPMicro: logP of the input species.

Usage:
  cxcalc [general options] [input files/strings] logp
[logp options] [input files/strings]

logp options:

 -h, --help                     this help message
 -p, --precision                <floating point precision as number of
                                fractional digits: 0-8 or inf>
(default: 2)
 -m, --method                   [vg|klop|phys|user|weighted]
                                (default: weighted)
     --trainingid               <training id>
 -w, --weights                  <wVG:wKLOP:wPHYS:wUSER> method weights
                                (default: 1:1:1:0)
                                wVG: weight of the VG method
                                wKLOP: weight of the KLOP method
                                wPHYS: weight of the PHYS method
                                wUSER: weight of the user defined
method
 -a, --anion                    <Cl- concentration>
                                (default: 0.1, range: [0.0, 0.25])
 -k, --kation                   <Na+ K+ concentration>
                                (default: 0.1, range: [0.0, 0.25])
 -t, --type                     [increments|logPMicro|logPTrue]
                                (default: logPTrue)
 -i, --increments               [true|false] show atomic increments
                                (default: false)
     --considertautomerization  [true|false] consider tautomerization
                                (default: false)
Multiple values for the same parameter
should be separated by commas (',' without space).

Example:
  cxcalc -S -t myLOGP logp -a 0.15 -k 0.05 test.mol
```

Some notes on plugin-specific options:

- The command line parameter `--precision` specifies the required floating point precision, that is, the number of required decimal digits in the output.
- The command line parameter `--type` specifies the result type: `increments` gives the atomic logp increment values for each atom in the molecule while `logPTrue` gives the overall logp value for the molecule. Both results can be queried by specifying both types separated by a comma: `-t increments,implh,logPTrue`.
- The command line parameter `--majortautomer` specifies if major tautomeric form of the input molecule should be taken as input for the logP calculation. In cases when an option takes `[true|false]` parameter values the `true` parameter value can be omitted: `cxcalc logp --majortautomer true mols.sdf` and `cxcalc logp --majortautomer mols.sdf` commands produce the same results.

cxcalc calculator functions

You can find the full list of available calculator functions here.

Configuring cxcalc

It is possible to configure cxcalc via a configuration file, which is a JAVA property file.

An example configuration file below shows its format:

```
charge=$chemaxon.marvin.calculations.ChargePlugin\
        $ChargePlugin.jar\
        $Charge\
        $p=precision:2;t=type:total;i=implh:false;r=resonance:false;
H=pH\
        $CHARGE\
        $Partial charge calculation.\nTypes aromaticsystem /
aromaticring calculate the sum of charges\nin the aromatic system /
aromatic ring containing the atom.\
        $-p, --precision=<floating point precision as number of
        \nfractional digits: 0-8 or inf>(default: 2);-t, --type=
[sigma|pi|total|implh|\naromaticsystem|aromaticsystemsigma|aromaticsyste
```

The key *charge* is the plugin name which refers to the calculation in cxcalc.

Configuration items are separated by '$' characters. The '\' characters allow property values to be expanded to multiple lines: the '\' character itself as well as leading white spaces in the next line are ignored.

The configuration items are as follows:

1. the plugin class with full package name
2. the plugin JAR name (with path relative to the `plugins` directory)
3. the plugin group name (used for grouping the available plugins in the help message)
4. the plugin specific parameters:
   `<short name>=<long name>:<default value>`

   separated by semicolons
5. the default SDF file tag name storing the results in case of SDF file output
6. a short description used in the plugin specific help message
7. the plugin specific help text (parameter description text) with newline characters replaced by semicolons
8. an example usage text (optional)

The plugin loading mechanism is the following: first the program tries to load the plugin class by the default class loader from the CLASSPATH. If this the plugin class is not found, then the JAR is loaded and the system tries to load the plugin class from there. If the plugin name is omitted, the plugin is loaded directly from the JAR where the `Plugin-Class` manifest attribute specifies the plugin class. If the JAR name is omitted, then the plugin is loaded from the CLASSPATH.

Missing configuration items should be denoted by '-' characters. For example, here is the plugin configuration from above with omitted JAR name:

```
charge=$chemaxon.marvin.calculations.ChargePlugin\
        $-\
        $Charge\
        $p=precision:2;t=type:total;i=implh:false;H=pH\
        $CHARGE\
        $Partial charge calculation.\nTypes aromaticsystem /
aromaticring calculate the sum of charges\nin the aromatic system /
aromatic ring con        taining the atom.\
        $-p, --precision=<floating point precision as number of
        \nfractional digits: 0-8 or inf> (default: 2);-t, --type=
[sigma|pi|total|implh|aromaticsystem|aromaticring]
        \n(default: total);-i, --implh=[true|false] implicit H charge
sum shown in brackets
        \n(for sigma and total charge only) (default: false);-H, --
pH=<pH value> takes physiological microspecies at this pH
        \n(default: no pH, takes the input molecule)\
        $cxcalc -S -o result.sdf -t myCHARGE charge -t pi,total -p 3
test.mol
```

Long parameter names in the *plugin specific parameters* section should correspond to the parameter property keys used in the plugin class in the `setParameters(Properties params)` method.

Examples

Here we give some examples on how to use cxcalc:

1. pK_a calculation with table form output, showing the two most significant acidic and the two most significant basic pK_a values (this is the default table output mode):

   ```
   cxcalc mols.sdf pka
   ```

2. The same with molecule ID-s taken from the `ID` tag of the input SDF file, writing three significant values from each pK_a type:

   ```
   cxcalc mols.sdf -i ID pka -a 3 -b 3
   ```

3. The same with setting minimum basic pK_a to `-5`, maximum acidic pK_a to 15:

   ```
   cxcalc mols.sdf -i ID pka -a 3 -b 3 -i -5 -x 15
   ```

4. Charge calculation for molecules in the `mols.sdf` file, writes results to the standard output in MRV format, charge values displayed in atom labels:

```
cxcalc -M charge mols.sdf
```

5. The same with output to the `molcharges.mrv` file to be created in the same directory, displaying the results in MarvinView:

```
cxcalc -M -o molcharges.mrv charge mols.sdf

mview molcharges.mrv
```

6. Log*P* calculation with both result types (atomic increments and overal molecule) and user defined SDF tag name, piping the result to MarvinView:

```
cxcalc -S mols.sdf -t LOGP_BOTH logp -t increments,logP | mview -
```

> Such piping does not work on Windows OS.

   By setting the `Table/Show Fields` option in MarvinView the SDF file tags will be shown in the table cells and in this way the charge values can be seen.

7. Elemental analysis (all result types), output in table form, molecule ID-s taken from the `ID` tag of the input SDF file, output written to text file `elemanal.txt`:

```
cxcalc -o elemanal.txt -i ID elemanal mols.sdf
```

8. A similar example with input taken from `mols.smiles` and output written as SDF to `elemanal.sdf` with `ELEMANAL` tag name:

```
cxcalc -S -t ELEMANAL -o elemanal.sdf elemanal mols.smiles
```

9. Writting molecular mass, log*P* and log*D* at pH `6.4` in the same table:

```
cxcalc mass logP logD -H 6.4 mols.smiles
```

10. Calculating some topological data:

```
cxcalc ringCount ringAtomCount ringBondCount mols.smiles
```