

The Mychem² Handbook

Jérôme PANSANEL <j.pansanel@pansanel.net>

The Mychem² Handbook

by Jérôme PANSANEL

0.6.0

Copyright © 2007, 2008 Jerome Pansanel

Abstract

The Mychem² Handbook is a documentation for the Mychem² software. This software provides a set of functions for handling chemical data within MySQL. Mychem² is based on Open Babel version 2.1.1.

Redistribution and use in source (XML DocBook) and 'compiled' forms (SGML, HTML, PDF, PostScript, RTF and so forth) with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code (SGML DocBook) must retain the above copyright notice, this list of conditions and the following disclaimer as the first lines of this file unmodified.
2. Redistributions in compiled form (transformed to other DTDs, converted to PDF, PostScript, RTF and other formats) must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

This documentation is provided "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the Mychem documentation be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this documentation, even if advised of the possibility of such damage.

Table of Contents

1. Introduction	1
Mychem ²	1
MySQL	1
Open Babel	1
2. Using Mychem	2
The Database	2
Creating the Database	2
Inserting Data	3
Examples	4
Calculating the Molecular Weight	4
Generating SMILES String	4
3. Command Reference	5
The <i>molecule</i> Type	5
Conversion Commands	5
Helper Commands	10
Modification Commands	11
Molmatch Commands	12
Property Commands	14
4. Credits and License	19
Contributions	19
Developers	19
Contributors	19
Copyright	19
License	19
A. Installation	21
How to Obtain Mychem	21
Source Package	21
SVN Trunk	21
Requirements	21
Compilation and Installation	22
Installing Mychem on GNU/Linux	22
Installing Mychem on Mac OS X	24
Installing Mychem on Microsoft Windows	25
Mychem API	27
B. Troubleshooting	28
MySQL-Related Errors	28
ERROR 2013 (HY000): Lost connection to MySQL server during query	28
Other Errors	28
C. MychemAdmin	29
Introduction	29
Installation	29
Requirements	29
Compilation and Installation	29
Using MychemAdmin	29
The Main Window	30
Database Settings	30
File Loading	31
D. SDF File Loading Using Python	32

Chapter 1. Introduction

Mychem²

Mychem² is a chemoinformatics extension for MySQL. This extension provides a set of functions that handles chemical data within the MySQL database. Mychem² is based on Open Babel v2.1.1, a well-known software in chemoinformatics. Therefore, Mychem proposes reliable and fast functions. These functions permit you to search, analyze and convert chemical data. More informations on Mychem are available on the Mychem website [<http://mychem.sourceforge.net/>].

MySQL

MySQL is one of the most popular Open Source SQL database server. It's a very fast, multi-threaded multi-user and robust application. MySQL Server is intended for mission-critical, heavy-load production systems as well as for embedding into mass-deployed software. Further informations about MySQL can be founded on the MySQL website [<http://www.mysql.com/>].

Open Babel

Open Babel is a chemical toolbox designed to speak the many languages of chemical data. It's an open, collaborative project allowing anyone to search, convert, analyze, or store data from molecular modeling, chemistry, solid-state materials, biochemistry, or related areas. You can find more informations about Open Babel on the Open Babel website [http://openbabel.sourceforge.net/wiki/Main_Page].

Chapter 2. Using Mychem

This chapter provides a short tutorial on the usage of Mychem. It will present you a simple way to create a chemical database with Mychem and how to use some functions. More details about each function used in this tutorial are available in the Chapter 3, *Command Reference*.

The Database

A chemical database is composed of one or several tables. The following example use a set of four tables, however the use of one table can be sufficient when working with small databases.

- *compound* - a table containing an unique id for each molecule and their name.
- *compound_inchi* - a table containing an unique reference to *compound* table and the InChI code.
- *compound_mol* - a table containing an unique reference to *compound* table and the structure in MDL Molfile format.
- *compound_fp* - a table containing an unique reference to the *compound* table and the fingerprint.

You can use the MychemAdmin software to create this set of tables and load MDL SDfiles. More details about MychemAdmin are given in the Appendix C, *MychemAdmin*. You can also create the database by using the instructions given in the next section.

Creating the Database

The structure of the database can be created with the following SQL code:

```
--
-- The 'mychem' database name is provided as example. You can
-- choose any other existing database.
--

USE mychem;

--
-- The `compound` table is used to store an unique id for each
-- compound. It can also contains some other data, like an unique
-- name, as used in your company.
--

CREATE TABLE `compound` (
  `id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(64) CHARACTER SET utf8
  COLLATE utf8_unicode_ci NOT NULL,
  UNIQUE (`id`)
) ENGINE=INNODB DEFAULT COMMENT='Compound Library';

--
-- The `compound_inchi` table is used to store the InChI code.
-- This table is used for exact match searching.
--

CREATE TABLE `compound_inchi` (
  `compound_id` INT(11) UNSIGNED NOT NULL,
  `inchi` TEXT CHARACTER SET utf8
```

```
    COLLATE utf8_unicode_ci NOT NULL,
    UNIQUE (`compound_id`),
    CONSTRAINT `compound_inchi_ibfk_1` FOREIGN KEY (`compound_id`)
    REFERENCES `compound` (`id`) ON DELETE CASCADE
    ON UPDATE NO ACTION
) ENGINE=INNODB DEFAULT COMMENT='InChI Library';

--
-- `compound_mol` is used to store the Molfiles.
--

CREATE TABLE `compound_mol` (
  `compound_id` INT(11) UNSIGNED NOT NULL,
  `mol` TEXT CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL,
  UNIQUE (`compound_id`),
  CONSTRAINT `compound_mol_ibfk_1` FOREIGN KEY (`compound_id`)
  REFERENCES `compound` (`id`) ON DELETE CASCADE
  ON UPDATE NO ACTION
) ENGINE=INNODB DEFAULT COMMENT='MDL Molfile Library';

--
-- `compound_fp` is used to store the fingerprint. This table is
-- used for similarity searching.
--

CREATE TABLE `compound_fp` (
  `compound_id` INT(11) UNSIGNED NOT NULL,
  `fp` TEXT CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL,
  UNIQUE (`compound_id`),
  CONSTRAINT `compound_fp_ibfk_1` FOREIGN KEY (`compound_id`)
  REFERENCES `compound` (`id`) ON DELETE CASCADE
  ON UPDATE NO ACTION
) ENGINE=INNODB DEFAULT COMMENT='Fingerprint Library';
```

Of course, additional tables can also be created. For example, a table storing chemical properties (molecular weight, LogP, ...) or SMILES string can also be interesting.

Inserting Data

The molecular structures contained in a MDL SDF file can be easily loaded into the database by using a Python or Perl script. To facilitate this step, an example script is listed in Appendix D, *SDF File Loading Using Python*. Once the *compound* and *compound_mol* tables are filled, you can populate the two remaining tables. To insert data into the *compound_inchi* and *compound_fp* tables, use the following SQL code:

```
--
-- Database: mychem
--

USE mychem;

--
-- Insertion of data into the `compound_inchi` table
--

INSERT INTO `compound_inchi` (`compound_id`,`inchi`)
```

```
SELECT `compound_mol`.`compound_id`,
molecule_to_inchi(`compound_mol`.`mol`)
FROM `compound_mol`

--
-- Insertion of data into the `compound_fp` table
--

INSERT INTO `compound_fp` (`compound_id`,`fp`)
SELECT `compound_mol`.`compound_id`,
fingerprint2(`compound_mol`.`mol`)
FROM `compound`,`compound_mol`
```

The database is now fully usable.

Examples

Once the database is created, you can use many chemical functions. Here's a (very) short overview of the possibilities.

Calculating the Molecular Weight

The computation of the molecular weight of a molecule is performed by the MOLWEIGHT() function. In the following example, the molecular weight of the amino acid *glycine* is calculated.

```
mysql> SELECT MOLWEIGHT(`compound_mol`.`mol`)
-> FROM `compound`,`compound_mol`
-> WHERE `compound`.`name`='glycine'
-> AND `compound`.`id`=`compound_mol`.`compound_id`;
-> 75.066600
```

Generating SMILES String

To generate the SMILES string of the amino acid *glycine*, use the MOLECULE_TO_SMILES() function:

```
mysql> SELECT MOLECULE_TO_SMILES(`compound_mol`.`mol`)
-> FROM `compound`,`compound_mol`
-> WHERE `compound`.`name`='glycine'
-> AND `compound`.`id`=`compound_mol`.`compound_id`;
-> C(N)C(=O)O
```

Chapter 3. Command Reference

This chapter describes all MySQL commands provided by Mychem. These commands are classified in several categories:

- Conversion Commands - details functions that convert chemical files. These functions are provided by the *conversion* module.
- Helper Commands - details functions that return general informations. These functions are provided by the *helper* module.
- Modification Commands - details functions that modify chemical data. These functions are provided by the *modification* module.
- Molmatch Commands - details functions that compare chemical structures. These functions are provided by the *molmatch* module.
- Property Commands - details functions that compute molecular properties. These functions are provided by the *property* module.

The *molecule* Type

Many functions are working with a variable called *molecule*. It can be either an argument or a return value. Since version 0.6.0 of Mychem, the format of this argument is MDL Molfile. In earlier versions, the format was InChI.

The speed of several functions has been measured when using the InChI and MDL Molfile formats. It has been shown that using MDL Molfile is a bit faster than InChI. Using the MDL Molfile format is also interesting, as it permits to store 2D and 3D coordinates. Moreover, the MDL Molfile format is commonly used by many softwares.

Conversion Commands

The *conversion* module provides functions for converting chemical data. Open Babel supports over 80 chemical file formats. To avoid having to many functions, this module defines a set of two functions for each format: `FORMAT_TO_MOLECULE` and `MOLECULE_TO_FORMAT`. However, new functions, like `CML_TO_SMILES`, can be requested on the feature tracker [http://sourceforge.net/tracker/?group_id=195099&atid=952084].

Note

If a function of the *conversion* module fails, it returns an empty string.

- `CML_TO_MOLECULE(cml)`

`CML_TO_MOLECULE()` converts a CML file to a *molecule*.

```
mysql> SELECT CML_TO_MOLECULE(cml_col) FROM tbl_name
-> WHERE name='glycine';
-> glycine
OpenBabel111190809032D

5 4 0 0 0 0 0 0 0 0 0999 V2000
206.0000 126.0000 0.0000 C 0 0 0 0 0
227.0075 113.8713 0.0000 C 0 0 0 0 0
227.0075 89.6139 0.0000 O 0 0 0 0 0
```

```

248.0151 126.0000 0.0000 O 0 0 0 0 0
184.9925 113.8713 0.0000 N 0 0 0 0 0
1 2 1 0 0 0
1 5 1 0 0 0
2 3 2 0 0 0
2 4 1 0 0 0
M END

```

- **MOLECULE_TO_CML(*molecule*)**

MOLECULE_TO_CML() converts a *molecule* to a CML file.

```

mysql> SELECT MOLECULE_TO_CML(molecule_col) FROM tbl_name
      -> WHERE name='glycine';
      -> <molecule id="glycine">
<atomArray>
  <atom id="a1" elementType="C" x2="206.000000" y2="126.000000"/>
  <atom id="a2" elementType="C" x2="227.007500" y2="113.871300"/>
  <atom id="a3" elementType="O" x2="227.007500" y2="89.613900"/>
  <atom id="a4" elementType="O" x2="248.015100" y2="126.000000"/>
  <atom id="a5" elementType="N" x2="184.992500" y2="113.871300"/>
</atomArray>
<bondArray>
  <bond atomRefs2="a2 a3" order="2"/>
  <bond atomRefs2="a2 a4" order="1"/>
  <bond atomRefs2="a1 a2" order="1"/>
  <bond atomRefs2="a1 a5" order="1"/>
</bondArray>
</molecule>

```

- **FINGERPRINT(*molecule,type*)**

FINGERPRINT() converts a *molecule* to a fingerprint (binary string). The second argument determines the type of fingerprint (FP2, FP3 or FP4).

```

mysql> SELECT FINGERPRINT(molecule_col,"FP2") FROM tbl_name
      -> WHERE name='glycine';
      -> binary fingerprint (type FP2)

```

Note

Tanimoto scoring can be improved by using a concatenation of different fingerprint types. The concatenation of fingerprints can be performed with the following query:

```

mysql> SELECT CONCAT(FINGERPRINT(molecule_col,"FP2"),
      -> FINGERPRINT(molecule_col,"FP3")) FROM tbl_name
      -> WHERE id=9;
      -> binary fingerprint (type FP2 + type FP3)

```

- **FINGERPRINT2(*molecule*)**

FINGERPRINT2() converts a *molecule* to a FP2 fingerprint (binary string). This type of fingerprint is based on linear fragments up to 7 atoms. The return value is a binary string of 128 bytes.

```
mysql> SELECT FINGERPRINT2(molecule_col,"FP2") FROM tbl_name
-> WHERE name='glycine';
-> binary fingerprint (type FP2)
```

- **FINGERPRINT3(*molecule*)**

FINGERPRINT3() converts a *molecule* to a FP3 fingerprint (binary string). FP3 is a fingerprint method created from a set of SMARTS patterns defining functional groups (see the `patterns.txt` file in the Open Babel directory).

```
mysql> SELECT FINGERPRINT3(molecule_col) FROM tbl_name
-> WHERE name='glycine';
-> binary fingerprint (type FP3)
```

- **FINGERPRINT4(*molecule*)**

FINGERPRINT4() converts a *molecule* to a FP4 fingerprint (binary string). FP4 is a fingerprint method created from a set of SMARTS patterns defining functional groups (see the `SMARTS_InteLigand.txt` file in the Open Babel directory).

```
mysql> SELECT FINGERPRINT4(molecule_col) FROM tbl_name
-> WHERE name='glycine';
-> binary fingerprint (type FP4)
```

- **INCHI_TO_MOLECULE(*inchi*)**

INCHI_TO_MOLECULE() converts an InChI to a *molecule*.

```
mysql> SELECT INCHI_TO_MOLECULE(inchi_col) FROM tbl_name
-> WHERE name='glycine';
->
OpenBabel111190809142D

  5  4  0  0  0  0  0  0  0  0  0999 V2000
    0.0000  0.0000  0.0000 C  0  0  0  0  0
    0.0000  0.0000  0.0000 C  0  0  0  0  0
    0.0000  0.0000  0.0000 O  0  0  0  0  0
    0.0000  0.0000  0.0000 O  0  0  0  0  0
    0.0000  0.0000  0.0000 N  0  0  0  0  0
  1  2  1  0  0  0
  1  5  1  0  0  0
  2  3  2  0  0  0
  2  4  1  0  0  0
M  END
```

- **MOLECULE_TO_INCHI(*molecule*)**

MOLECULE_TO_INCHI converts a *molecule* to an InChI.

```
mysql> SELECT MOLECULE_TO_INCHI(molecule_col) FROM tbl_name
```

```
-> WHERE name='glycine';
-> InChI=1/C2H5NO2/c3-1-2(4)5/h1,3H2,(H,4,5)/f/h4H
```

- **MOLECULE_TO_MOLECULE(*molecule*)**

MOLECULE_TO_MOLECULE() converts data from the old *molecule* format (InChI) to the new *molecule* format (MDL Molfile). This function is useful when updating a database with a new version of Mychem.

```
mysql> SELECT MOLECULE_TO_MOLECULE(molecule_col) FROM tbl_name
-> WHERE name='glycine';
->
OpenBabel111190809092D

5 4 0 0 0 0 0 0 0 0999 V2000
0.0000 0.0000 0.0000 C 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0
0.0000 0.0000 0.0000 N 0 0 0 0 0
0.0000 0.0000 0.0000 O 0 0 0 0 0
0.0000 0.0000 0.0000 O 0 0 0 0 0
1 2 1 0 0 0
1 3 1 0 0 0
2 4 2 0 0 0
2 5 1 0 0 0
M END
```

- **MOLFILE_TO_MOLECULE(*molfile*)**

MOLFILE_TO_MOLECULE() converts a MDL Molfile (V2000) to a *molecule*.

```
mysql> SELECT MOLFILE_TO_MOLECULE(molfile_col) FROM tbl_name
-> WHERE name='glycine';
-> glycine
OpenBabel111190809052D

5 4 0 0 0 0 0 0 0 0999 V2000
206.0000 126.0000 0.0000 C 0 0 0 0 0
227.0075 113.8713 0.0000 C 0 0 0 0 0
227.0075 89.6139 0.0000 O 0 0 0 0 0
248.0151 126.0000 0.0000 O 0 0 0 0 0
184.9925 113.8713 0.0000 N 0 0 0 0 0
1 2 1 0 0 0
1 5 1 0 0 0
2 3 2 0 0 0
2 4 1 0 0 0
M END
```

- **MOLECULE_TO_MOLFILE(*molecule*)**

MOLECULE_TO_MOLFILE() converts a *molecule* to a MDL Molfile (V2000).

```
mysql> SELECT MOLECULE_TO_MOLFILE(molecule_col) FROM tbl_name
-> WHERE name='glycine';
-> glycine
```

```

OpenBabel111190809062D

  5  4  0  0  0  0  0  0  0  0  0999 V2000
 206.0000 126.0000  0.0000 C  0  0  0  0  0
 227.0075 113.8713  0.0000 C  0  0  0  0  0
 227.0075  89.6139  0.0000 O  0  0  0  0  0
 248.0151 126.0000  0.0000 O  0  0  0  0  0
 184.9925 113.8713  0.0000 N  0  0  0  0  0
  1  2  1  0  0  0
  1  5  1  0  0  0
  2  3  2  0  0  0
  2  4  1  0  0  0
M  END

```

- **SMILES_TO_MOLECULE(*smiles*)**

SMILES_TO_MOLECULE() converts a SMILES string to a *molecule*.

```

mysql> SELECT SMILES_TO_MOLECULE(smiles_col) FROM tbl_name
      -> WHERE name='glycine';
      ->
OpenBabel111190809142D

```

```

  5  4  0  0  0  0  0  0  0  0  0999 V2000
   0.0000  0.0000  0.0000 C  0  0  0  0  0
   0.0000  0.0000  0.0000 C  0  0  0  0  0
   0.0000  0.0000  0.0000 O  0  0  0  0  0
   0.0000  0.0000  0.0000 O  0  0  0  0  0
   0.0000  0.0000  0.0000 N  0  0  0  0  0
  1  2  1  0  0  0
  1  5  1  0  0  0
  2  3  2  0  0  0
  2  4  1  0  0  0
M  END

```

- **MOLECULE_TO_SMILES(*molecule*)**

MOLECULE_TO_SMILES converts a *molecule* to a SMILES string.

```

mysql> SELECT MOLECULE_TO_SMILES(molecule_col) FROM tbl_name
      -> WHERE name='glycine';
      -> C(C(=O)O)N

```

Note

If you need a single canonical form for any particular molecule, regardless of atom order, the use of the **MOLECULE_TO_CANONICAL_SMILES()** function should be preferred.

- **MOLECULE_TO_CANONICAL_SMILES(*molecule*)**

MOLECULE_TO_CANONICAL_SMILES converts a *molecule* to a canonical SMILES string.

```

mysql> SELECT MOLECULE_TO_CANONICAL_SMILES(molecule_col)
      -> FROM tbl_name WHERE name='glycine';

```

```
-> NCC(=O)O
```

- **V3000_TO_MOLECULE(v3000)**

V3000_TO_MOLECULE() converts a MDL Molfile (V3000) to a *molecule*.

```
mysql> SELECT V3000_TO_MOLECULE(V3000_col) FROM tbl_name
-> WHERE name='glycine';
-> glycine
OpenBabel111190809082D

  5  4  0  0  0  0  0  0  0  0  0999 V2000
206.0000 126.0000  0.0000 C  0  0  0  0  0
227.0075 113.8713  0.0000 C  0  0  0  0  0
227.0075  89.6139  0.0000 O  0  0  0  0  0
248.0151 126.0000  0.0000 O  0  0  0  0  0
184.9925 113.8713  0.0000 N  0  0  0  0  0
  1  2  1  0  0  0
  1  5  1  0  0  0
  2  3  2  0  0  0
  2  4  1  0  0  0
M  END
```

- **MOLECULE_TO_V3000(molecule)**

MOLECULE_TO_V3000() converts a *molecule* to a MDL Molfile (V3000).

```
mysql> SELECT MOLECULE_TO_V3000(molecule_col) FROM tbl_name
-> WHERE name='glycine';
-> glycine
OpenBabel111190809132D

  0  0  0  0  0  999 V3000
M  V30 BEGIN CTAB
M  V30 COUNTS 5 4 0 0 0
M  V30 BEGIN ATOM
M  V30 1 C 206 126 0 0
M  V30 2 C 227.007 113.871 0 0
M  V30 3 O 227.007 89.6139 0 0
M  V30 4 O 248.015 126 0 0
M  V30 5 N 184.993 113.871 0 0
M  V30 END ATOM
M  V30 BEGIN BOND
M  V30 1 1 1 2
M  V30 2 1 1 5
M  V30 3 2 2 3
M  V30 4 1 2 4
M  V30 END BOND
M  V30 END CTAB
M  END
```

Helper Commands

The *helper* module provides functions that return the version of Mychem and Open Babel.

- **MYCHEM_VERSION()**

MYCHEM_VERSION() returns the Mychem version.

```
mysql> SELECT MYCHEM_VERSION();
      -> 0.6.0
```

- **OPENBABEL_VERSION()**

OPENBABEL_VERSION() returns the Open Babel version.

```
mysql> SELECT OPENBABEL_VERSION();
      -> 2.1.1
```

Modification Commands

The *modification* module provides functions that modify the chemical structure. This module is fully working starting on version 0.6.0 of Mychem.

Note

If a function of the *modification* module fails, it returns an empty string.

- **ADD_HYDROGENS(*molecule*)**

ADD_HYDROGENS() adds hydrogens to a *molecule* (makes explicit the hydrogen atoms).

```
mysql> SELECT ADD_HYDROGENS(molecule_col) FROM tbl_name
      -> WHERE name='glycine';
      -> glycine
      OpenBabel111190809242D
```

```
10  9  0  0  0  0  0  0  0  0  0999 V2000
 206.0000 126.0000  0.0000 C  0  0  0  0  0
 227.0075 113.8713  0.0000 C  0  0  0  0  0
 227.0075  89.6139  0.0000 O  0  0  0  0  0
 248.0151 126.0000  0.0000 O  0  0  0  0  0
 184.9925 113.8713  0.0000 N  0  0  0  0  0
 206.0000 126.7208 -0.8832 H  0  0  0  0  0
 206.0000 126.4888  1.0299 H  0  0  0  0  0
 248.9361 125.4682  0.0000 H  0  0  0  0  0
 184.8694 113.3367 -0.9765 H  0  0  0  0  0
 184.1380 114.5759  0.1668 H  0  0  0  0  0
 1  2  1  0  0  0
 1  5  1  0  0  0
 1  6  1  0  0  0
 1  7  1  0  0  0
 2  3  2  0  0  0
 2  4  1  0  0  0
 4  8  1  0  0  0
 5  9  1  0  0  0
 5 10  1  0  0  0
M  END
```

- **REMOVE_HYDROGENS(*molecule*)**

REMOVE_HYDROGENS() removes the hydrogens to a *molecule* (makes implicit the hydrogen atoms).

```
mysql> SELECT REMOVE_HYDROGENS(molecule_col) FROM tbl_name
-> WHERE name='glycine';
-> glycine
OpenBabel11190809252D

5 4 0 0 0 0 0 0 0 0 0999 V2000
206.0000 126.0000 0.0000 C 0 0 0 0 0
227.0075 113.8713 0.0000 C 0 0 0 0 0
227.0075 89.6139 0.0000 O 0 0 0 0 0
248.0151 126.0000 0.0000 O 0 0 0 0 0
184.9925 113.8713 0.0000 N 0 0 0 0 0
1 2 1 0 0 0
1 5 1 0 0 0
2 3 2 0 0 0
2 4 1 0 0 0
M END
```

- **STRIP_SALTS(*molecule*)**

STRIP_SALTS() removes all atoms except for the larger contiguous fragment.

```
mysql> SELECT STRIP_SALTS(molecule_col) FROM tbl_name
-> WHERE name='sodium 2-aminoacetate';
->
OpenBabel11190812342D

5 4 0 0 0 0 0 0 0 0999 V2000
0.0000 0.0000 0.0000 N 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0
0.0000 0.0000 0.0000 C 0 0 0 0 0
0.0000 0.0000 0.0000 O 0 0 0 0 0
0.0000 0.0000 0.0000 O 0 0 0 0 0
1 2 1 0 0 0
2 3 1 0 0 0
3 4 2 0 0 0
3 5 1 0 0 0
M CHG 1 5 -1
M END
```

Molmatch Commands

The *molmatch* module provides functions to compare chemical structures.

- **BIT_FP_AND(*fingerprint,fingerprint*)**

BIT_FP_AND() operates on two fingerprints (bit patterns) of equal length and performs the logical AND operation on each pair of corresponding bits. In each pair, the result is 1 if the both bits are 1.

Otherwise, the result is 0. If the two fingerprints do not have the same length, the function returns NULL.

```
mysql> SELECT BIT_FP_AND(fingerprint1,fingerprint2);
      -> binary fingerprint
```

Note

The BIT_FP_AND() function is very useful when working with structure fingerprints. For example, if a molecule (with a fingerprint *fp1*) is a substructure of an other molecule (with a fingerprint *fp2*), the following property is observed:

```
mysql> SELECT TANIMOTO(BIT_FP_AND(fp1,fp2),fp1);
      -> 1
```

- **BIT_FP_COUNT(*fingerprint*)**

BIT_FP_COUNT() returns the number of bits that are set in the *fingerprint* binary representation.

```
mysql> SELECT BIT_FP_COUNT(fp_col) FROM tbl_name
      -> WHERE name='1H-indole';
      -> 23
```

- **BIT_FP_OR(*fingerprint*,*fingerprint*)**

BIT_FP_OR() operates on two fingerprints (bit patterns) of equal length and performs the logical OR operation on each pair of corresponding bits. In each pair, if the first bit is 1 or the second bit is 1 (or both), the result is 1. Otherwise, the result is 0. If the two fingerprints do not have same length, the function returns NULL.

```
mysql> SELECT BIT_FP_OR(fingerprint1,fingerprint2);
      -> binary fingerprint
```

- **MATCH_SUBSTRUCT(*query_smarts*,*reference_smiles*)**

MATCH_SUBSTRUCT() checks if a *query_smarts* fragment is a substructure of a *reference_smiles* molecule. The first argument is a SMARTS string, whereas the second argument is a SMILES string. If the *query_smarts* is a substructure of *reference_smiles*, the function returns 1, otherwise, it returns 0.

```
mysql> SELECT MATCH_SUBSTRUCT('C=O', 'C(N)C(=O)O');
      -> 1
```

- **SUBSTRUCT_ATOM_IDS(*query_smarts*,*reference_smiles*)**

SUBSTRUCT_ATOM_IDS() returns the atom ids of a *reference_smiles* molecule that are contained in substructures matching a *query_smarts* fragment. The first argument is a SMARTS string, whereas the second argument is a SMILES strings. If a *reference_smiles* molecule contains several fragments matching a *query_smarts* fragment, a list of items is returned. Each item contains a fragment's atom ids and is separated from the next item by a semicolon character.

```
mysql> SELECT SUBSTRUCT_ATOM_IDS('C(=O)', 'NCC(=O)NCC(=O)O');
      -> 3 4 ; 7 8 ;
```

- **SUBSTRUCT_COUNT(query_smarts,reference_smiles)**

SUBSTRUCT_COUNT() returns the number of *query_smarts* fragments founded in a *reference_smiles* molecule. The first argument is a SMARTS string, whereas the second argument is a SMILES strings.

```
mysql> SELECT SUBSTRUCT_COUNT('C(=O)', 'NCC(=O)NCC(=O)O');
      -> 2
```

- **TANIMOTO(first_fingerprint,second_fingerprint)**

TANIMOTO() returns the tanimoto coefficient between two fingerprints. Fingerprints are bit patterns and can be generated with the FINGERPRINT() function. The return value is comprised between 0 and 1. The higher the tanimoto coefficient is to 1, the more the molecules are similar.

```
mysql> SELECT TANIMOTO(molecule_fp,fp_col) FROM tbl_name
      -> WHERE name='glycine';
      -> 0.8934
```

Note

The use of another Mychem functions (like FINGERPRINT() or FINGERPRINT2()) within the TANIMOTO() function makes the query slower. In order to get the best performance, you should use the SET function of MySQL:

```
mysql> SET @fp = (SELECT FINGERPRINT2(
      -> SMILES_TO_MOLECULE('C(C(=O)O)N'));
mysql> SELECT id FROM tbl_name WHERE TANIMOTO(@fp,fp_col)
      -> FROM tbl_name > 0.7;
      -> ids
```

Property Commands

The *property* module contains several functions that calculate molecular properties.

- **EXACTMASS(molecule)**

EXACTMASS() returns the monoisotopic molecular weight of a *molecule*. The monoisotopic molecular weight is defined as the molecular weight calculated using the mass of the most abundant isotope for each element of a molecule. The unit of the return value is $\text{g}\cdot\text{mol}^{-1}$.

```
mysql> SELECT EXACTMASS(molecule_col) FROM tbl_name
      -> WHERE name='glycine';
      -> 75.032028
```

- **IS_2D(molecule)**

IS_2D() returns 1 if a *molecule* has 2D coordinates.

```
mysql> SELECT IS_2D(molecule_col) FROM tbl_name
-> WHERE name='glycine';
-> 1
```

- **IS_3D(*molecule*)**

IS_3D() returns 1 if a *molecule* has 3D coordinates.

```
mysql> SELECT IS_3D(molecule_col) FROM tbl_name
-> WHERE name='glycine';
-> 0
```

- **IS_CHIRAL(*molecule*)**

IS_CHIRAL() returns 1 if a *molecule* is chiral.

```
mysql> SELECT IS_CHIRAL(molecule_col) FROM tbl_name
-> WHERE name='2S-Butan-2-ol';
-> 1
```

- **MOLFORMULA(*molecule*)**

MOLFORMULA() returns the molecular formula of a *molecule*.

```
mysql> SELECT MOLFORMULA(molecule_col) FROM tbl_name
-> WHERE name='glycine';
-> C2H5NO2
```

- **MOLLOGP(*molecule*)**

MOLLOGP() returns the LogP of a *molecule*.

```
mysql> SELECT MOLLOGP(molecule_col) FROM tbl_name
-> WHERE name='glycine';
-> -0.27
```

Note

Note that the result of this function is depending on the hydrogen atoms. If there's any doubt on the presence of hydrogen atoms in the *molecule*, it's recommended to use the **ADD_HYDROGENS()** function.

- **MOLMR(*molecule*)**

MOLMR() returns the molar refractivity of a *molecule*. The unit of the return value is $\text{J}\cdot\text{mol}^{-1}\cdot\text{K}^{-1}$.

```
mysql> SELECT MOLMR(molecule_col) FROM tbl_name
```

```
-> WHERE name='glycine';  
-> 16.2072
```

Note

Note that the result of this function is depending on the hydrogen atoms. If there's any doubt on the presence of hydrogen atoms in the *molecule*, it's recommended to use the `ADD_HYDROGENS()` function.

- **MOLPSA(*molecule*)**

MOLPSA() returns the topological polar surface area of a *molecule*. The unit of the return value is Å².

```
mysql> SELECT MOLPSA(molecule_col) FROM tbl_name  
-> WHERE name='glycine';  
-> 63.32
```

Note

Note that the result of this function is depending on the hydrogen atoms. If there's any doubt on the presence of hydrogen atoms in the *molecule*, it's recommended to use the `ADD_HYDROGENS()` function.

- **MOLWEIGHT(*molecule*)**

MOLWEIGHT() returns the molecular weight of a *molecule*. The unit of the return value is g.mol⁻¹.

```
mysql> SELECT MOLWEIGHT(molecule_col) FROM tbl_name  
-> WHERE name='glycine';  
-> 75.066600
```

- **NUMBER_OF_ACCEPTORS(*molecule*)**

NUMBER_OF_ACCEPTORS() returns the number of hydrogen-bond acceptors in a *molecule*.

```
mysql> SELECT NUMBER_OF_ACCEPTORS(molecule_col) FROM tbl_name  
-> WHERE name='glycine';  
-> 3
```

Note

Note that the result of this function is depending on the hydrogen atoms. If there's any doubt on the presence of hydrogen atoms in the *molecule*, it's recommended to use the `ADD_HYDROGENS()` function.

- **NUMBER_OF_ATOMS(*molecule*)**

NUMBER_OF_ATOMS() returns the number of atoms in a *molecule*.

```
mysql> SELECT NUMBER_OF_ATOMS(molecule_col) FROM tbl_name  
-> WHERE name='glycine';
```

```
-> 10
```

- **NUMBER_OF_BONDS(*molecule*)**

NUMBER_OF_BONDS() returns the number of bonds in a *molecule*.

```
mysql> SELECT NUMBER_OF_BONDS(molecule_col) FROM tbl_name
-> WHERE name='glycine';
-> 9
```

- **NUMBER_OF_DONORS(*molecule*)**

NUMBER_OF_DONORS() returns the numbers of hydrogen-bond donors in a *molecule*.

```
mysql> SELECT NUMBER_OF_DONORS(molecule_col) FROM tbl_name
-> WHERE name='glycine';
-> 2
```

Note

Note that the result of this function is depending on the hydrogen atoms. If there's any doubt on the presence of hydrogen atoms in the *molecule*, it's recommended to use the **ADD_HYDROGENS()** function.

- **NUMBER_OF_HEAVY_ATOMS(*molecule*)**

NUMBER_OF_HEAVY_ATOMS() returns the number of heavy atoms in a *molecule* (all atoms except hydrogen).

```
mysql> SELECT NUMBER_OF_HEAVY_ATOMS(molecule_col) FROM tbl_name
-> WHERE name='glycine';
-> 5
```

- **NUMBER_OF_RINGS(*molecule*)**

NUMBER_OF_RINGS() returns the number of rings in a *molecule*.

```
mysql> SELECT NUMBER_OF_RINGS(molecule_col) FROM tbl_name
-> WHERE name='adenine';
-> 2
```

- **NUMBER_OF_ROTABLE_BONDS(*molecule*)**

NUMBER_OF_ROTABLE_BONDS() returns the number of rotatable bonds in a *molecule*.

```
mysql> SELECT NUMBER_OF_ROTABLE_BONDS(molecule_col) FROM tbl_name
-> WHERE name='glycine';
-> 1
```

- **TOTAL_CHARGE(*molecule*)**

TOTAL_CHARGE() returns the total charge of a *molecule*.

```
mysql> SELECT TOTAL_CHARGE(SMILES_TO_MOLECULE('NCC(=O)[O-]');  
-> WHERE name='glycine';  
-> -1
```

Chapter 4. Credits and License

Contributions

This section lists the developers and contributors to Mychem.

Developers

These are developers that are involved in Mychem:

- Jérôme Pansanel <j.pansanel@pansanel.net>

Project founder and developer

- Aurélie De Luca <aureliedeluca@gmail.com>

Developer

- Bjoern Gruening <bjoern@gruenings.eu>

Developer and tester

Contributors

Contributors to Mychem are listed here:

- Ernst-Georg Schmid - Pgchem lead developer

Pgchem [<http://pgfoundry.org/projects/pgchem/>] is a chemoinformatics extension for PostgreSQL developed by Ernst-Georg Schmid. We have decided that Mychem Project should have the same functions as Pgchem. Of course, we are sharing our knowledge on Open Babel, Relational Database and extension programming.

- Chris Morley - Open Babel lead developer

For comments and help on OpenBabel-Discuss mailing list.

- Noel O'Boyle - Open Babel developer

For comments and help on OpenBabel-Discuss mailing list.

- Fredrik Wallner

For debugging and tests on Mac OS X.

Copyright

Mychem is Copyright © 2007, 2008 Imaxio SA

License

Mychem is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This software is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with the software; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA or see <http://www.gnu.org/licenses/>.

Appendix A. Installation

In this chapter, we will discuss the steps necessary to compile and install Mychem².

- How to Obtain Mychem - mainly concentrates on downloading the last stable version of Mychem.
- Requirements - lists the programs and libraries which you need installed to successfully compile Mychem.
- Compilation and Installation - leads you through all the steps of compilation and installation of the application.
- Mychem API - tells how to build the API documentation of Mychem.

How to Obtain Mychem

At this time, Mychem is only available as a source package that you need to compile. You can also get the last snapshot directly from the SVN trunk (this version can be unstable). As soon as possible, binaries will be also available for different OS.

Source Package

The last source package can be found on the Project Homepage [http://sourceforge.net/project/showfiles.php?group_id=195099] hosted by SourceForge.

SVN Trunk

The last development version of Mychem can be obtained by using anonymous SVN. In case you are using a command line interface, follow this step:

```
# svn co https://mychem.svn.sourceforge.net/mychem/trunk mychem
```

Requirements

In order to successfully compile and install Mychem, you need the following programs and libraries.

- C/C++ compiler

The compilation has been tested successfully with the GNU C Compiler (GCC).

- CMake version 2.4.5 or higher.

CMake is a multi-platform Makefile generator. It's a Free Software and can be downloaded on the CMake website [<http://www.cmake.org/HTML/Index.html>].

- MySQL version 4.0 or higher.

The MySQL database server is available from the MySQL website [<http://www.mysql.org>].

- Open Babel version 2.1.1.

The software is available from the Open Babel Home Page [http://openbabel.sourceforge.net/wiki/Main_Page].

Compilation and Installation

This section describes the compilation of Mychem's source files for GNU/Linux, Mac OS X and Microsoft Windows.

Installing Mychem on GNU/Linux

This section contains the compilation and installation instructions for Mychem on GNU/Linux.

Standard Installation on GNU/Linux

This section describes the standard way to compile and install Mychem on GNU/Linux. You have to check that the header files for the MySQL library and the Open Babel library are installed on your workstation. If they are not installed, the CMake software will raise an error when generating the Makefile. First, extract the appropriate source package. If you are using a command line interface, follow this instructions:

- For the tar gzipped archive:

```
# tar -xfzv mychem-0.6.0.tar.gz
```

- For the zip archive:

```
# unzip mychem-0.6.0.zip
```

CMake can build the libraries and executables into any directory. If the directory contains the source, the build is called *in source*. In other cases, it's called *out of source*. CMake strongly recommends and promotes building out of source.

- In source build:

```
# cd mychem-0.6.0
# cmake .
# make
# make install
```

- Build out-of-source (recommended):

```
# cd mychem-0.6.0
# mkdir build
# cd build
# cmake ..
# make
# make install
# cd ..
```

Note

You may need some root privileges to run **make install**.

Once the libraries are installed, you can create the SQL functions:

```
# mysql -u user -p < src/conversion/conversion.sql
# mysql -u user -p < src/helper/helper.sql
# mysql -u user -p < src/modification/modification.sql
# mysql -u user -p < src/molmatch/molmatch.sql
# mysql -u user -p < src/property/property.sql
```

Customized Installation

You can customized the build and installation process by modifying CMake arguments. For example, if you want to change the path of the installation directory:

```
# cd /path/to/mychem/build
# cmake -DCMAKE_INSTALL_PREFIX=/convenient/path ..
```

If you want to have more details about the compilation process, use the following option for the **make** command:

```
# make VERBOSE=1
```

Testing the installation

Since v0.5, Mychem includes a test suite. To build and use these programs, you have to set the MySQL connection settings and run the tests. The following example builds Mychem out-of-source and enables testing.

```
# cd mychem-0.6.0
# mkdir build
# cd build
# cmake -DMY_HOST=localhost -DMY_USER=user -DMY_PASSWD=passwd ..
# make
# make install
# cd ..
```

When running the command **cmake -DMY_HOST=localhost -DMY_USER=user -DMY_PASSWD=passwd ..**, you will see the following line in the status message:

```
-- Test module enabled
```

Note

If the user can access MySQL without a password, then you don't need to set the MY_PASSWD parameter. The two other parameters (MY_HOST and MY_USER) are mandatory.

Note

The password is not stored in a safe location. If you are doing tests on a critical server, please use directly the test executables and don't use the CMake facility to perform the tests ! You can find the test executables in the `/path/to/mychem/build/tests` directory.

To run the tests, use the command **make test**. You should see the following results:

```
Running tests...
Start processing tests
Test project /path/to/mychem/build
1/ 5 Testing ConversionTest           Passed
2/ 5 Testing HelperTest              Passed
3/ 5 Testing ModificationTest        Passed
4/ 5 Testing MolmatchTest            Passed
5/ 5 Testing PropertyTest            Passed

100% tests passed, 0 tests failed out of 5
```

You can find more details about the test results in the file `/path/to/mychem/build/Testing/Temporary/LastTest.log`.

Installation Troubleshooting

Building your application can raise some errors:

- If CMake returns the following error:

```
CMake Error: This project requires some variables to be set,
and cmake can not find them.
Please set the following variables:
OPENBABEL2_INCLUDE_DIR (ADVANCED)
OPENBABEL2_LIBRARIES (ADVANCED)
```

It means that CMake didn't find Open Babel. If you know where Open Babel is installed on your system, you can tell it to CMake with:

```
# cd /path/to/mychem-0.6.0/build
# cmake -DOPENBABEL2_INCLUDE_DIR=/path/to/openbabel/include \
-DOPENBABEL2_LIBRARIES=/path/to/library ..
```

Installing Mychem on Mac OS X

The installation of Mychem on Mac OS X is similar to the installation on GNU/Linux. The main differences are the parameter values for CMake. The following table shows common values for the CMake parameters. Note that these values can be different on your system.

CMake Parameters for Mac OS X Builds

OPENBABEL2_INCLUDE_DIR	/usr/local/include/openbabel-2.0
OPENBABEL2_LIBRARIES	/usr/local/lib/libopenbabel.dylib
MYSQL_INCLUDE_DIR	/Library/MySQL/include/mysql /Applications/MAMP/Library/include/mysql
MYSQL_LIBRARIES	/Library/MySQL/lib/mysql/libmysqlclient.dylib

/Applications/MAMP/Library/lib/mysql/libmysqlclient.dylib

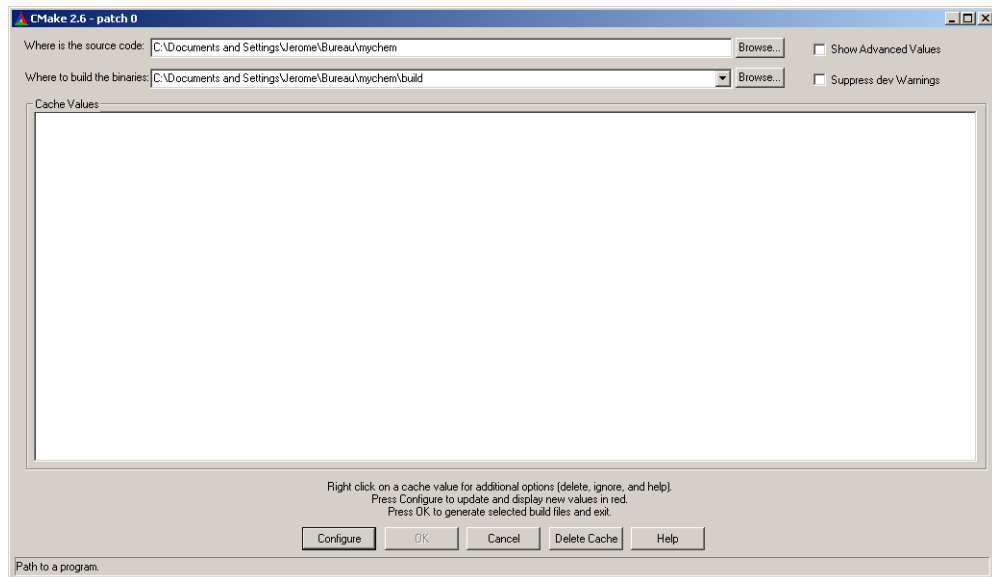
Installing Mychem on Microsoft Windows

This section describes contains the installation of Mychem on Microsoft Windows.

Installation using Microsoft Visual Studio Express 2005

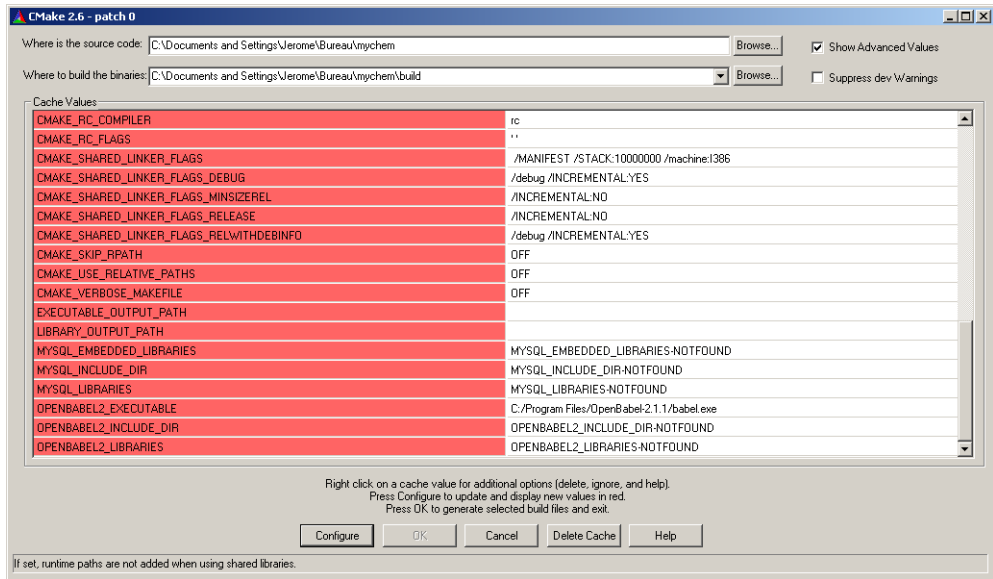
You can compile Mychem with Microsoft Visual C++ 2005 Express Edition (MSVC++). This software can be downloaded from the Microsoft MSDN Website [<http://www.microsoft.com/express/2005/>]. To complete the Microsoft Visual C++ software, install the SDK Platform. The instruction are given in the following article [[http://msdn2.microsoft.com/en-us/library/ms235626\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/ms235626(VS.80).aspx)]. The MySQL package for Windows contains all required libraries and include files for building Mychem. However, Open Babel does not provide such a package. You have to compile Open Babel. It can easily be done with MSVC++ and by following the instructions detailed on the Open Babel Website [http://openbabel.org/wiki/Install_%28MSVC%29].

Once your compilation environment is ready, you can generate the MSVC++ project file with CMake. Launch the CMake GUI and set up the source code directory and the build directory for the binaries.



Setting the source code and the build directory with the CMake utility.

Then, click on Configure. A window will appear to let you select what build system you want CMake to generate files for. Choose Visual Studio 8 2005 and click on Ok. After some processing, CMake will raise an error window, telling you that some variables are not founded. You will have to set it manually. Press twice the Ok button. Click on the Show Advanced Values checkbox to display the full list of parameters. Find the lines with OPENBABEL2_INCLUDE_DIR, OPENBABEL2_LIBRARIES, MYSQL_INCLUDE_DIR, MYSQL_LIBRARIES values. These lines are at the end of the list.

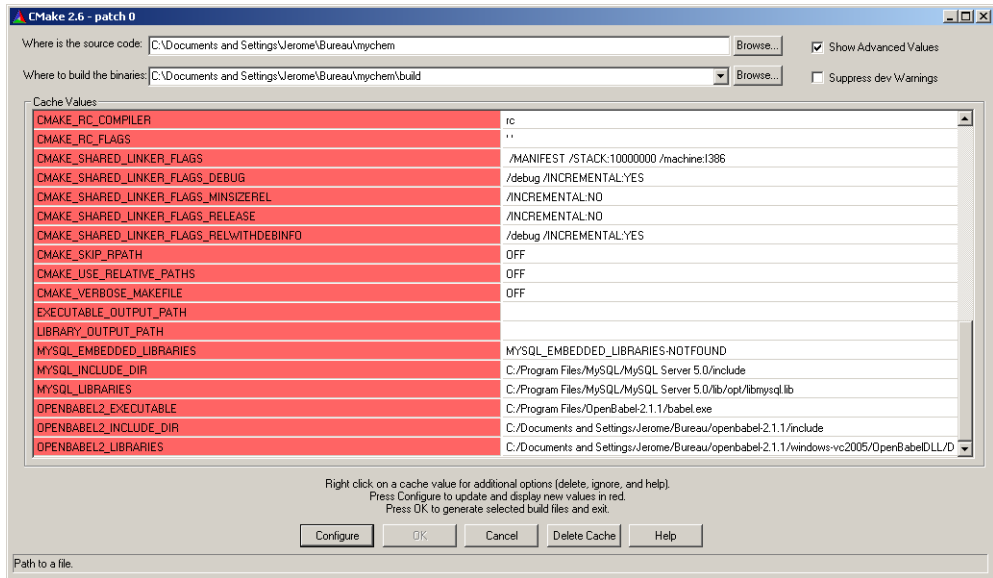


At the end of the value list, you have to set up some parameters.

You have to set OPENBABEL2_INCLUDE_DIR to the directory include contained in the Open Babel source directory (i.e., C:/path/to/openbabel/include) and OPENBABEL2_LIBRARIES to the file named OpenBabelDLL.lib (the library should be located in the C:/path/to/openbabel/windows-vc2005/OpenBabelDLL/Debug directory).

The MYSQL_INCLUDE_DIR and MYSQL_LIBRARIES parameters should be respectively set to C:\Program Files\MySQL\MySQL Server X.Y\include and C:\Program Files\MySQL\MySQL Server X.Y\opt\libmysql.lib, where X.Y is the version of MySQL.

Once the values are set, click on Configure and then on Ok. CMake generates the MSCV++ project file (mychem.sln) and exit.



All required values are filled. CMake is ready to generate the MSVC++ project file.

Once the project file is generated, open it with MSVC++. The mychem.sln project file should be located at C:/path/to/mychem/build/. You can generate each of these modules:

- conversion
- helper

- modification
- molmatch
- property

If you build the Release version of Mychem, each *module* DLL file is located in the `C:/path/to/mychem/src/module/Release/` directory and is named `mychem_module.dll`. You have to copy each of these DLL files into the MySQL bin directory. Then copy the `C:/path/to/openbabel/windows-vc2005/OpenBabelDLL/Debug/OpenBabelDLLD.dll` DLL file and all DLL files founded into the `openbabel-2.1.1/windows-vc2005` directory to the MySQL bin directory.

At last, restart MySQL and run the win32 SQL script founded into each *module* directory:

- `conversion_win32.sql`
- `helper_win32.sql`
- `modification_win32.sql`
- `molmatch_win32.sql`
- `property_win32.sql`

Mychem API

API documentation is available on the Mychem website [<http://mychem.sourceforge.net/doc/api/html/index.html>].

You can also build the documentation yourself, by using the Doxygen software [<http://www.doxygen.org/>]. To generate this documentation, use the following commands:

```
# cd /path/to/mychem-0.6.0
# doxygen Doxyfile
```

The API documentation can be read using a web browser at the following url: `file:///path/to/mychem-0.6.0/doc/api/index.html`

Appendix B. Troubleshooting

When running Mychem, you may encounter certain errors that prevent the Mychem software to run perfectly. The purpose of this chapter is to help you to diagnose and correct some of these errors.

MySQL-Related Errors

This section describes the errors encounter with MySQL when running Mychem.

ERROR 2013 (HY000): Lost connection to MySQL server during query

The following examples show an error messages you may encounter when using the SMILES_TO_MOLECULE function.

```
mysql> SELECT SMILES_TO_MOLECULE('CCOCC');  
ERROR 2013 (HY000): Lost connection to MySQL server during query
```

This problem has been observed with version of Mychem earlier to 0.6.0. To avoid this error, set the thread_stack parameter of MySQL to 192K. The thread_stack parameter is defined in the mysql-server configuration file.

Other Errors

If you encounter an error not listed here, please report it on our bug tracking system [http://sourceforge.net/tracker/?group_id=195099&atid=952081].

Appendix C. MychemAdmin

MychemAdmin is a software for managing chemical databases with MySQL. It will be replaced in the release v0.7.0 of Mychem by a new tool written in Java, and supported by the ChemiSQL project [http://chemdb.sourceforge.net/wiki/index.php/Main_Page].

- Introduction - presents the MychemAdmin software.
- Installation - details the build and installation of MychemAdmin.
- Using MychemAdmin - describes the use of MychemAdmin.

Introduction

MychemAdmin is a Python software for creating and managing chemical databases with MySQL. The graphical interface permits you to create tables and to load chemical data. You can load either MDL Molfiles or MDL SDfiles.

Installation

This section is a short overview of the build and installation procedure of MychemAdmin.

Requirements

MychemAdmin is written in Python and requires the following software:

- Python - Python is a powerful programming language. You can get it from the Python website [<http://www.python.org/>]. MychemAdmin has been successfully tested with Python 2.4 and Python 2.5.
- MySQLdb - a Python module that handle MySQL connection. You can download it from the project home page [<http://sourceforge.net/projects/mysql-python>].
- PyQt4 - Python binding for Qt4. The software is available from the Riverbank website [<http://www.riverbankcomputing.co.uk/pyqt/>].

Compilation and Installation

The installation of MychemAdmin is based on the `setup.py` script. This uses distutils [<http://www.python.org/sigs/distutils-sig/>]; the standard Python mechanism for installing packages. For the easiest installation just enter the command:

```
# python setup.py install
```

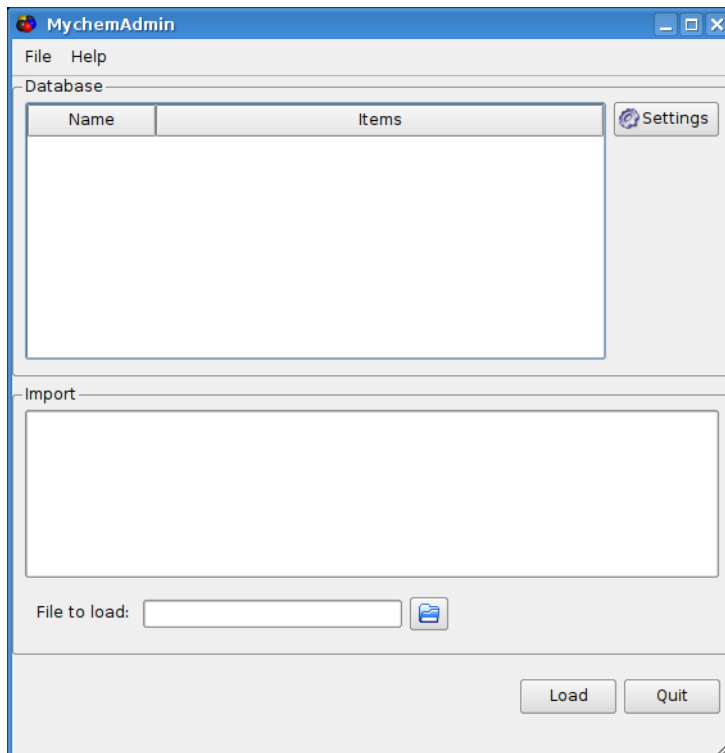
For more details about the available options, look at the *Installing Python Modules* distutils documentation, available from: <http://www.python.org/sigs/distutils-sig/doc/> [<http://www.python.org/sigs/distutils-sig/doc/>]

Using MychemAdmin

The following instructions describe how to use MychemAdmin.

The Main Window

The main window of MychemAdmin looks like:

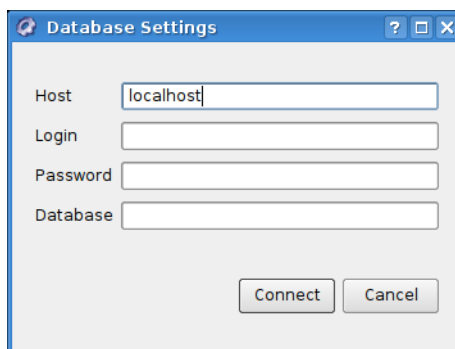


The main window is composed of two parts:

- The top half of the window is taken up by the database management. It's composed of a terminal displaying informations about the database and a Settings button.
- The second part is reserved for chemical file loading. It contains a dialog logging informations about the load process and a folder button for choosing the file to load.

Database Settings

First of all, you need to connect to the database. Click on the Settings button for opening the setting dialog:



- | | |
|----------|---|
| Host | the host where's the database running |
| Login | a user that can access the database and create tables |
| Password | a password for that user |
| Database | the database where the tables will be created. |

When all fields are filled, click on the Connect button. If the connection is successful, the dialog is closed. In the other case, an error message is displayed and the dialog stay open, letting you to correct the parameters.

File Loading

Once you are connected to the database, you can load a chemical file. Click on the folder button and select the file to load. When you have choose the right file, click on the Load button. The progress of the load process is displayed in the terminal.

Four tables are created:

- *filename* - the main table is named like the uploaded file. It contains a unique id and a name for each molecule.
- *filename_inchi* - contains the InChI code of each molecule and a reference to ``filename`.id``.
- *filename_mol* - stores the MDL Molfile of each molecule and a reference to ``filename`.id``.
- *filename_fp* - contains the fingerprint data and a reference to ``filename`.id``.

Appendix D. SDF File Loading Using Python

This appendix presents a Python script for loading a SDF File into a MySQL database. The script requires the MySQLdb Python module. It can be downloaded from the Mysql-Python Project Web Page [<http://mysql-python.sourceforge.net/>].

```
#!/bin/python

import sys
import MySQLdb

#####
#       Set some paramaters and connect to the database           #
#                                                                 #
host = "localhost" # Set the host
username = "" # Set the username
password = "" # Set the password
database = "" # Set the database name
table = "" # Set the table name

try:
    link = MySQLdb.connect(host = host,
                           user = username,
                           passwd = password,
                           db = database)

    cursor = link.cursor()
except MySQLdb.Error, e:
    print "Error %d: %s" % (e.args[0], e.args[1])
    sys.exit(1)

#                                                                 #
#####

#####
#       Parse the file and load the structures                   #
#                                                                 #
if len(sys.argv) != 2:
    print "Usage: upload_sdf.py FILE"
    link.close()
    sys.exit(1)
f_in = open(sys.argv[1], 'r')

ctTable = ""
lineCount = 0
molCount = 0
ctEnd = False

while 1:
    line = f_in.readline()
    if not line:
        break
    lineCount += 1
    if line[0:4] == "$$$$":
        if name == "":
```

```
        name = "Mol" + str(molCount)
        query = "INSERT INTO `%s` (`name`)" % table \
            + " VALUES ('%s')" % name
        cursor.execute(query)
        molid = cursor.lastrowid
        query = "INSERT INTO `%s_mol` (`molid`, `mol`)" % table \
            + " VALUES (%i, '%s')" % (molid, ctTable)
        cursor.execute(query)
        ctTable = ""
        lineCount = 0
        molCount += 1
        ctEnd = False
    elif not ctEnd:
        ctTable += line
        if lineCount == 1:
            name = line.strip()
            if line[0:6] == "M  END":
                ctEnd = True

#                                                                 #
#####

#####
#           Close the handlers and print a summary           #
#                                                                 #
link.commit()
cursor.close()
link.close()

print str(molCount) + " structures have been loaded."
#                                                                 #
#####
```