Open Babel Documentation

Geoffrey R Hutchison Chris Morley Craig James Chris Swain
Hans De Winter Tim Vandermeersch Noel M O’Boyle (Ed.)

Jul 14, 2023

Contents

The latest version of this documentation is available in several formats from http://openbabel.org/docs/dev/.

Chapter 1

Introduction

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.

1.1 Goals of the Open Babel project

Open Babel is a project to facilitate the interconversion of chemical data from one format to another – including file formats of various types. This is important for the following reasons:

- Multiple programs are often required in realistic workflows. These may include databases, modeling or computational programs, visualization programs, etc.
- Many programs have individual data formats, and/or support only a small subset of other file types.
- Chemical representations often vary considerably:
  - Some programs are 2D. Some are 3D. Some use fractional k-space coordinates.
  - Some programs use bonds and atoms of discrete types. Others use only atoms and electrons.
  - Some programs use symmetric representations. Others do not.
  - Some programs specify all atoms. Others use "residues" or omit hydrogen atoms.
- Individual implementations of even standardized file formats are often buggy, incomplete or do not completely match published standards.

As a free, and open source project, Open Babel improves by way of helping others. It gains by way of its users, contributors, developers, related projects, and the general chemical community. We must continually strive to support these constituencies.

We gratefully accept contributions in many forms – from bug reports, complaints, and critiques, which help us improve what we do poorly, to feature suggestions, code contributions, and other efforts, which direct our future development.

- For end users, we seek to provide a range of utility, from simple (or complex) file interconversion, to indexing, databasing, and transforming chemical and molecular data.
- For developers, we seek to provide an easy-to-use free and open source chemical library. This assists a variety of chemical software, from molecular viewers and visualization tools and editors to databases, property prediction tools, and in-house development.

To this end, we hope that our tools reflect several key points:

- As much chemical information and files should be read and understood by Open Babel. This means that we should always strive to support as many concepts as possible in a given file format, and support for additional file formats is beneficial to the community as a whole.
- Releases should be made to be “as good as we can make it” each and every time.
- Improving our code and our community to bring in additional contributions in many forms helps both developers and end-users alike. Making development easy for new contributors will result in better tools for users as well.

1.2 Frequently Asked Questions

1.2.1 General

What is Open Babel?

Put simply, Open Babel is a free, open-source version of the Babel chemistry file translation program. Open Babel is a project designed to pick up where Babel left off, as a cross-platform program and library designed to interconvert between many file formats used in molecular modeling, computational chemistry, and many related areas.

Open Babel includes two components, a command-line utility and a C++ library. The command-line utility is intended to be used as a replacement for the original babel program, to translate between various chemical file formats. The C++ library includes all of the file-translation code as well as a wide variety of utilities to foster development of other open source scientific software.

How does this relate to BabelChat, BabelFish, Babel IM, etc. . . ?

It doesn’t. Not surprisingly, “babel” is used frequently in a lot of software names.

Is it Open Babel or OpenBabel?

Your choice. It’s probably easier to call it Open Babel since that’s what it is–an open version of Babel. But if you like one-word, mixed-case project names, then go for OpenBabel. In that case, the space is just too small to be printed.

How does this relate to the original Babel and OELib, the “next” Babel?

The original Babel was written by Pat Walters and Matt Stahl, based on the “convert” program by Ajay Shah, and is still a remarkable application. Both Pat and Matt have moved on to other work. The original Babel is hosted by Smog.com on a Babel homepage, by the Computational Chemistry List (CCL) and of course by Open Babel at SourceForge.net.

Along the way, the two original authors started a rewrite of Babel into C++ they called OBabel, which was never really publicly released. But Matt used some of these ideas in OELib, which was generously released under the GNU GPL by his employer, OpenEye Software, and the last known version of this OELib is still available from our file repository. OpenEye decided that for their purposes OELib needed a rewrite (now called OEChem), but this would be closed-source to include some advanced algorithms. So the GPL’ed version of OELib would not be maintained. Instead, the free version of OELib was renamed and has become “Open Babel” with the blessing of Matt and other contributors.

Open Babel has evolved quite a lot since its birth in 2001.

What's the latest version?

As of this writing, the latest version is Open Babel 3.0.1. This is a stable version suitable for widespread use and development.

Can I use Open Babel code in a personal project?

One common misconception about the GNU GPL license for Open Babel is that it requires users to release any code that uses the Open Babel library. This is completely untrue. There are no restrictions on use of Open Babel code for personal projects, regardless of where you work (academia, industry, . . . wherever).

However, if you intend on releasing a software package that uses Open Babel code, the GPL requires that your package be released under the GNU GPL license. The distinction is between use and distribution. See *What's in it for me to contribute?* below for more on the licensing issues.

How do I cite Open Babel in a paper?

To support development of Open Babel, please cite:

- Hutchison et al. [obj2011]
- Open Babel, version 2.3.2, http://openbabel.org (accessed Oct 2012)

The first is a paper describing Open Babel; and the second is one way to cite a software package at a particular URL. Obviously, you should include the version number of Open Babel you used, and the date you downloaded the software or installed Open Babel.

1.2.2 Features, Formats, Roadmap

Why don't you support file format X?

The file formats currently supported are some of the more common file formats and, admittedly, those we use in our work. If you'd like to see other file formats added, we need one of:

- documentation on the file format
- working code to read the file format or translate it
- example files in the new file format and in some other format

The latter obviously is the easiest with text file formats. Binary files take some time to reverse engineer without documentation or working code. Also consider pointing developers to this FAQ and the "What's in it for me?" section.

When I convert from SMILES to MOL2/PDB/etc., why are all of the coordinates zero?

The SMILES format contains 2D information on the molecule. That is, it says which atoms are connected to which other atoms, and what type of bonds are present. MOL2, PDB and several other formats contain 3D coordinate information not present in the SMILES format. Since Open Babel does not attempt to generate 3D structure by default, all of the coordinates are set to zero. However, it is possible to generate 3D structure with the release of Open Babel 2.2.0 using the `--gen3d` option.

What sorts of features will be added in the future?

It's an open project, so if features are suggested or donated, they'll be considered as much as anything else on the drawing board. Some things are pretty clear from the roadmap.

1.2.3 What's in it for me to contribute?

What's in it for my chemistry software company?

If your product is closed-source or otherwise incompatible with the GPL, you unfortunately cannot link directly to the code library. You can, however, distribute Open Babel in unmodified form with your products to use the command-line interface. This is fairly easy because the Open Babel babel program allow reading from the standard input and writing to the standard output (functioning as a POSIX pipe).

If you decide to distribute binaries, you should either offer users the source if they want, or point them to the Open Babel website. Note that if you modify the source, you obviously can't point back to the Open Babel website – the GPL requires that you distribute the changed source. (Or you can convince us to incorporate the changes and point back to us.)

What's not to like with this deal? You can have Open Babel translate foreign file formats for you and can point users at the website for distribution. You don't need to write tons of code for all these formats and bug reports can be passed back to us.

Of course, there's one catch. You'll most likely need to add feature-rich support for your file formats. So if you contribute a small amount of code under the GPL to read/write your files, everything else is handled by Open Babel.

It's a win-win for everyone. The community benefits by having feature-rich translation code and open file formats. Your company and its programs benefit by the ability to read just about every format imaginable. Users benefit by using the programs they need for the tasks they need.

What's in it for me as an academic?

If you're an academic developer, you certainly should read the previous answer too. It takes little work on your part to interface with Open Babel and you get a lot in return.

But even if you're just an academic user, there's a lot of reasons to contribute. Most of us deal with a variety of file formats in our work. So it's useful to translate these cleanly. If a format isn't currently supported by Open Babel, see *above*. If you find bugs please report them. Since it's open source, you can patch the code yourself, recompile and have the problem fixed very quickly.

If you're inclined to write code, the GPL is an excellent option for the academic. You're the original copyright holder, so you can do whatever you want with the code, in addition to selling it. But if you've also licensed it under the GPL, no one can distribute it as proprietary (i.e., closed-source) software without your agreement. Fellow academics can use it directly, learn from it, improve it and contribute back to you. Isn't that why many of us went into science?

Once licensed under the GPL, the code must remain free to interested parties. If someone modifies it, that code must still remain under the GPL, free for all.

What's in it for an open-source software project?

Certainly the answers for closed-source software and academics also apply for you. Beyond that, if your code is compatible with the GPL, you can directly use Open Babel and all of the API. This is already happening with the Avogadro molecular editor, available under the GPL, and many others (see related projects). There's a lot of code in Open Babel beyond file translation and more to come. Why reinvent the wheel?

Why is this covered under the GPL instead of license X?

The short answer is that OpenEye Scientific Software employs Matt Stahl, one of the authors of the original Babel. They released a library called OELib under the GPL that did many things that Babel did. Later they decided to release the next version of OELib as a closed-source project–their choice for their code. We took the version of OELib still under GPL and went from there.

If you'd like to see Open Babel licensed differently, we'd suggest asking OpenEye if they'd consider releasing the old code under a new license, e.g. the LGPL. At that point, we'd consider whether Open Babel should be relicensed or not. Obviously all copyright holders must agree to the new license.

It's worth noting that since OpenEye is developing a closed-source library called OEChem and implies one reason for purchase is in closed-source development products. So we think it's highly unlikely that OpenEye would allow Open Babel to become a competitor by relicensing under the LGPL.

Where can I read more about the GNU GPL?

The Free Software Foundation maintains a FAQ list about the GNU GPL. The FAQ attempts to address common questions in an easy-to-read (i.e., not in legal language) form.

1.3 Thanks

Open Babel would not be what it is without the help of a cast of many. We are fundamentally a community project and aim to offer open development, responsive to users and contributors alike.

In addition to contributors of all sorts, a variety of related projects keep us on our toes. We would also like to thank everyone who has cited Open Babel in academic and technical literature, posters, and presentations.

Credits (in alphabetical order)

- Rich Apodaca
- Joshua Ballanco
- Michael Banck
- Ross Braithwaite
- Daniil Bratashov
- Francesco Bresciani
- Jean Brefort
- Alex Clark
- Joe Corkery
- Steve Constable
- Donald Curtis
- Andrew Dalke
- Daen de Leon
- Menno Deij
- Christian Ehrlicher
- Nick England
- Vincent Favre-Nicolin
- Maxim Fedorovsky
- Fabien Fontaine
- Malcolm Gillies
- Richard Gillilan
- Brian Goldman

- Rajarshi Guha
- Richard Hall
- Bob Hanson
- Marcus Hanwell
- Tommi Hassinen
- Bryan Herger
- David Hoekman
- Geoffrey Hutchison
- Benoît Jacob
- Craig James
- Mikael Johansson
- Stefan Kebekus
- Elmar Krieger
- Erik Kruus
- Daniel Leidert
- Christian Laggner
- Greg Landrum
- Eugene Leitl
- Teng Lin
- Zhiguo Liu
- Daniel Mansfield
- David Mathog
- Gerde Menche
- Dominik Mierzejewski
- Chris Morley
- Paul Mortenson
- Peter Murray-Rust
- Carsten Niehaus
- Anthony Nicholls
- Noel O'Boyle
- Sergei Patchkovskii
- Frank Peters
- Steffen Reith
- Louis Richard
- Roger Sayle
- Ernst-Georg Schmid
- Ajay Shah
- Kevin Shepherd
- Sangwoo Shim
- Andrew Smellie
- Matt Sprague
- Matt Stahl
- Chris Swain
- S Joshua Swamidass
- Bob Tolbert
- Sergey Trepalin
- Tim Vandermeersch
- Ugo Varetto
- Martin Vogt
- Izhar Wallach
- Fredrik Wallner
- Pat Walters
- Pawel Wolinski
- Joerg Kurt Wegner

There are probably many more who have contributed to Babel, OBabel, OELib or directly to Open Babel who are not listed here. Please help us keep this list updated. THANKS!

Chapter 2

Install Open Babel

Open Babel runs on Windows, Linux and MacOSX. You can either *install a binary package* (the easiest option) or *compile Open Babel yourself* (also easy, but much more geek cred).

2.1 Install a binary package

2.1.1 Windows

Open Babel is available as a binary installer for Windows, both 64-bit (preferred) or 32-bit (indicated by `x86` in the filename). It includes several command-line tools as well as a graphical user interface (GUI). The latest version can be download from GitHub.

Advanced users may be interested in compiling Open Babel themselves (see *Compiling Open Babel*).

2.1.2 Linux

Open Babel binary packages are available from many Linux distributions including Ubuntu, OpenSUSE and Fedora.

In general, we recommend using the latest release of Open Babel (currently 3.0.1). If this is not available for your Linux distribution, you should *compile Open Babel yourself*.

2.2 Compiling Open Babel

Open Babel is written in C++. Compiling is the process of turning this C++ into instructions that the computer's processor can understand, machine code.

Although pre-compiled (or "binary") packages are available for several platforms, there are several reasons you might want to compile Open Babel yourself:

- The current release (3.0.1) of Open Babel is not available for your platform. We recommend always using the latest release.
- You want more control over the features available. For example, perhaps you want the Python bindings but these were not included in your distribution.
- You want to use the latest development code.

- You want to add a new feature. It is easy to add new formats or operations to Open Babel as it has a plugin architecture (see *Adding plugins*).
- You just want to compile stuff yourself. We understand.

Open Babel can be compiled on Linux, MacOSX, BSDs and other Unixes, and also on Windows (with Cygwin, MinGW or MSVC).

2.2.1 Requirements

To build Open Babel, you **need** the following:

- The source code for the latest release of Open Babel
- A C++ compiler

  Open Babel is written in standards-compliant C++. The best-supported compilers are GCC 4 and MSVC++ 2008, but it also compiles with Clang and Intel Compiler 11.

- CMake 2.8 or newer

  Open Babel uses CMake as its build system. CMake is an open source cross-platform build system from KitWare.

  You need to install CMake 2.8 or newer. This is available as a binary package from the KitWare website; alternatively, it may be available through your package manager (on Linux). If necessary, you can also compile it yourself from the source code.

If you want to build the GUI (Graphical User Interface), you **need** the following in addition:

- wxWidgets 2.8 (or newer)

  Binary packages may be available through your package manager (*wx-common*, *wx2.8-headers* and *libwxbase2.8-dev* on Ubuntu) or from http://www.wxwidgets.org/downloads/. Otherwise, you could try compiling it yourself from the source code.

The following are **optional** when compiling Open Babel, but if not available some features will be missing:

- **`libxml2`** development headers are required to read/write CML files and other XML formats (the *libxml2-dev* package in Ubuntu)
- **`zlib`** development libraries are required to support reading gzipped files (the *zlib1g-dev* package in Ubuntu)
- **`Eigen`** version 2 or newer is **required** if using the language bindings in the release. In addition, if it not present, some API classes (OBAlign, OBConformerSearch) and plugins (the QEq and QTPIE charge models, the `--conformer` and `--align` operations) will not be available.

  Eigen may be available through your package manager (the *libeigen2-dev* package in Ubuntu). Alternatively, Eigen is available from http://eigen.tuxfamily.org. It doesn't need to be compiled or installed. Just unzip it and specify its location when configuring **`cmake`** (see below) using `-DEIGEN2_INCLUDE_DIR=whereever` or `-DEIGEN3_INCLUDE_DIR=wherever`.

- **`Cairo`** development libraries are required to support PNG depiction (the *libcairo2-dev* package in Ubuntu)
- If using GCC 3.x to compile (and not GCC 4.x), then the Boost headers are required for certain formats (CML, Chemkin, Chemdraw CDX, MDL RXN and RSMI)

If you want to use Open Babel using one of the supported **language bindings**, then the following notes may apply:

- You need the the Python development libraries to compile the Python bindings (package *python-dev* in Ubuntu)
- You need the the Perl development libraries to compile the Perl bindings (package *libperl-dev* in Ubuntu)

2.2.2 Basic build procedure

The basic build procedure is the same for all platforms and will be described first. After this, we will look at variations for particular platforms.

1. The recommended way to build Open Babel is to use a separate source and build directory; for example, `openbabel-2.3.2` and `build`. The first step is to create these directories:

```
$ tar zxf openbabel-2.3.2.tar.gz   # (this creates openbabel-2.3.2)
$ mkdir build
```

2. Now you need to run **cmake** to configure the build. The following will configure the build to use all of the default options:

```
$ cd build
$ cmake ../openbabel-2.3.2
```

3. If you need to specify an option, use the `-D` switch to **cmake**. For example, the following line sets the value of `CMAKE_INSTALL_PREFIX` and `CMAKE_BUILD_TYPE`:

```
$ cmake ../openbabel-2.3.2 -DCMAKE_INSTALL_PREFIX=~/Tools -DCMAKE_BUILD_TYPE=DEBUG
```

   We will discuss various possible options later.

4. At this point, it would be a good idea to compile Open Babel:

```
$ make
```

   Have a coffee while the magic happens. If you have a multi-processor machine and would prefer an expresso, try a parallel build instead:

```
$ make -j4    # parallel build across 4 processors
```

5. And finally, as root (or using `sudo`) you should install it:

```
# make install
```

2.2.3 Local build

> **Look Ma, no install!**
>
> With the right sort of environment variable magic (see *below*), you can actually use Open Babel straight from the build folder. But life is a bit easier if you install it somewhere, either system-wide or locally.

By default, Open Babel is installed in `/usr/local/` on a Unix-like system. This requires root access (or `sudo`). Even if you do have root access, you may not want to overwrite an existing installation or you may want to avoid conflicts with a version of Open Babel installed by your package manager.

The solution to all of these problems is to do a local install into a directory somewhere in your home folder. An additional advantage of a local install is that if you ever want to uninstall it, all you need to do is delete the installation directory; removing the files from a global install is more work.

1. To configure **cmake** to install into `~/Tools/openbabel-install`, for example, you would do the following:

```
$ cmake ../openbabel-2.3.2 -DCMAKE_INSTALL_PREFIX=~/Tools/openbabel-install
```

2. Then you can run **make** and **make install** without needing root access:

```
$ make && make install
```

2.2.4 Compile the GUI

The GUI is built using the wxWidgets toolkit. Assuming that you have already installed this (see *Requirements* above), you just need to configure **cmake** as follows:

```
$ cmake ../openbabel-2.3.2 -DBUILD_GUI=ON
```

When you run `make` and `make install`, the GUI will be automatically built and installed alongside the main Open Babel library and tools.

2.2.5 Compile language bindings

Eigen required

If you wish to compile the language bindings supplied in the release, Eigen version 2 or newer is required (see *Requirements* above).

1. When configuring CMake, include options such as `-DPYTHON_BINDINGS=ON -DRUBY_BINDINGS=ON` for whichever bindings you wish to build (valid names are `PYTHON`, `CSHARP`, `PERL`, `JAVA` or `RUBY`) or `-DALL_BINDINGS=ON` to build them all. The bindings will then be built and installed along with the rest of Open Babel. You should note any warning messages in the CMake output.
2. If CMake cannot find Java, you should set the value of the environment variable `JAVA_HOME` to the directory containing the Java `bin` and `lib` directories. For example, if you download the JDK from Sun and run the self-extracting .bin file, it creates a directory `jdk1.6.0_21` (or similar); you should set `JAVA_HOME` to the full path to this directory.
3. If CMake cannot find the Perl libraries (which happens on Ubuntu 10.10, surprisingly), you need to configure CMake with something like `-DPERL_LIBRARY=/usr/lib/libperl.so.5.10 -DPERL_INCLUDE_PATH=/usr/lib/perl/5.10.1/CORE`.
4. If you are compiling the CSharp bindings, you should specify the CSharp compiler to use with something like `-DCSHARP_EXECUTABLE=C:\Windows\Microsoft.NET\Framework\v3.5\csc.exe`.
5. When you run `make install`, all of the bindings will be installed to the same location as the Open Babel libraries (typically `/usr/local/lib`).
6. To prepare to use the bindings, add the install directory to the front of the appropriate environment variable: PYTHONPATH for Python, PERL5LIB for Perl, RUBYLIB for Ruby, CLASSPATH for Java, and MONO_PATH for Mono.

For example, for Python:

```
$ cmake ../openbabel-2.3.2 -DPYTHON_BINDINGS=ON
$ make
# make install
$ export PYTHONPATH=/usr/local/lib:$PYTHONPATH
```

2.2.6 Cygwin

The basic build instructions up above work just fine so long as you use the CMake provided by Cygwin rather than a native Windows installation.

If you get an error about `undefined reference to '_xmlFreeTextReader'`, you need to specify the location of the XML libraries with the `-DLIBXML2_LIBRARIES` option:

```
$ cmake ../openbabel-2.3.2 -DLIBXML2_LIBRARIES=/usr/lib/libxml2.dll.a
```

The language bindings don't seem to work under Cygwin. If you can get them to work, let us know. Also remember that anything that uses Cygwin runs slower than a native build using MinGW or MSVC++, so if speed is an issue you might prefer to compile with MinGW or MSVC++.

2.2.7 MinGW

Open Babel builds out of the box with MinGW. It's an awkward system to set up though, so here are some step-by-step instructions. . . TODO

2.2.8 Windows (MSVC)

The main Windows build used by Open Babel uses the Microsoft Visual C++ compiler (MSVC).

1. Set up the following environment variables:

    a. Add the CMake `bin` directory to the PATH.

    b. (Optional, see *Requirements* above) Set EIGEN2_INCLUDE_DIR to the location of the top level Eigen directory (if installed).

    c. (Optional, required for GUI) Set WXWIN to the top level directory of wxWidgets (if installed).

2. Install the Microsoft Visual C++ 2010 (or newer) compiler.

    We use the Visual C++ 2010 (10.0) Express Edition (available for free).

3. Open a command prompt, and change directory to the `windows-vc2008` subdirectory. To configure **`cmake`**, and generate the VC++ project files, run `default_build.bat`.

4. Double-click on `windows-vc2008/build/openbabel.sln` to start MSVC++. At the top of the window just below the menu bar, choose *Release* in the drop-down box.

5. On the left-hand side, right-click on the `ALL_BUILD` target, and choose *Build*.

2.2.9 Troubleshooting build problems

CMake caches some variables from run-to-run. How can I wipe the cache to start from scratch?

Delete `CMakeCache.txt` in the build directory. This is also a very useful file to look into if you have any problems.

How do I specify the location of the XML libraries?

CMake should find these automatically if they are installed system-wide. If you need to specify them, try using the `-DLIBXML2_LIBRARIES=wherever` option with CMake to specify the location of the DLL or SO file, and `-DLIBXML2_INCLUDE_DIR=wherever` to specify the location of the header files.

How do I specify the location of the ZLIB libraries?

CMake should find these automatically if they are installed system-wide. If you need to specify them, try using the `-DZLIB_LIBRARY=wherever` option with CMake to specify the location of the DLL or SO file, and `-DZLIB_INCLUDE_DIR=wherever` to specify the location of the header files.

What environment variables affect how Open Babel finds formats, plugins and libraries?

LD_LIBRARY_PATH - Used to find the location of the `libopenbabel.so` file. You should set this if you get error messages about not being able to find `libopenbabel.so`.

BABEL_LIBDIR - Used to find plugins such as the file formats If `obabel -L formats` does not list any file formats, then you need to set this environment variable to the directory where the file formats were installed, typically `/usr/local/lib/openbabel/`.

BABEL_DATADIR - Used to find the location of the data files used for fingerprints, forcefields, etc. If you get errors about not being able to find some .txt files, then you should set this to the name of the folder containing files such as `patterns.txt` and `MACCS.txt`. These are typically installed to `/usr/local/share/openbabel`.

2.2.10 Advanced build options

How do I control whether the tests are built?

The CMake option `-DENABLE_TESTS=ON` or `OFF` will look after this. To actually run the tests, use `make tests`.

How do I do a debug build?

`-DCMAKE_BUILD_TYPE=Debug` does a debug build (`gcc -g`). To revert to a regular build use `-DCMAKE_BUILD_TYPE=Release`.

How do I see what commands cmake is using to build?

Run Make as follows:

```
$ VERBOSE=1 make
```

How do I build one specific target?

Just specify the target when running Make. The following just builds the Python bindings:

```
$ make _openbabel
```

To speed things up, you can ask Make to ignore dependencies:

```
$ make _openbabel/fast
```

How do I create the SWIG bindings?

Use the `-DRUN_SWIG=ON` option with CMake. This requires SWIG 2.0 or newer. If the SWIG executable is not on the PATH, you will need to specify its location with `-DSWIG_EXECUTABLE=wherever`.

How do I build the Doxygen documentation?

Use the `-DBUILD_DOCS=ON` option with CMake. If the Doxygen executable is not on the PATH, you will need to specify its location with `-DDOXYGEN_EXECUTABLE=wherever`.

Chapter 3

obabel - Convert, Filter and Manipulate Chemical Data

obabel is a command-line program for interconverting between many file formats used in molecular modeling and computational chemistry and related areas. It can also be used for filtering molecules and for simple manipulation of chemical data.

3.1 Synopsis

- `obabel [-H <help-options>]`
- `obabel [-i <input-ID>] infile [-o <output-ID>] [-O outfile] [OPTIONS]`

3.2 Options

Information and help

- `obabel [-H <help-options>]`

  -H Output usage information

  -H <format-ID> Output formatting information and options for the format specified

  -Hall Output formatting information and options for all formats

  -L List plugin types (`charges`, `descriptors`, `fingerprints`, `forcefields`, `formats`, `loaders` and `ops`)

  -L <plugin type> List plugins of this type. For example, `obabel -L formats` gives the list of file formats.

  -L <plugin-ID> Details of a particular plugin (of any plugin type). For example, `obabel -L cml` gives details on the CML file format.

  -V Output version number

Conversion options

- `obabel [-i <input-ID>] infile [-o <output-ID>] [-O outfile] [OPTIONS]`
- `obabel -:"<text>" [-i <input-ID>] [-o <output-ID>] [-O outfile] [OPTIONS]`

Note: If only input and output files are given, Open Babel will guess the file type from the filename extension. For information on the file formats supported by Open Babel, please see *Supported File Formats and Options*. If text is provided using the `-:` notation, SMILES are assumed by default if an input format is not specified.

-a <options> Format-specific input options. Use `-H <format-ID>` to see options allowed by a particular format, or see the appropriate section in *Supported File Formats and Options*.

--add <list> Add properties (for SDF, CML, etc.) from descriptors in list. Use `-L descriptors` to see available descriptors.

--addfilename Add the input filename to the title.

--addinindex Append input index to title (that is, the index *before* any filtering)

--addoutindex Append output index to title (that is, the index *after* any filtering)

--addpolarh Like `-h`, but only adds hydrogens to polar atoms.

--addtotitle <text> Append the text after each molecule title

--append <list> Append properties or descriptor values appropriate for a molecule to its title. For more information, see *Append property values to the title*.

-b Convert dative bonds (e.g. `[N+]([O-])=O` to `N(=O)=O`)

-c Center atomic coordinates at (0,0,0)

-C Combine molecules in first file with others having the same name

--canonical Canonicalize the atom order. If generating canonical SMILES, do not use this option. Instead use the *Canonical SMILES format (can)*.

--conformer <options> Conformer searching to generate low-energy or diverse conformers. For more information, see *Generating conformers for structures*.

-d Delete hydrogens (make all hydrogen implicit)

--delete <list> Delete properties in list

-e Continue to convert molecules after errors

--energy <options> Forcefield energy evaluation. See *Forcefield energy and minimization*.

--errorlevel <N> Filter the level of errors and warnings displayed:

- 1 = critical errors only
- 2 = include warnings too (**default**)
- 3 = include informational messages too
- 4 = include "audit log" messages of changes to data
- 5 = include debugging messages too

-f <#> For multiple entry input, start import with molecule # as the first entry

--fillUC <param> For a crystal structure, add atoms to fill the entire unit cell based on the unique positions, the unit cell and the spacegroup. The parameter can either be `strict` (the default), which only keeps atoms inside the unit cell, or `keepconnect`, which fills the unit cell but keeps the original connectivity.

--filter <criteria> Filter based on molecular properties. See *Filtering molecules from a multi-molecule file* for examples and a list of criteria.

--gen2d Generate 2D coordinates

--gen3d Generate 3D coordinates. You can specify the speed of prediction. See *Specifying the speed of 3D coordinate generation*.

-h Add hydrogens (make all hydrogen explicit)

--highlight <substructure color> Highlight substructures in 2D depictions. Valid colors are black, gray, white, red, green, blue, yellow, cyan, purple, teal and olive. Additional colors may be specified as hexadecimal RGB values preceded by `#`. Multiple substructures and corresponding colors may be specified.

-i <format-ID> Specifies input format. See *Supported File Formats and Options*.

-j, --join Join all input molecules into a single output molecule entry

-k Translate computational chemistry modeling keywords. See the computational chemistry formats (*Computational chemistry formats*), for example *GAMESS Input (gamin, inp)* and *Gaussian Input (com, gau, gjc, gjf)*.

-l <#> For multiple entry input, stop import with molecule # as the last entry

--largest <#N descriptor> Only convert the N molecules which have the largest values of the specified descriptor. Preceding the descriptor by ~ inverts this filter.

-m Produce multiple output files, to allow:

- Splitting one input file - put each molecule into consecutively numbered output files
- Batch conversion - convert each of multiple input files into a specified output format

--minimize <options> Forcefield energy minimization. See *Forcefield energy and minimization*.

-o <format-ID> Specifies output format. See *Supported File Formats and Options*.

-p <pH> Add hydrogens appropriate for pH (use transforms in `phmodel.txt`)

--partialcharge <charge-method> Calculate partial charges by the specified method. List available methods using `obabel -L charges`.

--property <name value> Add or replace a property (for example, in an SD file)

-r Remove all but the largest contiguous fragment (strip salts)

--readconformer Combine adjacent conformers in multi-molecule input into a single molecule. If a molecule has the same structure as the preceding molecule, as determined from its SMILES, it is not output but its coordinates are added to the preceding molecule as an additional conformer. There can be multiple groups of conformers, but the molecules in each group must be adjacent.

-s <SMARTS> Convert only molecules matching the SMARTS pattern specified

-s <filename.xxx> Convert only molecules with the molecule in the file as a substructure

--separate Separate disconnected fragments into individual molecular records

--smallest <#N descriptor> Only convert the N molecules which have the smallest values of the specified descriptor. Preceding the descriptor by ~ inverts this filter.

--sort Output molecules ordered by the value of a descriptor. See *Sorting molecules*.

--title <title> Add or replace molecular title

--unique, --unique <param> Do not convert duplicate molecules. See *Remove duplicate molecules*.

--writeconformers Output multiple conformers as separate molecules

-x <options> Format-specific output options. use `-H <format-ID>` to see options allowed by a particular format, or see the appropriate section in *Supported File Formats and Options*.

-v <SMARTS> Convert only molecules **NOT** matching the SMARTS pattern specified

-z Compress the output with gzip (not on Windows)

3.3 Examples

The examples below assume the files are in the current directory. Otherwise you may need to include the full path to the files e.g. `/Users/username/Desktop/mymols.sdf` and you may need to put quotes around the filenames (especially on Windows, where they can contain spaces).

Standard conversion:

```
obabel ethanol.xyz -O ethanol.pdb
babel ethanol.xyz ethanol.pdb
```

Conversion if the files do not have an extension that describes their format:

```
obabel -ixyz ethanol.aa -opdb -O ethanol.bb
babel -ixyz ethanol.aa -opdb ethanol.bb
```

Molecules from multiple input files (which can have different formats) are normally combined in the output file:

```
obabel ethanol.xyz acetal.sdf benzene.cml -O allmols.smi
```

Conversion from a SMI file in STDIN to a Mol2 file written to STDOUT:

```
obabel -ismi -omol2
```

Split a multi-molecule file into `new1.smi`, `new2.smi`, etc.:

```
obabel infile.mol -O new.smi -m
```

In Windows this can also be written:

```
obabel infile.mol -O new*.smi
```

Multiple input files can be converted in batch format too. To convert all files ending in .xyz (`*.xyz`) to PDB files, you can type:

```
obabel *.xyz -opdb -m
```

Open Babel will not generate coordinates unless asked, so while a conversion from SMILES to SDF will generate a valid SDF file, the resulting file will not contain coordinates. To generate coordinates, use either the `--gen3d` or the `--gen2d` option:

```
obabel infile.smi -O out.sdf --gen3d
```

If you want to remove all hydrogens (i.e. make them all implicit) when doing the conversion the command would be:

```
obabel mymols.sdf -osmi -O outputfile.smi -d
```

If you want to add hydrogens (i.e. make them all explicit) when doing the conversion the command would be:

```
obabel mymols.sdf -O outputfile.smi -h
```

If you want to add hydrogens appropriate for pH7.4 when doing the conversion the command would be:

```
obabel mymols.sdf -O outputfile.smi -p
```

The protonation is done on an atom-by-atom basis so molecules with multiple ionizable centers will have all centers ionized.

Of course you don’t actually need to change the file type to modify the hydrogens. If you want to add all hydrogens the command would be:

```
obabel mymols.sdf -O mymols_H.sdf -h
```

Some functional groups e.g. nitro or sulphone can be represented either as `[N+]([O-])=O` or `N(=O)=O`. To convert all to the dative bond form:

```
obabel mymols.sdf -O outputfile.smi -b
```

If you only want to convert a subset of molecules you can define them using `-f` and `-l`. To convert molecules 2-4 of the file `mymols.sdf` type:

```
obabel mymols.sdf -f 2 -l 4 -osdf -O outputfile.sdf
```

Alternatively you can select a subset matching a SMARTS pattern, so to select all molecules containing bromobenzene use:

```
obabel mymols.sdf -O selected.sdf -s "c1ccccc1Br"
```

You can also select the subset that do *not* match a SMARTS pattern, so to select all molecules not containing bromobenzene use:

```
obabel mymols.sdf -O selected.sdf -v "c1ccccc1Br"
```

You can of course combine options, so to join molecules and add hydrogens type:

```
obabel mymols.sdf -O myjoined.sdf -h -j
```

Files compressed with gzip are read transparently, whether or not they have a .gz suffix:

```
obabel compressed.sdf.gz -O expanded.smi
```

On platforms other than Windows, the output file can be compressed with gzip, but note if you don’t specify the .gz suffix it will not be added automatically, which could cause problems when you try to open the file:

```
obabel mymols.sdf -O outputfile.sdf.gz -z
```

This next example reads the first 50 molecules in a compressed dataset and prints out the SMILES of those containing a pyridine ring, together with the index in the file, the ID (taken from an SDF property) as well as the output index:

```
obabel chembl_02.sdf.gz -osmi -l 50 -s c1ccccn1 --append chebi_id
       --addinindex --addoutindex
```

For the test data (taken from ChEMBLdb), this gave:

```
N1(CCN(CC1)c1c(cc2c3c1OCC(n3cc(c2=O)C(=O)O)C)F)C         3 100146 1
c1(c(=O)c2c(n(c1)OC)c(c(N1CC(CC1)CNCC)c(c2)F)F)C(=O)O   6 100195 2
S(=O)(=O)(Nc1ncc(cc1)C)c1c2c(c(N(C)C)ccc2)ccc1          22 100589 3
c1([nH]c2c(c1)cccc2)C(=O)N1CCN(c2c(N(CC)CC)cccn2)CC1    46 101536 4
```

3.4 Format Options

Individual file formats may have additional formatting options. These are listed in the documentation for the individual formats (see *Supported File Formats and Options*) or can be shown using the `-H <format-Id>` option, e.g. `-H cml`.

To use these additional options, input format options are preceded by `-a`, e.g. `-as`. Output format options, which are much more common, are preceded by `-x`, e.g. `-xn`. So to read the 2D coordinates (rather than the 3D) from a *CML file* and generate an *SVG file* displaying the molecule on a black background, the relevant options are used as follows:

```
obabel mymol.cml out.svg -a2 -xb
```

3.5 Append property values to the title

The command line option `--append` adds extra information to the title of the molecule.

The information can be calculated from the structure of the molecule or can originate from a property attached to the molecule (in the case of CML and SDF input files). It is used as follows:

```
obabel infile.sdf -osmi --append "MW CAT_NO"
```

`MW` is the ID of a descriptor which calculates the molecular weight of the molecule, and `CAT_NO` is a property of the molecule from the SDF input file. The values of these are added to the title of the molecule. For input files with many molecules these additions are specific to each molecule. (Note that the related option `--addtotitle` simply adds the same text to every title.)

The append option only takes one parameter, which means that it may be necessary to enclose all of the descriptor IDs or property names together in a single set of quotes.

If the name of the property in the SDF file (internally the Attribute in OBPairData) contains spaces, these spaces should be replaced by underscore characters, '_'. So the example above would also work for a property named `CAT NO`.

By default, the extra items are added to the title separated by spaces. But if the first character in the parameter is a punctuation character other than '_', it is used as the separator instead. If the list starts with "t", a tab character is used as a separator.

3.6 Generating conformers for structures

The command line option `--conformer` allows performing conformer searches using a range of different algorithms and options:

- `--log` - output a log of the energies (default = no log)
- `--nconf #` - number of conformers to generate

Forcefield-based methods for finding stable conformers:

- `--systematic` - systematically (exhaustively) generate all conformers
- `--random` - randomly generate conformers
- `--weighted` - weighted rotor search for lowest energy conformer
- `--ff <name>` - select a forcefield (default = MMFF94)

Genetic algorithm based methods (default):

- `--children #` - number of children to generate for each parent (default = 5)
- `--mutability #` - mutation frequency (default = 5)
- `--converge #` - number of identical generations before convergence is reached
- `--score #` - scoring function [rmsd|energy] (default = rmsd)

You can use them like this (to generate 50 conformers, scoring with MMFF94 energies but default genetic algorithm options):

```
obabel EtOT5D.cml -O EtOT5D0.xyz --conformer --nconf 50 --score energy
```

or if you also wish to generate 3D coordinates, followed by conformer searching try something like this:

```
obabel ligand.babel.smi -O ligand.babel.sdf --gen3d --conformer --nconf 20 --weighted
```

3.7 Filtering molecules from a multimolecule file

Six of the options above can be used to filter molecules:

- `-s` - convert molecules that match a SMARTS string
- `-v` - convert molecules that don’t match a SMARTS string
- `-f` and `-l` - convert molecules in a certain range
- `--unique` - only convert unique molecules (that is, remove duplicates)
- `--filter` - convert molecules that meet specified chemical (and other) criteria

This section focuses on the `--filter` option, which is very versatile and can select a subset of molecules based either on properties imported with the molecule (as from a SDF file) or from calculations made by Open Babel on the molecule.

The aim has been to make the option flexible and intuitive to use; don’t be put off by the long description.

You use it like this:

```
obabel filterset.sdf -osmi --filter "MW<130 ROTATABLE_BOND > 2"
```

It takes one parameter which probably needs to be enclosed in double quotes to avoid confusing the shell or operating system. (You don’t need the quotes with the Windows GUI.) The parameter contains one or more conditional tests. By default, these have all to be true for the molecule to be converted. As well as this implicit AND behaviour, you can write a full Boolean expression (see below). As you can see, there can be spaces or not in sensible places and the conditional tests could be separated by a comma or semicolon.

You can filter on two types of property:

- An SDF property, as the identifier ROTATABLE_BOND could be. There is no need for it to be previously known to Open Babel.
- A descriptor name (internally, an ID of an OBDescriptor object). This is a plug-in class so that new objects can easily be added. MW is the ID of a descriptor which calculates molecular weight. You can see a list of available descriptors using:

  ```
  obabel -L descriptors
  ```

  or from a menu item in the GUI.

Faster filtering

Open Babel provides a number of utility file formats (see *Supported File Formats and Options*). Of these, using the *copy format* as the output format is particularly useful when filtering (see *Copy raw text (copy)*). This copies the content of the molecular file directly from input to output. If you are not converting the molecules between different formats, this procedure is much faster and avoids any possibility of information loss.

In addition, if you are converting SDF files and are filtering based on the title, you should consider using `-aT` (see *MDL MOL format (mdl, mol, sd, sdf)*). Rather than perceiving the chemistry of the entire molecule, this option will only read in the title.

The descriptor names are case-insensitive. With the property names currently, you need to get the case right. Both types of identifier can contain letters, numbers and underscores, '_'. Properties can contain spaces, but then when writing the name in the filter parameter, you need to replace them with underscores. So in the example above, the test would also be suitable for a property 'ROTATABLE BOND'.

Open Babel uses a SDF-like property (internally this is stored in the class OBPairData) in preference to a descriptor if one exists in the molecule. So with the example file, which can be found here:

```
obabel filterset.sdf -osmi --filter "logP>5"
```

converts only a molecule with a property logP=10.900, since the others do not have this property and logP, being also a descriptor, is calculated and is always much less than 5.

If a property does not have a conditional test, then it returns true only if it exists. So:

```
obabel filterset.sdf -osmi --filter "ROTATABLE_BOND MW<130"
```

converts only those molecules with a ROTATABLE_BOND property and a molecular weight less than 130. If you wanted to also include all the molecules without ROTATABLE_BOND defined, use:

```
obabel filterset.sdf -osmi --filter "!ROTATABLE_BOND || (ROTATABLE_BOND & MW<130)"
```

The `!` means negate. AND can be & or &&, OR can be | or ||. The brackets are not strictly necessary here because & has precedent over | in the normal way. If the result of a test doesn't matter, it is parsed but not evaluated. In the example, the expression in the brackets is not evaluated for molecules without a ROTATABLE_BOND property. This doesn't matter here, but if evaluation of a descriptor involved a lot of computation, it would pay to include it late in the boolean expression so that there is a chance it is skipped for some molecules.

Descriptors must have a conditional test and it is an error if they don't. The default test, as used by MW or logP, is a numerical one, but the parsing of the text, and what the test does is defined in each descriptor's code (a virtual function in the OBDescriptor class). Three examples of this are described in the following sections.

3.7.1 String descriptors

```
obabel filterset.sdf -osmi --filter "title='Ethanol'"
```

The descriptor *title*, when followed by a string (here enclosed by single quotes), does a case-sensitive string comparison. (‘ethanol’ wouldn’t match anything in the example file.) The comparison does not have to be just equality:

```
obabel filterset.sdf -osmi --filter "title>='D'"
```

converts molecules with titles Dimethyl Ether and Ethanol in the example file.

It is not always necessary to use the single quotes when the meaning is unambiguous: the two examples above work without them. But a numerical, rather than a string, comparison is made if both operands can be converted to numbers. This can be useful:

```
obabel filterset.sdf -osmi --filter "title<129"
```

will convert the molecules with titles 56 123 and 126, which is probably what you wanted.

```
obabel filterset.sdf -osmi --filter "title<'129'"
```

converts only 123 and 126 because a string comparison is being made.

String comparisons can use `*` as a wildcard if used as the first or last character of the string (anywhere else a `*` is a normal character). So `--filter "title='*ol'"` will match molecules with titles ‘methanol’, ‘ethanol’ etc. and `--filter "title='eth*'` will match ‘ethanol’, ‘ethyl acetate’, ‘ethical solution’ etc. Use a `*` at both the first and last characters to test for the occurrence of a string, so `--filter "title='*ol*'"` will match ‘oleum’, ‘polonium’ and ‘ethanol’.

3.7.2 SMARTS descriptor

This descriptor will do a SMARTS test (substructure and more) on the molecules. The smarts ID can be abbreviated to s and the = is optional. More than one SMARTS test can be done:

```
obabel filterset.sdf -osmi --filter "s='CN' s!='[N+]'"
```

This provides a more flexible alternative to the existing `-s` and `-v` options, since the SMARTS descriptor test can be combined with other tests.

3.7.3 InChI descriptor

```
obabel filterset.sdf -osmi --filter "inchi='InChI=1/C2H6O/c1-2-3/h3H,2H2,1H3'"
```

will convert only ethanol. It uses the default parameters for InChI comparison, so there may be some messages from the InChI code. There is quite a lot of flexibility on how the InChI is presented (you can miss out the non-essential bits):

```
obabel filterset.sdf -osmi --filter "inchi='1/C2H6O/c1-2-3/h3H,2H2,1H3'"
obabel filterset.sdf -osmi --filter "inchi='C2H6O/c1-2-3/h3H,2H2,1H3'"
obabel filterset.sdf -osmi --filter "inchi=C2H6O/c1-2-3/h3H,2H2,1H3"
obabel filterset.sdf -osmi --filter "InChI=1/C2H6O/c1-2-3/h3H,2H2,1H3"
```

all have the same effect.

The comparison of the InChI string is done only as far as the parameter's length. This means that we can take advantage of InChI's layered structure:

```
obabel filterset.sdf -osmi --filter "inchi=C2H6O"
```

will convert both Ethanol and Dimethyl Ether.

3.8 Substructure and similarity searching

For information on using **obabel** for substructure searching and similarity searching, see *Molecular fingerprints and similarity searching*.

3.9 Sorting molecules

The `--sort` option is used to output molecules ordered by the value of a descriptor:

```
obabel  infile.xxx  outfile.xxx  --sort desc
```

If the descriptor desc provides a numerical value, the molecule with the smallest value is output first. For descriptors that provide a string output the order is alphabetical, but for the InChI descriptor a more chemically informed order is used (e.g. "CH4" is before than "C2H6", "CH4" is less than "ClH" hydrogen chloride).

The order can be reversed by preceding the descriptor name with ~, e.g.:

```
obabel  infile.xxx  outfile.yyy  --sort ~logP
```

As a shortcut, the value of the descriptor can be appended to the molecule name by adding a + to the descriptor, e.g.:

```
obabel  aromatics.smi  -osmi  --sort ~MW+
 c1ccccc1C=C   styrene 104.149
 c1ccccc1C     toluene 92.1384
 c1ccccc1      benzene 78.1118
```

3.10 Remove duplicate molecules

The `--unique` option is used to remove, i.e. not output, any chemically identical molecules during conversion:

```
obabel  infile.xxx  outfile.yyy  --unique [param]
```

The optional parameter *param* defines what is regarded as "chemically identical". It can be the name of any descriptor, although not many are likely to be useful. If *param* is omitted, the InChI descriptor is used. Other useful descriptors are 'cansmi' and 'cansmiNS' (canonical SMILES, with and without stereochemical information),'title' and truncated InChI (see below).

A message is output for each duplicate found:

```
Removed methyl benzene - a duplicate of toluene (#1)
```

Clearly, this is more useful if each molecule has a title. The `(#1)` is the number of duplicates found so far.

If you wanted to identify duplicates but not output the unique molecules, you could use the *null format*:

```
obabel  infile.xxx  -onul  --unique
```

3.10.1 Truncated InChI

It is possible to relax the criterion by which molecules are regarded as "chemically identical" by using a truncated InChI specification as *param*. This takes advantage of the layered structure of InChI. So to remove duplicates, treating stereoisomers as the same molecule:

```
obabel  infile.xxx  outfile.yyy  --unique /nostereo
```

Truncated InChI specifications start with `/` and are case-sensitive. *param* can be a concatenation of these e.g. `/nochg/noiso`:

```
/formula   formula only
/connect   formula and connectivity only
/nostereo  ignore E/Z and sp3 stereochemistry
/nosp3     ignore sp3 stereochemistry
/noEZ      ignore E/Z stereoochemistry
/nochg     ignore charge and protonation
/noiso     ignore isotopes
```

3.10.2 Multiple files

The input molecules do not have to be in a single file. So to collect all the unique molecules from a set of MOL files:

```
obabel  *.mol  uniquemols.sdf  --unique
```

If you want the unique molecules to remain in individual files:

```
obabel  *.mol  U.mol  -m  --unique
```

On the GUI use the form:

```
obabel  *.mol  U*.mol  --unique
```

Either form is acceptable on the Windows command line.

The unique molecules will be in files with the original name prefixed by 'U'. Duplicate molecules will be in similar files but with zero length, which you will have to delete yourself.

3.11 Aliases for chemical groups

There is a limited amount of support for representing common chemical groups by an alias, e.g. benzoic acid as `Ph-COOH`, with two alias groups. Internally in Open Babel, the molecule usually has a 'real' structure with the alias names present as only an alternative representation. For MDL MOL and SD files alias names can be read from or written to an 'A' line. The more modern RGroup representations are not yet recognized. Reading is transparent; the alias group is expanded and the 'real' atoms given reasonable coordinates if the the molecule is 2D or 3D. Writing in alias form, rather than the 'real' structure, requires the use of the `-xA` option. SVGFormat will also display any aliases present in a molecule if the `-xA` option is set.

The alias names that are recognized are in the file `superatoms.txt` which can be edited.

Normal molecules can have certain common groups given alternative alias representation using the `--genalias` option. The groups that are recognized and converted are a subset of those that are read. Displaying or writing them still requires the `-xA` option. For example, if `aspirin.smi` contained `O=C(O)c1ccccc1OC(=O)C`, it could be displayed with the aliases `COOH` and `OAc` by:

```
obabel aspirin.smi  -O out.svg  --genalias  -xA
```

3.12 Forcefield energy and minimization

Open Babel supports a number of forcefields which can be used for energy evaluation as well as energy minimization. The available forcefields as listed as follows:

```
C:\>obabel -L forcefields
GAFF    General Amber Force Field (GAFF).
Ghemical    Ghemical force field.
MMFF94    MMFF94 force field.
MMFF94s    MMFF94s force field.
UFF    Universal Force Field.
```

To evaluate a molecule's energy using a forcefield, use the `--energy` option. The energy is put in an OBPairData object "Energy" which is accessible via an SDF or CML property or `--append` (to title). Use `--ff <forcefield_id>` to select a forcefield (default is Ghemical) and `--log` for a log of the energy calculation. The simplest way to output the energy is as follows:

```
obabel infile.xxx -otxt --energy --append "Energy"
```

To perform forcefield minimization, the `--minimize` option is used. The following shows typical usage:

```
obabel infile.xxx -O outfile.yyy --minimize --steps 1500 --sd
```

The available options are as follows:

```
--log        output a log of the minimization process (default= no log)
--crit <converge>     set convergence criteria (default=1e-6)
--sd         use steepest descent algorithm (default = conjugate gradient)
--newton     use Newton2Num linesearch (default = Simple)
--ff <forcefield-id>       select a forcefield (default = Ghemical)
--steps <number>    specify the maximum number of steps (default = 2500)
--cut        use cut-off (default = don't use cut-off)
--rvdw <cutoff>     specify the VDW cut-off distance (default = 6.0)
--rele <cutoff>     specify the Electrostatic cut-off distance (default = 10.0)
--freq <steps>     specify the frequency to update the non-bonded pairs (default = 10)
```

Note that for both `--energy` and `--minimize`, hydrogens are made explicit before energy evaluation.

3.13 Aligning molecules or substructures

The `--align` option aligns molecules to the first molecule provided. It is typically used with the `-s` option to specify an alignment based on a substructure:

```
obabel pattern.www  dataset.xxx  -O outset.yyy  -s SMARTS  --align
```

Here, only molecules matching the specified SMARTS pattern are converted and are aligned by having all their atom coordinates modified. The atoms that are used in the alignment are those matched by SMARTS in the first output molecule. The subsequent molecules are aligned so that the coordinates of atoms equivalent to these are as nearly as possible the same as those of the pattern atoms. The atoms in the various molecules can be in any order. Tha alignment ignores hydrogen atoms but includes symmetry. Note that the standalone program **obfit** has similar functionality.

The first input molecule could also be part of the data set:

```
obabel dataset.xxx  -O outset.yyy  -s SMARTS  --align
```

This form is useful for ensuring that a particular substructure always has the same orientation in a 2D display of a set of molecules. 0D molecules, for example from SMILES, are given 2D coordinates before alignment.

See documentation for the -s option for its other possible parameters. For example, the matching atoms could be those of a molecule in a specified file.

If the -s option is not used, all of the atoms in the first molecule are used as pattern atoms. The order of the atoms must be the same in all the molecules.

The output molecules have a property (represented internally as OBPairData) called rmsd, which is a measure of the quality of the fit. To attach it to the title of each molecule use --append rmsd.

To output the two conformers closest to the first conformer in a dataset:

```
obabel dataset.xxx  -O outset.yyy  --align  --smallest 2 rmsd
```

3.14 Specifying the speed of 3D coordinate generation

When you use the --gen3d option, you can specify the speed and quality. The following shows typical usage:

```
obabel infile.smi -O out.sdf --gen3d fastest
```

The available options are as follows:

option	description
fastest	No cleanup
fast	Force field cleanup (100 cycles)
med (default)	Force field cleanup (100 cycles) + Fast rotor search (only one permutation)
slow	Force field cleanup (250 cycles) + Fast rotor search (permute central rotors)
slowest	Force field cleanup (500 cycles) + Slow rotor search
better	Same as slow
best	Same as slowest
dist, dg	Use distance geometry method (unstable)

You can also specify the speed by an integer from 1 (slowest) to 5 (fastest).

Chapter 4

The Open Babel GUI

The **obabel** command line program converts chemical objects (currently molecules or reactions) from one file format to another. The Open Babel graphical user interface (GUI) is an alternative to using the command line and has the same capabilities. Since Open Babel 2.3, the GUI is available cross-platform on Windows, Linux and MacOSX. On Windows, you can find it in the Start Menu in the Open Babel folder; on Linux and MacOSX, the GUI can be started with the **obgui** command.

Since the functionality of the GUI mirrors that of **obabel**, you should consult the *previous chapter* to learn about available features and how to use them. This chapter describes the general use of the GUI and then focuses on features that are specific to the GUI.

4.1 Basic operation

Although the GUI presents many options, the basic operation is straightforward:

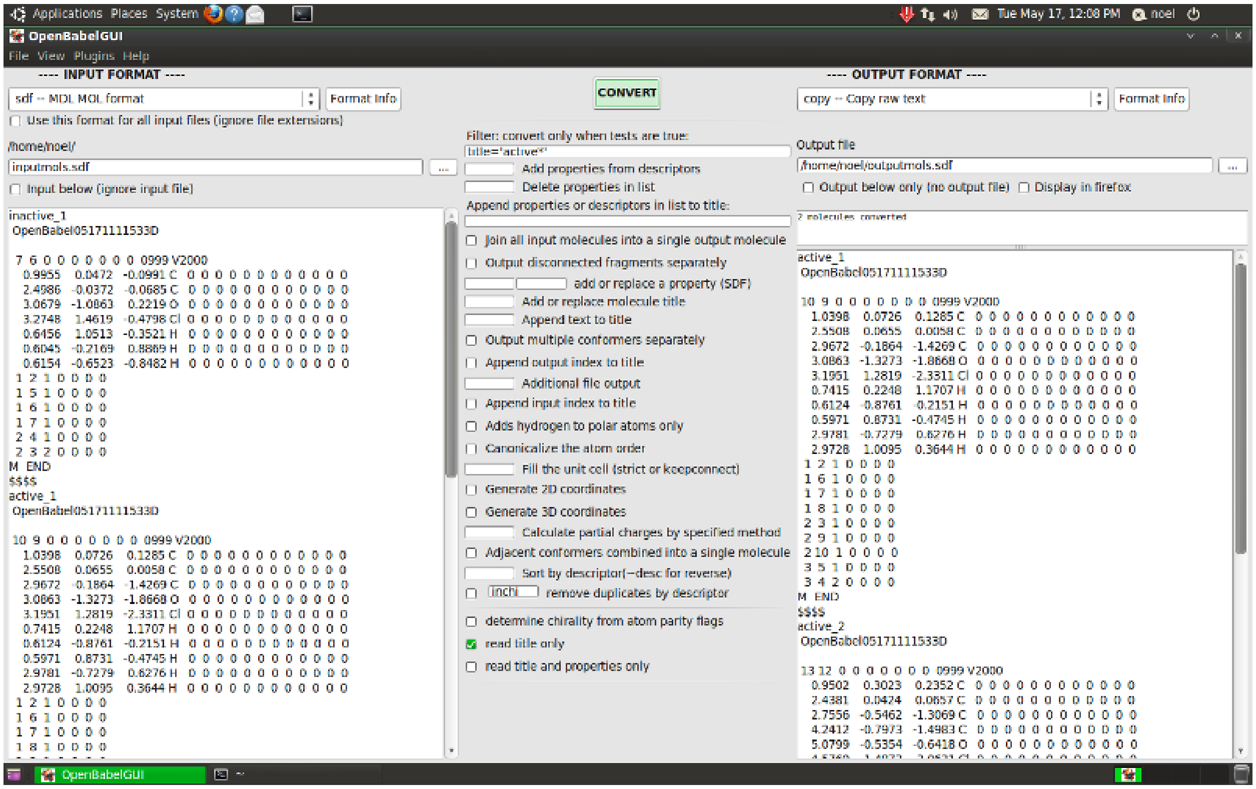
- Select the type of the type of the input file from the dropdown list.
- Click the ". . . " button and select the file. Its contents are displayed in the textbox below.
- Choose the output format and file in a similar way. You can merely display the output without saving it by not selecting an output file or by checking "*Output below only..*".
- Click the "*Convert*" button.

The message window below the button gives the number of molecules converted, and the contents of the output file are displayed.

By default, all the molecules in an input file are converted if the output format allows multiple molecules.

4.2 Options

The options in the middle are those appropriate for the type of chemical object being converted (molecule or reaction) and the input and output formats. They are derived from the description text that is displayed with the `-Hxxx` option in the command line interface and with the "*Format info*" buttons here. You can switch off the display of any of the various types of option using the *View* menu if the screen is getting too cluttered.



---- INPUT FORMAT ----

sdf -- MDL MOL format

□ Use this format for all input files (ignore file extensions)

/home/noel/

inputmols.sdf

□ Input below (ignore input file)

```
inactive_1
 OpenBabel05171111533D

 7  6  0  0  0  0  0  0  0  0999 V2000
    0.9955    0.0472   -0.0991 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.4986   -0.0372   -0.0685 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.0679   -1.0863    0.2219 O   0  0  0  0  0  0  0  0  0  0  0  0
    3.2748    1.4619   -0.4798 Cl  0  0  0  0  0  0  0  0  0  0  0  0
    0.6456    1.0513   -0.3521 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.6045   -0.2169    0.8869 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.6154   -0.6523   -0.8482 H   0  0  0  0  0  0  0  0  0  0  0  0
  1  2  1  0  0  0  0
  1  5  1  0  0  0  0
  1  6  1  0  0  0  0
  1  7  1  0  0  0  0
  2  4  1  0  0  0  0
  2  3  2  0  0  0  0
M  END
$$$$
active_1
 OpenBabel05171111533D

 10  9  0  0  0  0  0  0  0  0999 V2000
    1.0398    0.0726    0.1285 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.5508    0.0655    0.0058 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.9672   -0.1864   -1.4269 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.0863   -1.3273   -1.8668 O   0  0  0  0  0  0  0  0  0  0  0  0
    3.1951    1.2819   -2.3311 Cl  0  0  0  0  0  0  0  0  0  0  0  0
    0.7415    0.2248    1.1707 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.6124   -0.8761   -0.2151 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.5971    0.8731   -0.4745 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.9781   -0.7279    0.6276 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.9728    1.0095    0.3644 H   0  0  0  0  0  0  0  0  0  0  0  0
  1  2  1  0  0  0  0
  1  6  1  0  0  0  0
  1  7  1  0  0  0  0
  1  8  1  0  0  0  0
```

Filter: convert only when tests are true:

title='active*'

Add properties from descriptors

Delete properties in list

Append properties or descriptors in list to title:

□ Join all input molecules into a single output molecule

□ Output disconnected fragments separately

add or replace a property (SDF)

Add or replace molecule title

Append text to title

□ Output multiple conformers separately

□ Append output index to title

Additional file output

□ Append input index to title

□ Adds hydrogen to polar atoms only

□ Canonicalize the atom order

Fill the unit cell (strict or keepconnect)

□ Generate 2D coordinates

□ Generate 3D coordinates

Calculate partial charges by specified method

□ Adjacent conformers combined into a single molecule

Sort by descriptor(−desc for reverse)

□ inchi remove duplicates by descriptor

□ determine chirality from atom parity flags

☑ read title only

□ read title and properties only

---- OUTPUT FORMAT ----

copy -- Copy raw text

Output file

/home/noel/outputmols.sdf

□ Output below only (no output file) □ Display in firefox

```
2 molecules converted
```

```
active_1
 OpenBabel05171111533D

 10  9  0  0  0  0  0  0  0  0999 V2000
    1.0398    0.0726    0.1285 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.5508    0.0655    0.0058 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.9672   -0.1864   -1.4269 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.0863   -1.3273   -1.8668 O   0  0  0  0  0  0  0  0  0  0  0  0
    3.1951    1.2819   -2.3311 Cl  0  0  0  0  0  0  0  0  0  0  0  0
    0.7415    0.2248    1.1707 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.6124   -0.8761   -0.2151 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.5971    0.8731   -0.4745 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.9781   -0.7279    0.6276 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.9728    1.0095    0.3644 H   0  0  0  0  0  0  0  0  0  0  0  0
  1  2  1  0  0  0  0
  1  6  1  0  0  0  0
  1  7  1  0  0  0  0
  1  8  1  0  0  0  0
  2  3  1  0  0  0  0
  2  9  1  0  0  0  0
  2 10  1  0  0  0  0
  3  5  1  0  0  0  0
  3  4  2  0  0  0  0
M  END
$$$$
active_2
 OpenBabel05171111533D

 13 12  0  0  0  0  0  0  0  0999 V2000
    0.9502    0.3023    0.2352 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.4381    0.0424    0.0657 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.7556   -0.5462   -1.3069 C   0  0  0  0  0  0  0  0  0  0  0  0
    4.2412   -0.7973   -1.4983 C   0  0  0  0  0  0  0  0  0  0  0  0
    5.0799   -0.5354   -0.6418 O   0  0  0  0  0  0  0  0  0  0  0  0
```

Fig. 1: **Screenshot of GUI running on BioLinux 6.0, an Ubuntu derivative**

4.3 Multiple input files

You can select multiple input files in the input file dialog in the normal way (for example, using the Control key in Windows). In the input filename box, each filename is displayed relative to the path shown just above the box, which is the path of the first file. You can display any of the files by moving the highlight with Tab/Shift Tab, Page Up/Down, the mouse wheel, or by double clicking.

Selecting one or more new file names normally removes those already present, but they can instead be appended by holding the Control key down when leaving the file selection dialog.

Files can be also be dragged and dropped (e.g. from Windows Explorer), adding the file when the Control key is pressed, replacing the existing files when it is not.

Normally each file is converted according to its extension and the input files do not have to be all the same, but if you want to use non-standard file names set the checkbox "*Use this format for all input files. . .*"

If you want to combine multiple molecules (from one or more files) into a single molecule with disconnected parts, use option "*Join all input molecules. . .*"

4.4 Wildcards in filenames

When input filenames are typed in directly, any of them can contain the wildcard characters `*` and `?`. Typing Enter will replace these by a list of the matching files. The wildcarded names can be restored by typing Enter while holding down the Shift key. The original or the expanded versions will behave the same when the "*Convert*" button is pressed.

By including the wildcard `*` in both the input and output filenames you can carry out batch conversion. Suppose there were files `first.smi`, `second.smi`, `third.smi`. Using `*.smi` as the input filename and `*.mol` as the output filename would produce three files `first.mol`, `second.mol` and `third.mol`. If the output filename was `NEW_*.mol`, then the output files would be `NEW_first.mol`, etc.

4.5 Local input

By checking the "*Input below...*" checkbox you can type the input text directly. The text box changes colour to remind you that it is this text and not the contents of any files that will be converted.

4.6 Output file

The output file name can be fully specified with a path, but if it is not, then it is considered to be relative to the input file path.

4.7 Graphical display

The chemical structures being converted can be displayed (as SVG) in an external program. By default this is Firefox but it can be changed from an item on the *View* menu (for instance, Opera and Chrome work fine). When "*Display in firefox*" (under the output file name) is checked, the structures will be shown in a new Firefox tab. With multiple molecules the display can be zoomed (mousewheel) and panned (dragging with mouse button depressed). Up to 100 molecules are easily handled but with more the system may be slow to manipulate. It may also be slow to generate, especially if 2D atom coordinates have to be calculated (e.g.from SMILES). A new Firefox tab is opened each time *Convert* is pressed.

4.8 Using a restricted set of formats

It is likely that you will only be interested in a subset of the large range of formats handled by Open Babel. You can restrict the choice offered in the dropdown boxes, which makes routine selection easier. Clicking "*Select set of formats*" on the *View* menu allows the formats to be displayed to be selected. Subsequently, clicking "*Use restricted set of formats*" on the *View* menu toggles this facility on and off.

Using a restricted set overcomes an irritating bug in the Windows version. In the file *Open* and *Save* dialogs the files displayed can be filtered by the *current format*, *All Chemical Formats*, or *All Files*. The *All Chemical Formats* filter will only display the first 30 possible formats (alphabetically). The *All Files* will indeed display all files and the conversion processes are unaffected.

4.9 Other features

Most of the interface parameters, such as the selected format and the window size and position, are remembered between sessions.

Using the *View* menu, the input and output text boxes can be set not to wrap the text. At present you have to restart the program for this to take effect.

The message box at the top of the output text window receives program output on error and audit logging, and some progress reports. It can be expanded by dragging down the divider between the windows.

4.10 Example files

In the Windows distribution, there are three chemical files included to try out:

- **serotonin.mol** which has 3D atom coordinates
- **oxamide.cml** which is 2D and has a large number of properties that will be seen when converting to SDF
- **FourSmallMols.cml** which (unsurprisingly) contains four molecules with no atom coordinates and can be used to illustrate the handling of multiple molecules:

  Setting the output format to SMI (which is easy to see), you can convert only the second and third molecules by entering `2` and `3` in the appropriate option boxes. Or convert only molecules with C-O single bonds by entering `CO` in the SMARTS option box.

Chapter 5

Tutorial on using the GUI

This chapter gives step-by-step descriptions on how to use Open Babel's graphical user interface (GUI) to carry out some common tasks. It may also be used as the basis of a practical on cheminformatics, and to this end several questions are interspersed with the tutorial text.

For more information on the GUI itself, see the *previous chapter*.

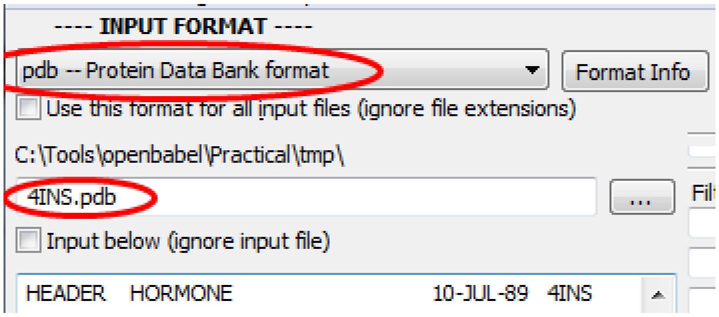
5.1 Converting chemical file formats

The most common use of Open Babel is to convert chemical file formats. The following examples show how this is done.

5.1.1 File conversion

Let's convert a PDB file to MOL format:

- Create a folder on the Desktop called `Work`
- Download the PDB file for insulin (`4ins`) from the Protein Data Bank and save it in the `Work` folder
- Set the input file format to PDB and the input filename to the downloaded PDB file



---- INPUT FORMAT ----

pdb -- Protein Data Bank format

□ Use this format for all input files (ignore file extensions)

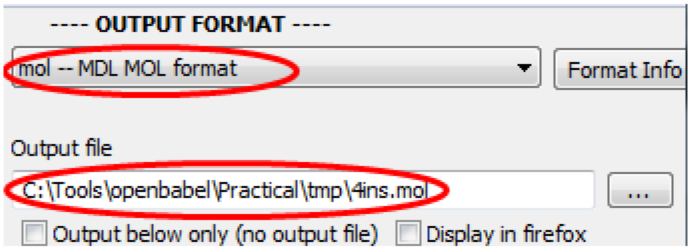
C:\Tools\openbabel\Practical\tmp\

4INS.pdb

□ Input below (ignore input file)

```
HEADER    HORMONE                    10-JUL-89   4INS
```

- Set the output file format to MOL and the output filename to file:*4ins.mol* in the `Work` folder
- Now click *CONVERT*



---- OUTPUT FORMAT ----

mol -- MDL MOL format

Output file

C:\Tools\openbabel\Practical\tmp\4ins.mol

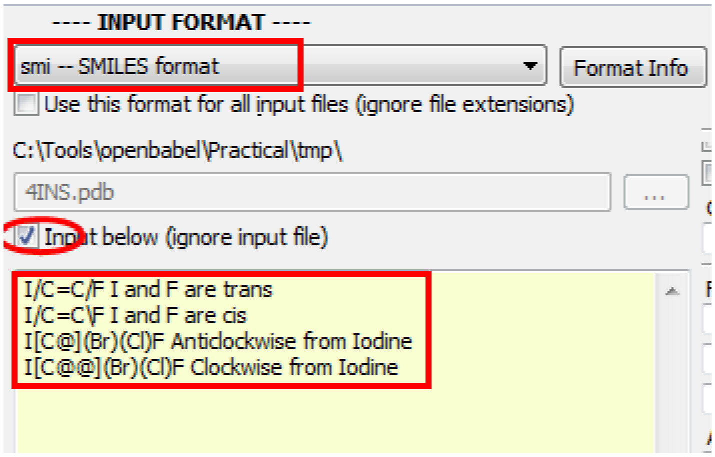
□ Output below only (no output file) □ Display in firefox

5.1.2 Converting without files

Rather than use input and output files, it is possible to paste the contents of a chemical file format into the input box, and see the results of the conversion in the output box.

Here we will try this with the SMILES format, and illustrate how stereochemistry is handled by SMILES:

Screenshot



---- INPUT FORMAT ----

smi -- SMILES format

□ Use this format for all input files (ignore file extensions)

C:\Tools\openbabel\Practical\tmp\

4INS.pdb

☑ Input below (ignore input file)

```
I/C=C/F I and F are trans
I/C=C\F I and F are cis
I[C@](Br)(Cl)F Anticlockwise from Iodine
I[C@@](Br)(Cl)F Clockwise from Iodine
```

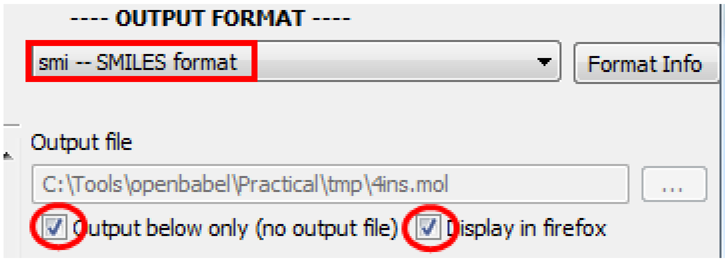
- Choose the SMILES format as the input format
- Tick the box *Input below (ignore input file)*
- Copy and paste the following SMILES strings (and molecule titles) into the input box:

```
I/C=C/F I and F are trans
I/C=C\F I and F are cis
I[C@](Br)(Cl)F Anticlockwise from Iodine
I[C@@](Br)(Cl)F Clockwise from Iodine
```

- Choose the SMILES format as the output format
- Tick the box for *Output below only* and *Display in Firefox*
- Click *CONVERT*.

Which stereobond does Open Babel set?

There are four bonds from each stereocentre. Open Babel carefully chooses which bond to set as the stereobond by considering whether the bond connects two stereocentres, whether the bond is part of a ring, and the angular distance between bonds.



---- OUTPUT FORMAT ----

smi -- SMILES format

Output file

C:\Tools\openbabel\Practical\tmp\4ins.mol

☑ Output below only (no output file) ☑ Display in firefox

In the resulting depiction, note that Open Babel only sets a single stereobond for a chiral centre. This is not ambiguous - it means that the stereobond is either above or below the plane, with the remaining three bonds the opposite site of the plane.

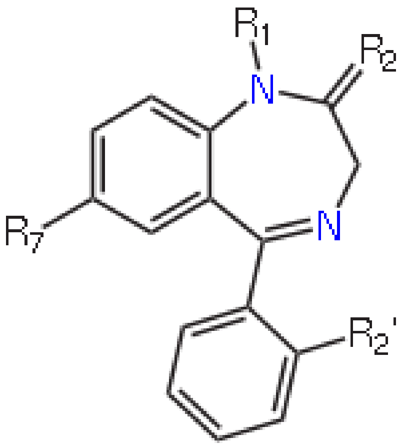
Q. Can you figure out whether the depiction of the tetrahedral centre is consistent with the SMILES string?

Note: Open Babel 2.3.2 introduces a twisted double bond to indicate unknown cis/trans stereochemistry (e.g. IC=CF). See here for more info.

5.2 Filtering structures

Setup

We are going to use a dataset of 16 benzodiazepines. These all share the following substructure (image from Wikipedia):



- Create a folder on the Desktop called `Work` and save benzodiazepines.sdf there
- Set up a conversion from SDF to SMI and set `benzodiazepines.sdf` as the input file
- Tick *Display in Firefox*
- Click *CONVERT*

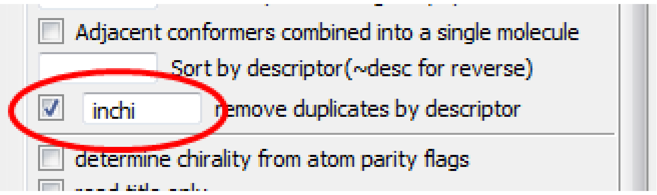
Remove duplicates

If you look carefully at the depictions of the first and last molecules (top left and bottom right) you will notice that they depict the same molecule.

Q. Look at the SMILES strings for the first and last molecules. If the two molecules are actually the same, why are the two SMILES strings different? (Hint: try using `CAN - canonical SMILES` instead of `SMI`.)

We can remove duplicates based on the InChI (for example):

- Tick the box beside *remove duplicates by descriptor* and enter `inchi` as the descriptor



□ Adjacent conformers combined into a single molecule

Sort by descriptor(~desc for reverse)

☑ inchi remove duplicates by descriptor

□ determine chirality from atom parity flags

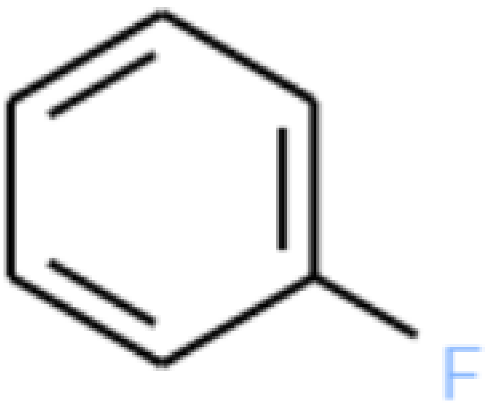
- Click *CONVERT*

Duplicates can be removed based on any of the available descriptors. The full list can be found in the menu under *Plugins*, *descriptors*.

Q. Are any of the other descriptors useful for removing duplicates?

Filtering by substructure

Q. How many of the molecules contain the following substructure?



The SMILES string for this molecule is `c1ccccc1F`. This is also a valid SMARTS string.

Q. Use the SMARTSviewer at the ZBH Center for Bioinformatics, University of Hamburg, to verify the meaning of the SMARTS string `c1ccccc1F`.

Removing potentially toxic molecules

Filtering a dataset of molecules by substructure is particularly useful if you need to remove molecules with problematic functional groups. For example, particular functional groups are associated with toxicological problems.

Let's filter the molecules using this substructure:

- In the Options section, enter `c1ccccc1F` into the box labeled *Convert only if match SMARTS or mols in file*
- Click *CONVERT*.

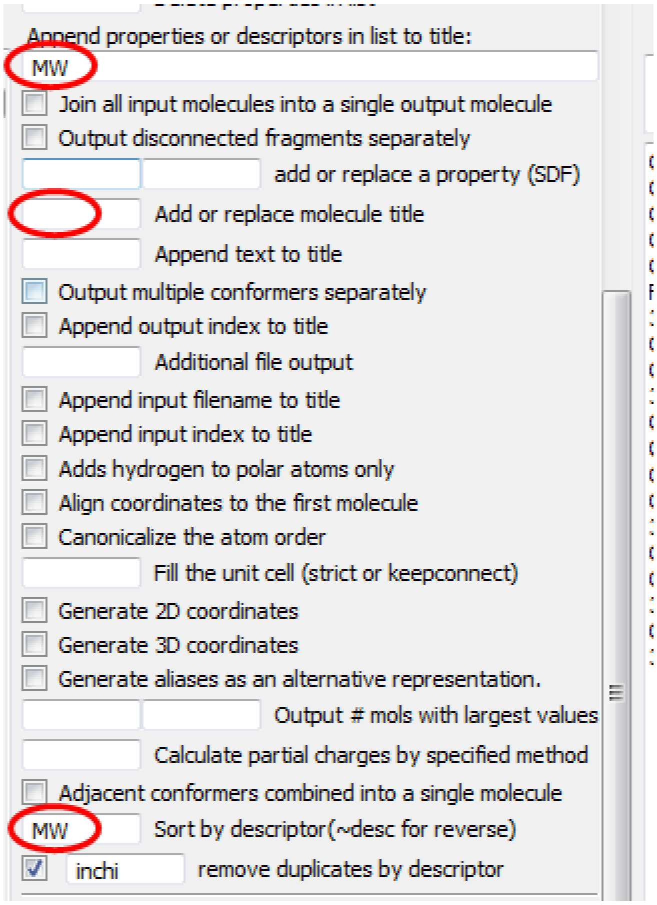
Q. How many structures are matched?

- Now find all those that are not matched by preceding the SMARTS filter with a tilde `~`, i.e. `~c1ccccc1F`.
- Click *CONVERT*.

Q. How many structures are not matched?

Filter by descriptor

Screenshot



Append properties or descriptors in list to title: MW

□ Join all input molecules into a single output molecule

□ Output disconnected fragments separately

add or replace a property (SDF)

Add or replace molecule title

Append text to title

□ Output multiple conformers separately

□ Append output index to title

Additional file output

□ Append input filename to title

□ Append input index to title

□ Adds hydrogen to polar atoms only

□ Align coordinates to the first molecule

□ Canonicalize the atom order

Fill the unit cell (strict or keepconnect)

□ Generate 2D coordinates

□ Generate 3D coordinates

□ Generate aliases as an alternative representation.

Output # mols with largest values

Calculate partial charges by specified method

□ Adjacent conformers combined into a single molecule

MW Sort by descriptor(~desc for reverse)

☑ inchi remove duplicates by descriptor

As discussed above, Open Babel provides several descriptors. Here we will focus on the molecular weight, `MW`.

To begin with, let's show the molecular weights in the depiction:

- Clear the existing title by entering a single space into the box *Add or replace molecule title*
- Set the title to the molecular weight by entering `MW` into the box *Append properties or descriptors in list to title*
- Click *CONVERT*

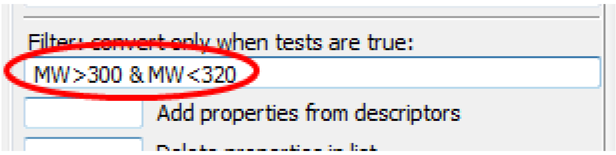
You should see the molecular weight below each molecule in the depiction. Notice also that the SMILES output has the molecular weight beside each molecule. This could be useful for preparing a spreadsheet with the SMILES string and various calculated properties.

Now let's sort by molecular weight:

- Enter `MW` into the box *Sort by descriptor* and click *CONVERT*

Finally, here's how to filter based on molecular weight. Note that none of the preceding steps are necessary for the filter to work. We will convert all those molecules with molecular weights between 300 and 320 (in the following expression `&` signifies Boolean AND):

- Enter `MW>300 & MW<320` into the box *Filter convert only when tests are true* and click *CONVERT*



Filter: convert only when tests are true:

MW>300 & MW<320

Add properties from descriptors

Delete properties in list

Filter by property

The SDF format, in common with some other file formats, allows property fields for each molecule. Open Babel allows the user to filter using these, add the value to the title, remove or replace values.

Q. If `|` (the pipe symbol, beside Z on the UK keyboard) signifies Boolean OR, how would you instead convert all those molecules that do not have molecular weights between 300 and 320?

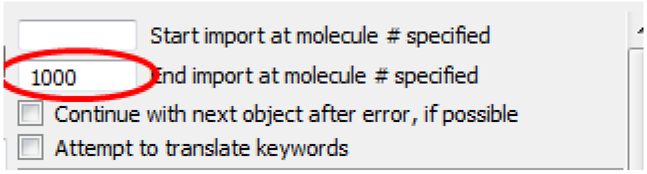
Note: Open Babel 2.3.2 allows specific substructures to be highlighted in a depiction. It also allows depictions to be aligned based on a substructure.

5.3 Substructure and similarity searching a large dataset

Open Babel provides a format called the `fs -- fastsearch index` which should be used when searching large datasets (like ChEMBL) for molecules similar to a particular query. There are faster ways of searching (like using a chemical database) but FastSearch is convenient, and should give reasonable performance for most people.

To demonstrate similarity searching, we will use the first 1000 molecules in the latest release of ChEMBL:

- Download the 2D SDF version of ChEMBL, `chembl_nn.sdf.gz`, from the ChEMBLdb download site and save in your *Work* folder. (Note: this is a gzipped file, but Open Babel will handle this without problems.)
- Set up an SDF to SDF conversion, set `chembl_nn.sdf.gz` as the input file and `1000_chembl.sdf` as the output file.
- Only convert the first 1000 molecules by entering `1000` in the box *End import at molecule # specified*.



Start import at molecule # specified

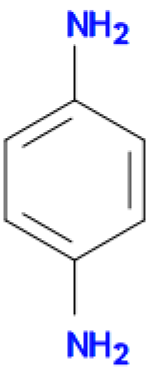
1000 End import at molecule # specified

□ Continue with next object after error, if possible

□ Attempt to translate keywords

- Click *CONVERT*

We can going to use the following structure for substructure and similarity searching. It can be represented by the SMILES string `Nc1ccc(N)cc1`.



How the search works

Behind the scenes, the FastSearch index simply stores a path-based binary fingerprint for each molecule. When used to search, similarity is measured based on the Tanimoto coefficient. For exact search, hits are verified by Open Babel's graph isomorphism matcher.

Next, we will create a FastSearch index for this dataset of 1000 molecules:

- Convert `1000_chembl.sdf` from SDF to FS format, with an output filename of `1000_chembl.fs`

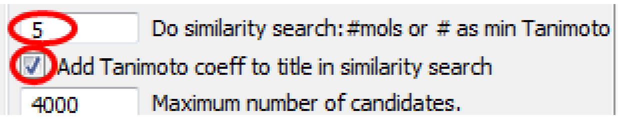
By using this FastSearch index, the speed of substructure and similarity searching is much improved. First of all, let's do a substructure search:

- Set up a conversion from FS to SMILES with `1000_chembl.fs` as the input file. Tick the box for *Output below only* and *Display in Firefox*
- Enter `Nc1ccc(N)cc1` into the box *Convert only if match SMARTS or mol in file*
- Click *CONVERT*

Q. How does the speed of the substructure search compare to if you used `1000_chembl.sdf` as the input file instead?

Next, let's find the 5 most similar molecules to the same query. The Tanimoto coefficient of a path-based fingerprint is used as the measurement of similarity. This has a value from 0.0 to 1.0 (maximum similarity) and we will display the value below each molecule:

- Set up the FS to SMILES conversion as before, and again enter `Nc1ccc(N)cc1` into the box *Convert only if match SMARTS or mol in file*
- Enter `5` into the box *Do similarity search: #mols or # as min Tanimoto*
- Tick the box *Add Tanimoto coefficient to title in similarity search*
- Click *CONVERT*



5 Do similarity search:#mols or # as min Tanimoto

☑ Add Tanimoto coeff to title in similarity search

4000 Maximum number of candidates.

Q. Look at the 5 most similar molecules. Can you tell why they were regarded as similar to the query?

Chapter 6

Molecular fingerprints and similarity searching

Molecular fingerprints are a way of encoding the structure of a molecule. The most common type of fingerprint is a series of binary digits (bits) that represent the presence or absence of particular substructures in the molecule. Comparing fingerprints allows you to determine the similarity between two molecules, to find matches to a query substructure, etc.

Open Babel provides several fingerprints of different types:

- *Fingerprint format*: the path-based fingerprint FP2; substructure based fingerprints FP3, FP4 and MACCS; user-defined substructures
- *Multilevel Neighborhoods of Atoms (MNA) (mna)*: a circular fingerprint
- *MolPrint2D format (mpd)*: a circular fingerprint
- *Spectrophores*™: a fingerprint that encodes the 3D structure of a molecule

The next two sections describe the *Fingerprint format* and *Spectrophores* in depth. For the others, see the relevant sections listed above.

6.1 Fingerprint format

The *Fingerprint format (fpt)* is a utility file format that provides access to a number of substructure-based fingerprints, and that enables the user to carry out similarity and substructure searching. You can see the available fingerprints using the following command:

```
$ babel -L fingerprints
FP2    Indexes linear fragments up to 7 atoms.
FP3    SMARTS patterns specified in the file patterns.txt
FP4    SMARTS patterns specified in the file SMARTS_InteLigand.txt
MACCS    SMARTS patterns specified in the file MACCS.txt
```

At present there are four types of fingerprints:

- **FP2**, a path-based fingerprint which indexes small molecule fragments based on linear segments of up to 7 atoms (somewhat similar to the Daylight fingerprints):

  > A molecule structure is analysed to identify linear fragments of length from 1-7 atoms. Single atom fragments of C, N, and O are ignored. A fragment is terminated when the atoms form a ring.

For each of these fragments the atoms, bonding and whether they constitute a complete ring is recorded and saved in a set so that there is only one of each fragment type. Chemically identical versions, (i.e. ones with the atoms listed in reverse order and rings listed starting at different atoms) are identified and only a single canonical fragment is retained.

Each remaining fragment is assigned a hash number from 0 to 1020 which is used to set a bit in a 1024 bit vector

- **FP3** uses a series of SMARTS queries stored in `patterns.txt`
- **FP4** uses a series of SMARTS queries stored in `SMARTS_InteLigand.txt`
- **MACCS** uses the SMARTS patterns in `MACCS.txt`

Note: Note that you can tailor the latter three fingerprints to your own needs by adding your own SMARTS queries to these files. On UNIX and Mac systems, these files are frequently found in `/usr/local/share/openbabel` under a directory for each version of Open Babel.

See also:

The sections on the *fingerprint* and *fastsearch* formats contain additional detail.

6.1.1 Similarity searching

Small datasets

For relatively small datasets (<10,000's) it is possible to do similarity searches without the need to build a similarity index, however larger datasets (up to a few million) can be searched rapidly once a fastsearch index has been built.

On small datasets these fingerprints can be used in a variety of ways. The following command gives you the Tanimoto coefficient between a SMILES string in `mysmiles.smi` and all the molecules in `mymols.sdf`:

```
babel  mysmiles.smi  mymols.sdf -ofpt

MOL_00000067   Tanimoto from first mol = 0.0888889
MOL_00000083   Tanimoto from first mol = 0.0869565
MOL_00000105   Tanimoto from first mol = 0.0888889
MOL_00000296   Tanimoto from first mol = 0.0714286
MOL_00000320   Tanimoto from first mol = 0.0888889
MOL_00000328   Tanimoto from first mol = 0.0851064
MOL_00000338   Tanimoto from first mol = 0.0869565
MOL_00000354   Tanimoto from first mol = 0.0888889
MOL_00000378   Tanimoto from first mol = 0.0816327
MOL_00000391   Tanimoto from first mol = 0.0816327
11 molecules converted
```

The default fingerprint used is the FP2 fingerprint. You change the fingerprint using the `f` output option as follows:

```
babel mymols.sdf -ofpt -xfFP3
```

The `-s` option of **`babel`** is used to filter by SMARTS string. If you wanted to know the similarity only to the substituted bromobenzenes in `mymols.sdf` then you might combine commands like this (note: if the query molecule does not match the SMARTS string this will not work as expected, as the first molecule in the database that matches the SMARTS string will instead be used as the query):

```
babel mysmiles.smi mymols.sdf -ofpt -s c1ccccc1Br

MOL_00000067   Tanimoto from first mol = 0.0888889
MOL_00000083   Tanimoto from first mol = 0.0869565
MOL_00000105   Tanimoto from first mol = 0.0888889
```

If you don't specify a query file, **babel** will just use the first molecule in the database as the query:

```
babel mymols.sdf  -ofpt

MOL_00000067
MOL_00000083   Tanimoto from MOL_00000067 = 0.810811
MOL_00000105   Tanimoto from MOL_00000067 = 0.833333
MOL_00000296   Tanimoto from MOL_00000067 = 0.425926
MOL_00000320   Tanimoto from MOL_00000067 = 0.534884
MOL_00000328   Tanimoto from MOL_00000067 = 0.511111
MOL_00000338   Tanimoto from MOL_00000067 = 0.522727
MOL_00000354   Tanimoto from MOL_00000067 = 0.534884
MOL_00000378   Tanimoto from MOL_00000067 = 0.489362
MOL_00000391   Tanimoto from MOL_00000067 = 0.489362
10 molecules converted
```

Large datasets

On larger datasets it is necessary to first build a fastsearch index. This is a new file that stores a database of fingerprints for the files indexed. You will still need to keep both the new .fs fastsearch index and the original files. However, the new index will allow significantly faster searching and similarity comparisons. The index is created with the following command:

```
babel mymols.sdf -ofs
```

This builds `mymols.fs` with the default fingerprint (unfolded). The following command uses the index to find the 5 most similar molecules to the molecule in `query.mol`:

```
babel mymols.fs results.sdf -squery.mol -at5
```

or to get the matches with Tanimoto>0.6 to 1,2-dicyanobenzene:

```
babel mymols.fs results.sdf -sN#Cc1ccccc1C#N -at0.6
```

6.1.2 Substructure searching

Small datasets

This command will find all molecules containing 1,2-dicyanobenzene and return the results as SMILES strings:

```
babel mymols.sdf -sN#Cc1ccccc1C#N results.smi
```

If all you want output are the molecule names then adding `-xt` will return just the molecule names:

```
babel mymols.sdf -sN#Cc1ccccc1C#N results.smi -xt
```

The parameter of the -s option in these examples is actually SMARTS, which allows a richer matching specification, if required. It does mean that the aromaticity of atoms and bonds is significant; use [#6] rather than C to match both aliphatic and aromatic carbon.

The -s option's parameter can also be a file name with an extension. The file must contain a molecule, which means only substructure matching is possible (rather than full SMARTS). The matching is also slightly more relaxed with respect to aromaticity.

Large datasets

First of all, you need to create a fastsearch index (see above). The index is created with the following command:

```
babel mymols.sdf -ofs
```

Substructure searching is as for small datasets, except that the fastsearch index is used instead of the original file. This command will find all molecules containing 1,2-dicyanobenzene and return the results as SMILES strings:

```
babel mymols.fs -ifs -sN#Cc1ccccc1C#N results.smi
```

If all you want output are the molecule names then adding -xt will return just the molecule names:

```
babel mymols.fs -ifs -sN#Cc1ccccc1C#N results.smi -xt
```

6.1.3 Case study: Search ChEMBLdb

This case study uses a combination of the techniques described above for similarity searching using large databases and using small databases. Note that we are using the default fingerprint for all of these analyses. The default fingerprint is FP2, a path-based fingerprint (somewhat similar to the Daylight fingerprints).

(1) Download Version 2 of ChEMBLdb from ftp://ftp.ebi.ac.uk/pub/databases/chembl/ChEMBLdb/releases/.

(2) After unzipping it, make a fastsearch index (this took 18 minutes on my machine for the 500K+ molecules):

```
babel chembl_02.sdf -ofs
```

(3) Let's use the first molecule in the sdf file as a query. Using Notepad (or on Linux, head -79 chembl_02.sdf) extract the first molecule and save it as first.sdf. Note that the molecules in the ChEMBL sdf do not have titles; instead, their IDs are stored in the "chebi_id" property field.

(4) This first molecule is 100183. Check its ChEMBL page. It's pretty weird, but is there anything similar in ChEMBLdb? Let's find the 5 most similar molecules:

```
babel chembl_02.fs mostsim.sdf -s first.sdf -at5
```

(5) The results are stored in mostsim.sdf, but how similar are these molecules to the query?:

```
babel first.sdf mostsim.sdf -ofpt
>
>   Tanimoto from first mol = 1
Possible superstructure of first mol
>   Tanimoto from first mol = 0.986301
>   Tanimoto from first mol = 0.924051
Possible superstructure of first mol
>   Tanimoto from first mol = 0.869048
Possible superstructure of first mol
```

(continues on next page)

(continued from previous page)

```
>   Tanimoto from first mol = 0.857143
6 molecules converted
76 audit log messages
```

(6) That's all very well, but it would be nice to show the ChEBI IDs. Let's set the title field of `mostsim.sdf` to the content of the "chebi_id" property field, and repeat step 5:

```
babel mostsim.sdf mostsim_withtitle.sdf --append "chebi_id"
babel first.sdf mostsim_withtitle.sdf -ofpt
>
>100183   Tanimoto from first mol = 1
Possible superstructure of first mol
>124893   Tanimoto from first mol = 0.986301
>206983   Tanimoto from first mol = 0.924051
Possible superstructure of first mol
>207022   Tanimoto from first mol = 0.869048
Possible superstructure of first mol
>607087   Tanimoto from first mol = 0.857143
6 molecules converted
76 audit log messages
```

(7) Here are the ChEMBL pages for these molecules: 100183, 124893, 206983, 207022, 607087. I think it is fair to say that they are pretty similar. In particular, the output states that 206983 and 207022 are possible superstructures of the query molecule, and that is indeed true.

(8) How many of the molecules in the dataset are superstructures of the molecule in `first.sdf`? To do this and to visualize the large numbers of molecules produced, we can output to SVG format (see *SVG 2D depiction (svg)*):

```
obabel chembl_02.fs  -O out.svg  -s first.sdf
```

Note that **`obabel`** has been used here because of its more flexible option handling.

This command does a substructure search and puts the 47 matching structures in the file `out.svg`. This can be viewed in a browser like Firefox, Opera or Chrome (but not Internet Explorer). The display will give an overall impression of the set of molecules but details can be seen by zooming in with the mousewheel and panning by dragging with a mouse button depressed.

(9) The substructure that is being matched can be highlighted in the output molecules by adding another parameter to the `-s` option. Just for variety, the display is also changed to a black background, 'uncolored' (no element-specific coloring), and terminal carbon not shown explicitly. (Just refresh your browser to see the modified display.)

```
obabel chembl_02.fs  -O out.svg  -s first.sdf green  -xb -xu -xC
```

This highlighting option also works when the `-s` option is used without fastsearch on small datasets.

(10) The substructure search here has two stages. The indexed fingerprint search quickly produces 62 matches from the 500K+ molecules in the dataset. Each of these is then checked by a slow detailed isomorphism check. There are 15 false positives from the fingerprint stage. These are of no significance, but you can see them using:

```
obabel chembl_02.fs  -O out.svg  -s ~first.sdf
```

The fingerprint search is unaffected but the selection in the second stage is inverted.

6.2 Spectrophores™

6.2.1 Introduction

Spectrophores[1] are one-dimensional descriptors generated from the property fields surrounding the molecules. This technology allows the accurate description of molecules in terms of their surface properties or fields. Comparison of molecules' property fields provides a robust structure-independent method of aligning actives from different chemical classes. When applied to molecules such as ligands and drugs, Spectrophores can be used as powerful molecular descriptors in the fields of chemoinformatics, virtual screening, and QSAR modeling.

Commercial Support for Spectrophores

Commercial support for Spectrophores is available from Silicos NV, the developers of the Spectrophore technology.

Silicos is a fee-for-service company empowering open source chemo-informatics virtual screening technologies for the discovery of novel lead compounds and database characterization. Silicos fully endorses the concept of open innovation and open source software development, and provides its clients with a wide variety of computational chemistry-based lead discovery services, including Open Babel support, training and code development. Please visit Silicos for more details.

The computation of Spectrophores is independent of the position and orientation of the molecule and this enables easy and fast comparison of Spectrophores between different molecules. Molecules having similar three-dimensional properties and shapes always yield similar Spectrophores. A Spectrophore is calculated by surrounding the three-dimensional conformation of the molecule by a three-dimensional arrangement of points, followed by calculating the interaction between each of the atom properties and the surrounding the points. The three-dimensional arrangement of the points surrounding the molecule can be regarded as an 'artificial' cage or receptor, and the interaction calculated between the molecule and the cage can be regarded as an artificial representation of an affinity value between molecule and cage. Because the calculated interaction is dependent on the relative orientation of the molecule within the cage, the molecule is rotated in discrete angles and the most favorable interaction value is kept as final result. The angular stepsize at which the molecule is rotated along its three axis can be specified by the user and influences the accuracy of the method.

The Spectrophore code was developed by Silicos NV, and donated to the OpenBabel project in July 2010 (see sidebar for information on commercial support). Spectrophores can be generated either using the command-line application **obspectrophore** (see next section) or through the API (`OBSpectrophore`, as described in the **:obapi:'API documentation <OBSpectrophore>'**).

6.2.2 obspectrophore

Usage

```
obspectrophore -i <input file> [options]
```

Parameter details

-i <input file> *Specify the input file*

[1] Spectrophore is a registered trademark of Silicos NV.

Spectrophores will be calculated for each molecule in the input file. The filetype is automatically detected from the file extension.

-n <type> *The type of normalization that should be performed*

Valid values are (without quotes):

- No (default)
- ZeroMean
- UnitStd
- ZeroMeanAndUnitStd

-a <accuracy> *The required accuracy expressed as the angular stepsize*

Only the following discrete values are allowed: 1, 2, 5, 10, 15, 20 (default), 30, 36, 45, 60

-s <type> *The kind of cages that should be used*

The cage type is specified in terms of the underlying pointgroup: P1 or P-1. Valid values are (without quotes):

- No (default)
- Unique
- Mirror
- All

-r <resolution> *The required resolution expressed as a real positive number*

The default value is 3.0 Angstrom. Negative values or a value of 0 generates an error message.

-h *Displays help*

6.2.3 Implementation

Atomic properties

The calculation of a Spectrophore™ starts by calculating the atomic contributions of each property from which one wants to calculate a Spectrophore. In the current implementation, four atomic properties are converted into a Spectrophore; these four properties include the atomic partial charges, the atomic lipophilicities, the atomic shape deviations and the atomic electrophilicities. The atomic partial charges and atomic electrophilicity properties are calculated using the electronegativity equalisation method (EEM) as described by Bultinck and coworkers [bll2002] [blc2003]. Atomic lipophilic potential parameters are calculated using a rule-based method. Finally, the atomic shape deviation is generated by calculating, for each atom, the atom's deviation from the average molecular radius. This is done in a four step process:

- The molecular center of geometry (COG) is calculated
- The distances between each atom and the molecular COG are calculated
- The average molecular radius is calculated by averaging all the atomic distances
- The distances between each atom and the COG are then divided by the average molecular radius and centered on zero

Interaction between the atoms and cage points

Following the calculation of all required atomic properties, the next step in the calculation of a Spectrophore consists of determining the total interaction value V(c,p) between each of the atomic contributions of property p with a set of interaction points on an artificial cage c surrounding the molecular conformation.

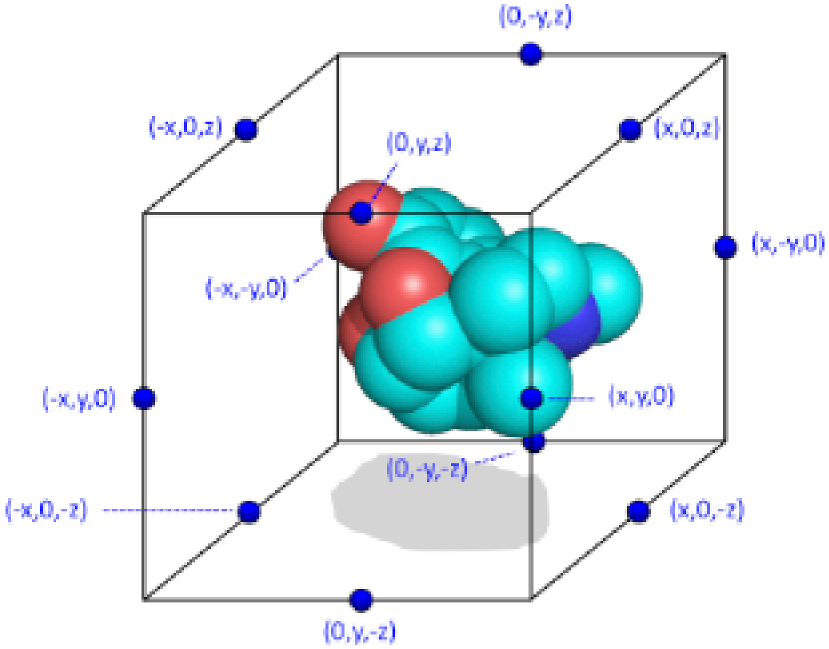


Fig. 1: **Schematic representation of a molecule surrounded by the artifical cage**

For this purpose, each of these interaction points i on cage c is assigned a value P(c,i) which is either +1 or -1, with the constraint that the sum of all interaction points on a particular cage should be zero. In a typical Spectrophore calculation, a cage is represented as a rectangular box encompassing the molecular conformation in all three dimensions, with the centers of the box edges being the interaction points. Such a configuration gives twelve interaction points per cage, and, in the case of a non-stereospecific distribution of the interaction points, leads to 12 different cages. Although there are no particular requirements as to the dimensions of the rectangular cage, the distance between the interaction points and the geometrical extremes of the molecule should be such that a meaningful interaction value between each cage point and the molecular entity can be calculated. In this respect, the default dimensions of the cage are constantly adjusted to enclose the molecule at a minimum distance of 3 A along all dimensions. This cage size can be modified by the user and influences the resolution of the Spectrophore.

The total interaction value V(c,p) between the atomic contribution values A(j,p) of property p for a given molecular conformation and the cage interaction values P(c,i) for a given cage c is calculated according a standard interaction energy equation. It takes into account the Euclidean distance between each atom and each cage point. This total interaction V(c,p) for a given property p and cage c for a given molecular conformation is minimized by sampling the molecular orientation along the three axis in angular steps and the calculation of the interaction value for each orientation within the cage.

The final total interaction V(c,p) for a given cage c and property p corresponds to the lowest interaction value obtained this way, and corresponds to the c'th value in the one-dimensional Spectrophore vector calculated for molecular property p. As a result, a Spectrophore is organized as a vector of minimized interaction values V, each of these organized in order of cages and property values. Since for a typical Spectrophore implementation twelve different cages are used, the total length of a Spectrophore vector equals to 12 times the number of properties. Since four different properties are used in the current implementation (electrostatic, lipophilic, electrophilic potentials, and an additional shape index as described before), this leads to a total Spectrophore length of 48 real values per molecular conformation.

Since Spectrophore descriptors are dependent on the actual three-dimensional conformation of the molecule, a typical analysis includes the calculation of Spectrophores from a reasonable set of different conformations. It is then up to the user to decide on the most optimal strategy for processing the different Spectrophore vectors. In a typical virtual screening application, calculating the average Spectrophore vector from all conformations of a single molecule may be a good strategy; other applications have benefit from calculating a weighted average or the minimal values. For each molecule in the input file, a Spectrophore is calculated and printed to standard output as a vector of 48 numbers (in the case of a non-stereospecific Spectrophore. The 48 doubles are organised into 4 sets of 12 doubles each:

- numbers 01-11: Spectrophore values calculated from the atomic partial charges;
- numbers 13-24: Spectrophore values calculated from the atomic lipophilicity properties;
- numbers 25-36: Spectrophore values calculated from the atomic shape deviations;
- numbers 37-48: Spectrophore values calculated from the atomic electrophilicity properties;

6.2.4 Choice of Parameters

Accuracy

As already mentioned, the total interaction between cage and molecule for a given property is minimized by sampling the molecular orientation in angular steps of a certain magnitude. As a typical angular step size, 20 degrees was found to be the best compromise between accuracy and computer speed. Larger steps sizes are faster to calculate but have the risk of missing the global interaction energy minimum, while smaller angular steps sizes do sample the rotational space more thoroughly but at a significant computational cost. The accuracy can be specified by the user using the `-a` option.

Resolution

Spectrophores capture information about the property fields surrounding the molecule, and the amount of detail that needs to be captured can be regulated by the user. This is done by altering the minimal distance between the molecule and the surrounding cage. The resolution can be specified by the user with the `-r` option. The default distance along all dimensions is 3.0 Angstrom. The larger the distance, the lower the resolution.

With a higher resolution, more details of the property fields surrounding the molecule are contained by the Spectrophore. On the other hand, low resolution settings may lead to a more general representation of the property fields, with little or no emphasis on small local variations within the fields. Using a low resolution can be the method of choice during the initial virtual screening experiments in order to get an initial, but not so discriminative, first selection. This initial selection can then further be refined during subsequent virtual screening steps using a higher resolution. In this setting, small local differences in the fields between pairs of molecules will be picked up much more easily.

The absolute values of the individual Spectrophore data points are dependent on the used resolution. Low resolution values lead to small values of the calculated individual Spectrophore data points, while high resolutions will lead to larger data values. It is therefore only meaningful to compare only Spectrophores that have been generated using the same resolution settings or after some kind of normalization is performed. Computation time is not influenced by the specified resolution and hence is identical for all different resolution settings.

Stereospecificity

Some of the cages that are used to calculated Spectrophores have a stereospecific distribution of the interaction points. The resulting interaction values resulting from these cages are therefore sensitive to the enantiomeric configuration of the molecule within the cage. The fact that both stereoselective as well as stereo non-selective cages can be used makes it possible to include or exclude stereospecificity in the virtual screening search. Depending on the desired output, the stereospecificity of Spectrophores can be specified by the user using the `-s` option:

- **No stereospecificity (default):** Spectrophores are generated using cages that are not stereospecific. For most applications, these Spectrophores will suffice.
- **Unique stereospecificity:** Spectrophores are generated using unique stereospecific cages.
- **Mirror stereospecificity:** Mirror stereospecific Spectrophores are Spectrophores resulting from the mirror enantiomeric form of the input molecules.

The differences between the corresponding data points of unique and mirror stereospecific Spectrophores are very small and require very long calculation times to obtain a sufficiently high quality level. This increased quality level is triggered by the accuracy setting and will result in calculation times being increased by at least a factor of 100. As a consequence, it is recommended to apply this increased accuracy only in combination with a limited number of molecules, and when the small differences between the stereospecific Spectrophores are really critical. However, for the vast majority of virtual screening applications, this increased accuracy is not required as long as it is not the intention to draw conclusions about differences in the underlying molecular stereoselectivity. Non-stereospecific Spectrophores will therefore suffice for most applications.

Normalisation

It may sometimes be desired to focus on the relative differences between the Spectrophore data points rather than focussing on the absolute differences. In these cases, normalization of Spectrophores may be required. The current implementation offers with the `-n` option the possibility to normalize in four different ways:

- No normalization (default)
- Normalization towards zero mean
- Normalization towards standard deviation
- Normalization towards zero mean and unit standard deviation

In all these cases, normalization is performed on a ‘per-property’ basis, which means that the data points belonging to the same property set are treated as a single set and that normalization is only performed on the data points within each of these sets and not across all data points.

Normalization may be important when comparing the Spectrophores of charged molecules with those of neutral molecules. For molecules carrying a global positive charge, the resulting Spectrophore data points of the charge and electrophilicity properties will both be shifted in absolute value compared to the corresponding data points of the respective neutral species. Normalization of the Spectrophores removes the original magnitude differences for the data points corresponding to the charge and electrophilicity properties of charged and neutral species. Therefore, if the emphasis of the virtual screening consists of the identification of molecules with similar property fields without taking into account differences in absolute charge, then Spectrophores should be normalized towards zero mean. However, if absolute charge differences should be taken into account to differentiate between molecules, unnormalized Spectrophores are recommended.

Chapter 7

obabel vs Chemistry Toolkit Rosetta

The Chemistry Toolkit Rosetta is the brainchild of Andrew Dalke. It is a website that illustrates how to program various chemical toolkits to do a set of tasks. To make it easily understandable, these tasks are probably on the simpler side of those in the real world. The Rosetta already contains several examples of using the Open Babel Python bindings to carry out tasks.

Here we focus on the use of the command line application **obabel** to accomplish the tasks listed in the Rosetta. Inevitably we will struggle with more complicated tasks; however this section is intended to show how far you can go simply using **obabel**, and to illustrate some of its less common features. Some of the tasks cannot be done exactly as specified, but they are are usually close enough to useful.

Note that except for the examples involving piping, the GUI could also be used. Also the copy output format at present works only for files with Unix line endings.

7.1 Heavy atom counts from an SD file

> *For each record from the benzodiazepine file, print the total number of heavy atoms in each record (that is, exclude hydrogens). The output is one output line per record, containing the count as an integer. If at all possible, show how to read directly from the gzip'ed input SD file.*

```
obabel benzodiazepine.sdf.gz -otxt --title "" --append atoms -d -l5
```

The *txt format* outputs only the title but we set that to nothing and then append the result. The *atoms* descriptor counts the number of atoms after the `-d` option has removed the hydrogens. The `-l5` limits the output to the first 5 molecules, in case you really didn't want to print out results for all 12386 molecules.

7.2 Convert a SMILES string to canonical SMILES

> *Parse two SMILES strings and convert them to canonical form. Check that the results give the same string.*

```
obabel -:"CN2C(=O)N(C)C(=O)C1=C2N=CN1C" -:"CN1C=NC2=C1C(=O)N(C)C(=O)N2C" -ocan
```

giving:

```
Cn1cnc2c1c(=O)n(C)c(=O)n2C
Cn1cnc2c1c(=O)n(C)c(=O)n2C
2 molecules converted
```

7.3 Report how many SD file records are within a certain molecular weight range

> *Read the benzodiazepine file and report the number of records which contain a molecular weight between 300 and 400.*

```
obabel benzodiazepine.sdf.gz -onul --filter "MW>=300 MW<=400"
3916 molecules converted
```

7.4 Convert SMILES file to SD file

> *Convert a SMILES file into an SD file. The conversion must do its best to use the MDL conventions for the SD file, including aromaticity perception. Note that the use of aromatic bond types in CTABs is only allowed for queries, so aromatic structures must be written in a Kekule form. Because the stereochemistry of molecules in SD files is defined solely by the arrangement of atoms, it is necessary to assign either 2D or 3D coordinates to the molecule before generating output. The coordinates do not have to be reasonable (i.e. it's ok if they would make a chemist scream in horror), so long as the resulting structure is chemically correct.*

```
obabel infile.smi -O outfile.sdf --gen3D
```

7.5 Report the similarity between two structures

> *Report the similarity between "CC(C)C=CCCCCC(=O)NCc1ccc(c(c1)OC)O" (PubChem CID 1548943) and "COC1=C(C=CC(=C1)C=O)O" (PubChem CID 1183).*

Two types of fingerprint are used: the default FP2 path-based one, and FP4 which is structure key based:

```
obabel -:"CC(C)C=CCCCCC(=O)NCc1ccc(c(c1)OC)O" -:"COC1=C(C=CC(=C1)C=O)O" -ofpt
Tanimoto from first mol = 0.360465

obabel -:"CC(C)C=CCCCCC(=O)NCc1ccc(c(c1)OC)O" -:"COC1=C(C=CC(=C1)C=O)O" -ofpt
       -xfFP4
Tanimoto from first mol = 0.277778
```

7.6 Find the 10 nearest neighbors in a data set

> *The data will come from the gzip'ed SD file of the benzodiazepine data set. Use the first structure as the query structure, and use the rest of the file as the targets to find the 10 most similar structures. The output is sorted by similarity, from most similar to least. Each target match is on its own line, and the line contains the similarity score in the first column in the range 0.00 to 1.00 (preferably to 2 decimal places), then a space, then the target ID, which is the title line from the SD file.*

A fastsearch index, using the default FP2 fingerprint, is prepared first:

```
obabel benzodiazepine.sdf -ofs
```

The query molecule (first in the file) is extracted:

```
obabel benzodiazepine.sdf -O first.sdf -l1
```

The similarity search of the index file for the 10 most similar molecules is done. The output is to *Title format (txt)*, with the `-aa` option of *Fastsearch format (fs)* adding the Tanimoto score:

```
obabel benzodiazepine.fs -otxt -s first.sdf -at 10 -aa

623918 1
450820 1
1688 1
20351792 0.993007
9862446 0.986111
398658 0.97931
398657 0.97931
6452650 0.978873
450830 0.978873
3016 0.978873
10 molecules converted
```

The Tanimoto coefficient comes second, rather than first as requested and is not formatted to two decimal places, but the information is still there.

7.7 Depict a compound as an image

> *Depict the SMILES "CN1C=NC2=C1C(=O)N(C(=O)N2C)C" as an image of size 200x250 pixels. The image should be in PNG format if possible, otherwise in GIF format. If possible, give it the title "Caffeine". It should display the structure on a white background.*

Open Babel can output 2D structures as *PNG*. The `-d` makes hydrogen implicit. Width and height are set with the -xw and -xh options.:

```
obabel -:"CN1C=NC2=C1C(=O)N(C(=O)N2C)C Caffeine" -O out.png -xw 200 -xh 250 -d
```

Open Babel also supports outputting *SVG*, which is resolution independent as a vector format.:

```
obabel -:"CN1C=NC2=C1C(=O)N(C(=O)N2C)C Caffeine" -O out.svg -d
```

7.8 Highlight a substructure in the depiction

> *Read record 3016 from the benzodiazepine SD file. Find all atoms which match the SMARTS "c1ccc2c(c1)C(=NCCN2)c3ccccc3" and highlight them in red. All other atoms must be drawn in black.*
>
> *The resulting image should be 200x250 pixels and on a white background. The resulting image file should be in PNG (preferred) or GIF format.*

```
obabel benzodiazepine.sdf.gz -O out.png --filter "title=3016"
       -s "c1ccc2c(c1)C(=NCCN2)c3ccccc3 red" -xu -xw 200 -xh 250 -d
```

Open Babel can output 2D structures as *PNG*. The compressed data file can be used as input. The `-d` makes hydrogen implicit and the `-xu` removes the element-specific coloring. Width and height are set with the -xw and -xh options.

This is slow (about a minute) because each molecule is fully interpreted, although in most cases only the title is required. The task can be done 10 times faster by using the uncompressed file, converting only the title (the `-aT` option) and copying the SD text to standard out when a match occurs. This is piped to a second command which outputs the structure.:

```
obabel benzodiazepine.sdf -ocopy --filter "title=3016" -aT |
       obabel -isdf -O out.png -s "c1ccc2c(c1)C(=NCCN2)c3ccccc3 red" -xu -xw 200 -xh␣
↪250 -d
```

Open Babel also supports outputting *SVG*, which is resolution independent as a vector format.:

```
obabel benzodiazepine.sdf.gz -O out.svg --filter "title=3016"
       -s "c1ccc2c(c1)C(=NCCN2)c3ccccc3 red" -xu -d

obabel benzodiazepine.sdf -ocopy --filter "title=3016" -aT |
       obabel -isdf -O out.svg -s "c1ccc2c(c1)C(=NCCN2)c3ccccc3 red" -xu -d
```

7.9 Align the depiction using a fixed substructure

> *Use the first 16 structures of the benzodiazepine SD file to make a 4x4 grid of depictions as a single image. The first structure is in the upper-left corner, the second is to its right, and so on. Each depiction should include the title field of the corresponding record, which in this case is the PubChem identifier.*
>
> *Use "[#7]~1~[#6]~[#6]~[#7]~[#6]~[#6]~2~[#6]~[#6]~[#6]~[#6]~[#6]12" as the common SMARTS substructure. This is the fused ring of the benzodiazepine system but without bond type or atom aromaticity information. Use the first molecule as the reference depiction. All other depictions must have the depiction of their common substructure aligned to the reference.*

Since Open Babel 2.3.1 this can be done in one line:

```
obabel benzodiazepine.sdf.gz -O out.png -l16 --align -d -xu -xw 400 -xh 400
       -s"[#7]~1~[#6]~[#6]~[#7]~[#6]~[#6]~2~[#6]~[#6]~[#6]~[#6]~[#6]12 green"
```

The depiction has some cosmetic tweaks: the substructure is highlighted in green; `-d` removes hydrogen; `-xu` removes the element specific coloring. Open Babel also supports outputting *SVG*, which is resolution independent as a vector format.:

```
obabel benzodiazepine.sdf.gz -O out.svg -l16 --align -d -xu
       -s"[#7]~1~[#6]~[#6]~[#7]~[#6]~[#6]~2~[#6]~[#6]~[#6]~[#6]~[#6]12 green"
```

In earlier versions the **`obfit`** program can be used. First extract the first molecule for the reference and the first 16 to be displayed:

```
obabel benzodiazepine.sdf.gz -O firstbenzo.sdf -l1
obabel benzodiazepine.sdf.gz -O sixteenbenzo.sdf -l16
```

Then use the program **`obfit`**, which is distributed with Open Babel:

```
obfit "[#7]~1~[#6]~[#6]~[#7]~[#6]~[#6]~2~[#6]~[#6]~[#6]~[#6]~[#6]12"
      firstbenzo.sdf  sixteenbenzo.sdf > 16out.sdf
```

Display the 16 molecules (with implicit hydrogens) as *SVG* (earlier versions of Open Babel do not support *PNG*):

```
obabel 16out.sdf -O out.png -d -xw 400 -xh 400
```

7.10 Perform a substructure search on an SDF file and report the number of false positives

> *The sample database will be gzip'ed SD file of the benzodiazepine data set. The query structure will be defined as "C1C=C(NC=O)C=CC=1".*

The default FP2 fingerprint is sensitive to whether a bond is aromatic or not. So this Kekule structure needs to be converted to its aromatic form. As this happens automatically on conversion, the easiest way is to store the SMILES string in a file, and use this file to specify the search pattern.

Prepare an index (of the unzipped data file):

```
obabel benzodiazepine.sdf -ofs
```

Do the substructure search. A very large number of molecules match the query, so the maximum number of hits has to be increased with the `-al 9000` option. By virtue of the ~ it is the false positives that are output (to nowhere) but their number is reported:

```
obabel benzodiazepine.fs -onul -s ~substruct.smi -al 9000
8531 candidates from fingerprint search phase
12 molecules converted
```

7.11 Calculate TPSA

> *The goal of this task is get an idea of how to do a set of SMARTS matches when the data comes in from an external table.*
>
> *Write a function or method named "TPSA" which gets its data from the file "tpsa.tab". The function should take a molecule record as input, and return the TPSA value as a float. Use the function to calculate the TPSA of "CN2C(=O)N(C)C(=O)C1=C2N=CN1C". The answer should be 61.82, which agrees exactly with Ertl's online TPSA tool but not with PubChem's value of 58.4.*

Open Babel's command line cannot parse tables with custom formats. But the TPSA descriptor, defined by a table in the file `psa.txt`, is already present and can be used as follows:

```
obabel -:CN2C(=O)N(C)C(=O)C1=C2N=CN1C -osmi --append TPSA
```

giving:

```
Cn1c(=O)n(C)c(=O)c2c1ncn2C      61.82
1 molecule converted
```

The table in `tpsa.tab` and Open Babel's `psa.txt` have the same content but different formats. The first few rows of `tpsa.tab` are:

```
psa   SMARTS  description
23.79 [N0;H0;D1;v3]   N#
23.85 [N+0;H1;D1;v3]  [NH]=
26.02 [N+0;H2;D1;v3]  [NH2]-
```

and the equivalent lines from Open Babel's `psa.txt`:

```
[N]#* 23.79
[NH]=*        23.85
[NH2]-*       26.02
```

It is possible to add new descriptors without having to recompile. If another property, *myProp*, could be calculated using a table in `myprop.txt` with the same format as `psa.txt`, then a descriptor could set up by adding the following item to `plugindefines.txt`:

```
OBGroupContrib
myProp          # name of descriptor
myprop.txt      # data file
Coolness index  # brief description
```

The following would then output molecules in increasing order of *myProp* with the value added to the title:

```
obabel infile.smi -osmi --sort myProp+
```

7.12 Working with SD tag data

> *The input file is SD file from the benzodiazepine data set. Every record contains the tags PUBCHEM_CACTVS_HBOND_DONOR, PUBCHEM_CACTVS_HBOND_ACCEPTOR and PUBCHEM_MOLECULAR_WEIGHT, and most of the records contain the tag PUBCHEM_XLOGP3.*
>
> *The program must create a new SD file which is the same as the input file but with a new tag data field named "RULE5". This must be "1" if the record passes Lipinski's rule, "0" if it does not, and "no logP" if the PUBCHEM_XLOGP3 field is missing.*

This exercise is a bit of a stretch for the Open Babel command-line. However, the individual lines may be instructional, since they are more like the sort of task that would normally be attempted.

```
obabel benzodiazepine.sdf.gz -O out1.sdf --filter "PUBCHEM_CACTVS_HBOND_DONOR<=5 &
       PUBCHEM_CACTVS_HBOND_ACCEPTOR<=10 & PUBCHEM_MOLECULAR_WEIGHT<=500 &
       PUBCHEM_XLOGP3<=5"
       --property "RULE5" "1"

obabel benzodiazepine.sdf.gz -O out2.sdf --filter "!PUBCHEM_XLOGP3"
       --property "RULE5" "no logP"

obabel benzodiazepine.sdf.gz -O out3.sdf --filter "!PUBCHEM_XLOGP3 &
       !(PUBCHEM_CACTVS_HBOND_DONOR<=5 & PUBCHEM_CACTVS_HBOND_ACCEPTOR<=10 &
       PUBCHEM_MOLECULAR_WEIGHT<=500 & PUBCHEM_XLOGP3<=5)"
       --property "RULE5" "0"
```

The first command converts only molecules passing Lipinski's rule, putting them in `out1.sdf`, and adding an additional property, *RULE5*, with a value of `1`.

The second command converts only molecules that do not have a property *PUBCHEM_XLOGP3*.

The third command converts only molecules that do have a *PUBCHEM_XLOGP3* and which fail Lipinski's rule.

Use **`cat`** or **`type`** at the command prompt to concatenate the three files `out1.sdf`, `out2.sdf`, `out3.sdf`.

These operations are slow because the chemistry of each molecule is fully converted. As illustrated below, the filtering alone could have been done more quickly using the uncompressed file and the `-aP` option, which restricts the reading of the SDF file to the title and properties only, and then copying the molecule's SDF text verbatim with `-o copy`. But adding the additional property is not then possible:

```
obabel benzodiazepine.sdf -o copy -O out1.sdf -aP --filter
       "PUBCHEM_CACTVS_HBOND_DONOR<=5 & PUBCHEM_CACTVS_HBOND_ACCEPTOR<=10 &
       PUBCHEM_MOLECULAR_WEIGHT<=500 & PUBCHEM_XLOGP3<=5"
```

7.13 Unattempted tasks

A number of the Chemical Toolkit Rosetta tasks cannot be attempted as the **obabel** tool does not (currently!) have the necessary functionality. These include the following:

- Detect and report SMILES and SDF parsing errors
- Ring counts in a SMILES file
- Unique SMARTS matches against a SMILES string
- Find the graph diameter
- Break rotatable bonds and report the fragments
- Change stereochemistry of certain atoms in SMILES file

To handle these tasks, you need to use the Open Babel library directly. This is the subject of the next section.

Chapter 8

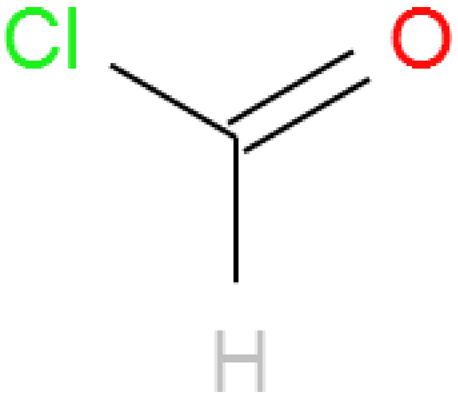
2D Depiction

As the old Chinese proverb has it, a molecular depiction is worth a thousand words. This chapter covers everything relevant to using Open Babel to generate or read/write a 2D depiction, expected by most chemists for print or website purposes.

When we talk about a depiction in cheminformatics, there are really two different concepts covered by this term:

1. Graphical display of a molecule's structure as a 2D image (such as the PNG and SVG formats). Here is an example:

```
obabel -:C(=O)Cl -O acidchloride.png
```



2. Storage of the 2D coordinates (and associated stereo symbols) associated with Concept 1 (using formats such

as Mol and Mol2). Here is the connection table from the corresponding Mol file for the above depiction:

```
3  2  0  0  0  0  0  0  0  0999 V2000
  0.8660   -0.5000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  1.7321   -0.0000    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
  0.0000    0.0000    0.0000 Cl  0  0  0  0  0  0  0  0  0  0  0  0
1  2  2  0  0  0  0
1  3  1  0  0  0  0
```

Note: The focus in this chapter is on 2D depiction and not 3D. It is of course possible to generate and store 3D coordinates in many of the file formats supported by Open Babel, but the only support for depiction is the Povray format, used to create ray-traced ball-and-stick diagrams of molecules. Other Open Source chemistry projects such as Avogadro, PyMOL, and Jmol cover this area very well.

8.1 Molecular graphics

As of Open Babel 2.3.2, there are three output formats for displaying a 2D image:

1. PNG format: This is a bitmap format used to create images of a certain pixel width. These images can be inserted into Word documents or displayed on web pages.
2. SVG format: This is a vector format, which can be scaled to generate images of any size without loss of quality. In particular, Open Babel's SVG images can be interactively zoomed and panned using a modern web browser.
3. ASCII format: This is a depiction of a molecule using ASCII text. This can be useful if you are logged into a remote server, or are working at the command-line, and just need a basic check of the identity of a molecule.

All of these formats support multimolecule files. The PNG and SVG formats arrange the molecules into rows and columns (you can specify the number of rows or columns if you wish), while the ASCII format just uses a single column. The remainder of this chapter will concentrate on the PNG and SVG formats; for more information on the ASCII format, see the format description [ref].

Chapter 9

3D Structure Generation

Open Babel provides support for generating a reasonable 3D structure just given connectivity information. It also has the ability to generate multiple conformers for each molecule. These topics are discussed below.

9.1 Generate a single conformer

There are several steps involved in generating a low-energy conformer from a 0D or 2D structure.

9.1.1 OBBuilder

The **:obapi:‘OBBuilder‘** class is the part of Open Babel that can take a 2D or 0D structure and generate a 3D structure. The 3D structure is made very quickly using a combination of rules (e.g. sp^3 atoms should have four bonds arranged in a tetrahedron) and common fragments (e.g. cyclohexane is shaped like a chair).

The 3D structures that come straight out of OBBuilder may be useful for some purposes but most people will want to “clean them up”. This is because they may have clashes or have high energy structures due to some strain. The conformer search or geometry optimization methods described below are typically used after calling OBBuilder.

Full discussion of the methods for coordinate generation is available in ‘Fast, efficient fragment-based coordinate generation for Open Babel’ **J. Cheminf.* (2019) **11**, Art. 49.<https://doi.org/10.1186/s13321-019-0372-5>*. Please cite this paper if you use the coordinate generation features in Open Babel.

The functionality of OBBuilder is not directly available through **obabel** but it is used as the necessary first step of the Gen3D operation discussed below.

9.1.2 Conformer searching

Given a 3D structure, the goal of conformer searching is to find a low energy conformation. This may be useful as a “clean-up” procedure after an initial 3D structure generation. Note that conformer searching does not alter stereochemistry.

The Open Babel library provides access to several algorithms for conformer searching. All of these algorithms adopt the torsion-driving approach; that is, conformations are generated by setting torsion angles to one of a number of allowed values. The allowed values are listed in the data file `torlib.txt`; for example, C-C bonds in alkanes have three allowed values: -60, 60 and 180.

1. **:obapi:‘Systematic Rotor Search <SystematicRotorSearch>‘**: Systematically iterate through all possible conformers according to Open Babel’s torsion library. This approach is thorough and will find the global minimum. However as the number of conformations increases by multiples for each additional rotational bond, this can take quite a while for molecules with even just 7 rotatable bonds. This approach scales to the power of N, where N is the number of rotatable bonds.
2. **:obapi:‘Fast Rotor Search <FastRotorSearch>‘**: This iterates through the same conformer space as the SystematicRotorSearch but it greedily optimises the torsion angle at each rotatable bond in turn, starting from the most central. Thus it scales linearly with the number of rotatable bonds.
3. **:obapi:‘Random Rotor Search <RandomRotorSearch>‘**: Conformations are generated by randomly choosing from the allowed torsion angles.
4. **:obapi:‘Weighted Rotor Search <WeightedRotorSearch>‘**: This method uses an iterative procedure to find a global minimum. As with the Random Rotor Search, it randomly choses from the allowed torsion angles but the choice is reweighted based on the energy of the generated conformer. Over time, the generated conformer for each step should become increasingly better.

For each of these methods, the lowest energy conformation found is selected. In some cases, the entire set of conformations generated is also available. Many of these methods include an option to optimize the geometry of conformations during the search. This greatly slows down the procedure but may produce more accurate results.

The choice of which algorithm to use depends on the speed/accuracy tradeoff with which you are happy, and also on the number of rotatable bonds in the molecule. Are you looking for a reasonable structure for 3D display? Or are you looking for a structure close to the global minimum?

To use from **obabel**, see the help for the conformer operation (`obabel -L conformer`). This operation is used both for conformer searching and for the genetic algorithm conformer generation described below.

Here is an example of use from Python:

```
>>> ff = ob.OBForceField.FindForceField("mmff94")
>>> ff.Setup(obmol)
True
>>> print(ff.Energy())
15.179054202
>>> ff.SystematicRotorSearch(100)
>>> print(ff.Energy())
10.8861155747
```

9.1.3 Gen3D

To illustrate how some of the above methods might be used in practice, consider the **gen3d** operation. This operation (invoked using `--gen3d` at the commandline) generates 3D structures for 0D or 2D structures using the following series of steps, all of which have been described above:

1. Use the OBBuilder to create a 3D structure using rules and fragment templates
2. Do 250 steps of a steepest descent geometry optimization with the MMFF94 forcefield
3. Do 200 iterations of a Weighted Rotor conformational search (optimizing each conformer with 25 steps of a steepest descent)
4. Do 250 steps of a conjugate gradient geometry optimization

Taken together, all of these steps ensure that the generated structure is likely to be the global minimum energy conformer. However, for many applications where 100s if not 1000s of molecules need to be processed, gen3d is rather slow:

1. `--fastest` only generate coordinates, no force field or conformer search

2. `--fast` perform quick forcefield optimization
3. `--medium` **(default)** forcefield optimization + fast conformer search
4. `--better` more optimization + fast conformer search
5. `--best` more optimization + significant conformer search

Details on some of the trade-offs involved are outlined in ‘Fast, efficient fragment-based coordinate generation for Open Babel’ *J. Cheminf.* (2019) **11**, Art. 49.<https://doi.org/10.1186/s13321-019-0372-5>. If you use the 3D coordinate generation, please cite this paper.

9.2 Generate multiple conformers

In contrast to conformer searching, the goal of conformer generation is not simply to find a low energy conformation but to generate several different conformations. Such conformations may be required by another piece of software such as some protein-ligand docking and pharmacophore programs. They may also be useful if considering writing some sort of shape comparison method.

Open Babel has two distinct conformer generating codes:

1. Confab: A systematic conformer generator that generates all diverse low-energy conformers.
2. Genetic algorithm: This is a stochastic conformer generator that generates diverse conformers either on an energy or RMSD basis

9.2.1 Genetic algorithm

A genetic algorithm is a general computational method to find a globally optimum solution to some multiparameter problem. It involves a population of conformers which after a series of generations, iteratively arrive at an optimal solution in terms of either RMSD diversity or energy.

Information about using this method is available at the command-line using: `obabel -L conformer`. Although labelled as “Conformer Searching”, if you choose the genetic algorithm method (which is the default) then you can save the conformers in the final generation using `--writeconformers`. For example, the following line creates 30 conformers optimized for RMSD diversity:

```
obabel startingConformer.mol -O ga_conformers.sdf --conformer --nconf 30
       --score rmsd --writeconformers
```

In this case `--score rmsd` was not strictly necessary as RMSD diversity was the default in any case.

9.2.2 Confab

Confab systematically generates all diverse low-energy conformers for molecules. To run Confab use the `--confab` operation, and to assess the results by calculating RMSDs to reference structures, use the **confabreport** output format.

confab operator

- `obabel <inputfile> -O <outputfile> --confab [confab options]` for typical usage
- `obabel -L confab` for help text

The *inputfile* should contain one or more 3D structures (note that 2D structures will generate erroneous results). Generated conformers are written to the *outputfile*. All of the conformers for a particular molecule will have the same title as the original molecule.

--rcutoff <rmsd>	RMSD cutoff (default 0.5 Angstrom)
--ecutoff <energy>	Energy cutoff (default 50.0 kcal/mol)
--conf <#confs>	Max number of conformers to test (default is 1 million)
--original	Include the input conformation as the first conformer
--verbose	Verbose - display information on torsions found

confabreport format

- `obabel <inputfile> [-O <outputfile>] -o confabreport -xf <reference_file> [-xr <rmsd>]` for typical usage
- `obabel -L confabreport` for help text

Once a file containing conformers has been generated by Confab, the result can be compared to the original input structures or a set of reference structures using this output format. Conformers are matched with reference structures using the molecule title. For every conformer, there should be a reference structure (but not necessarily *vice versa*).

-f <filename>	File containing reference structures
-r <rmsd>	RMSD cutoff (default 0.5 Angstrom) The number of structures with conformers within this RMSD cutoff of the reference will be reported.

Example

The example file, bostrom.sdf, contains 36 molecules which have from 1 to 11 rotatable bonds (see *Bostrom, Greenwood, Gottfries, J Mol Graph Model, 2003, 21, 449*).

We can generate and test up to 100K conformers using Confab as follows:

```
> obabel bostrom.sdf -O confs.sdf --confab --conf 100000

**Starting Confab 1.1.0
**To support, cite Journal of Cheminformatics, 2011, 3, 8.
..Input format = sdf
..Output format = sdf
..RMSD cutoff = 0.5
..Energy cutoff = 50
..Conformer cutoff = 1000000
..Write input conformation? False
..Verbose? False

**Molecule 1
..title = 1a28_STR_1_A_1__C__
..number of rotatable bonds = 1
..tot conformations = 12
..tot confs tested = 12
..below energy threshold = 10
..generated 3 conformers
```

(continues on next page)

(continued from previous page)

```
... etc, etc

0 molecules converted
```

To check how many of the generated conformers are within 1.0 A RMSD of the original structures, we can use the confabreport format as follows:

```
> obabel confs.sdf -oconfabreport -xf bostrom.sdf -xr 1.0

**Generating Confab Report
..Reference file = bostrom.sdf
..Conformer file = confs.sdf

..Molecule 1
..title = 1a28_STR_1_A_1__C__
..number of confs = 3
..minimum rmsd = 0.0644801
..confs less than cutoffs: 0.2 0.5 1 1.5 2 3 4 100
..1 1 3 3 3 3 3 3
..cutoff (1) passed =  Yes

... etc, etc

**Summary
..number of molecules = 36
..less than cutoff(1) = 35
52271 molecules converted
```

Chapter 10

Molecular Mechanics and Force Fields

Used by a number of features, such as 3D coordinate generation, conformer searching, etc., Open Babel provides support for a variety of all-atom molecular mechanics force fields. The key idea is to use classical mechanics to rapidly simulate molecular systems.

Each force field method is parameterized for a set of possible molecules (e.g., proteins, organic molecules, etc.), building in assumptions about how various aspects of the molecules contribute to the overall potential energy.

The total potential energy of the system is usually given as a sum of multiple components, including some or all of (but not limited to):

- Bond stretching
- Angle bending
- Dihedral torsions
- Out-of-plane bending
- Van der Waals repulsion
- Atomic partial charges (electrostatic)

Open Babel supports several force field methods. In general, we recommend use of either the *Generalized Amber Force Field (gaff)* or *MMFF94 Force Field (mmff94)* for organic molecules, and the *Universal Force Field (uff)* for other types of molecules.

10.1 Generalized Amber Force Field (gaff)

The AMBER force field (or more accurately, family of force fields used with the AMBER software are designed mainly for biomolecules (i.e., proteins, DNA, RNA, carbohydrates, etc.).

A general set of parameters for small organic molecules to allow simulations of drugs and small molecule ligands in conjugtion with biomolecules is provided by GAFF. Parameters exist for almost all molecules made of C, N, O, H, S, P, F, Cl, Br, and I, and are compatible with the AMBER functional forms.

Typically, GAFF expects partial charges assigned using quantum chemistry (i.e., HF/6-31G* RESP charges or AM1-BCC). The Open Babel implementation can use other partial charges as available, although with lower resulting accuracy.

In general, GAFF is expected to provide accuracy (in terms of geometry and energies) on par or better than the *MMFF94 Force Field (mmff94)*.

Note: If you use GAFF, you should cite the appropriate paper: Wang, J., Wolf, R. M.; Caldwell, J. W.;Kollman, P. A.; Case, D. A. "Development and testing of a general AMBER force field". *Journal of Computational Chemistry,* **2004** v. 25, 1157-1174.

10.2 Ghemical Force Field (ghemical)

The Ghemical force field matches an existing open source package, which provided a force field for geometry optimization and molecular dynamics similar to the Tripos-5.2 force field method (which is proprietary). It performs acceptably on providing geometries of organic-like molecules.

We recommend use of either the *Generalized Amber Force Field (gaff)* or *MMFF94 Force Field (mmff94)* for organic molecules, and the *Universal Force Field (uff)* for other types of molecules.

10.3 MMFF94 Force Field (mmff94)

The MMFF94 force field (and the related MMFF94s) were developed by Merck and are sometimes called the Merck Molecular Force Field, although MMFF94 is no longer considered an acronym.

The method provides good accuracy across a range of organic and *drug-like* molecules. The core parameterization was provided by high-quality quantum calculations, rather than experimental data, across ~500 test molecular systems.

The method includes parameters for a wide range of atom types including the following common organic elements: C, H, N, O, F, Si, P, S, Cl, Br, and I. It also supports the following common ions: Fe^{+2}, Fe^{+3}, F^{-}, Cl^{-}, Br^{-}, Li^{+}, Na^{+}, K^{+}, Zn^{+2}, Ca^{+2}, Cu^{+1}, Cu^{+2}, and Mg^{+2}. The Open Babel implementation should automatically perform atom typing and recognize these elements.

MMFF94 performs well at optimizing geometries, bond lengths, angles, etc. and includes electrostatic and hydrogen-bonding effects.

Note: If you use MMFF94 you should cite the appropriate papers:

1. Thomas A. Halgren, *J. Comput. Chem.,* 17, 490-519 **(1996).**
2. Thomas A. Halgren, *J. Comput. Chem.,* 17, 520-552 **(1996).**
3. Thomas A. Halgren, *J. Comput. Chem.,* 17, 553-586 **(1996).**
4. Thomas A. Halgren and Robert B. Nachbar, *J. Comput. Chem.,* 17, 587-615 **(1996).**
5. Thomas A. Halgren, *J. Comput. Chem.,* 17, 616-641 **(1996).**

Some experiments and most theoretical calculations show significant pyramidal "puckering" at nitrogens in isolated structures. The MMFF94s (static) variant has slightly different out-of-plane bending and dihedral torsion parameters to planarize certain types of delocalized trigonal N atoms, such as aromatic aniline. This provides a better match to the time-average molecular geometry in solution or crystal structures.

If you are comparing force-field optimized molecules to crystal structure geometries, we recommend using the MMFF94s variant for this reason. All other parameters are identical.

However, if you are perfoming "docking" simulations, consideration of active solution conformations, or other types of computational studies, we recommend using the MMFF94 variant, since one form or another of the N geometry will predominate.

Note: If you use MMFF94s, you should also cite the following paper that details that method:

6. Thomas A. Halgren, *J. Comput. Chem.*, 20, 720-729 **(1999).**

10.4 Universal Force Field (uff)

One problem with traditional force fields is a limited set of elements and atom types. The Universal Force Field (UFF) was developed to provide a set of rules and procedures for producing appropriate parameters across the entire periodic table.

While some implementations of UFF use the QEq partial charge model, the original manuscript and authors of UFF determined the parameterization without an electrostatic model. Consequently, by default the Open Babel implementation does not use electrostatic interactions.

Note: If you use UFF, you should cite the appropriate paper:

Rappe, A. K.; Casewit, C. J.; Colwell, K. S.; Goddard, W. A. III; Skiff, W. M.; "UFF, a full periodic table force field for molecular mechanics and molecular dynamics simulations." *J Am Chem Soc*, **1992** v. 114, 10024-10039.

Chapter 11

Write software using the Open Babel library

Behind the **obabel** command line program lies a complete cheminformatics toolkit, the Open Babel library. Using this library, you can write your own custom scripts and software for yourself or others.

Note: Any software that uses the Open Babel library must abide by terms of the GNU Public License version 2. This includes all of the supporting language bindings (for example, Python scripts) as well as C++ programs. To summarise, if you are considering distributing your software to other people, you must make your source code available to them on request.

Open Babel is a C++ library and can easily be used from C++. In addition it can be accessed from Python, Perl, Ruby, CSharp and Java. These are referred to as language bindings (the Python bindings, etc.) and they were automatically generated from the C++ library using SWIG. For Python we also provide a module (Pybel) that makes it easier to access features of the bindings.

11.1 The Open Babel API

The API (Application Programming Interface) is the set of classes, methods and variables that a programming library provides to the user. The Open Babel API is implemented in C++, but the same set of classes, methods and variables are accessed through the various language bindings.

The API documentation is automatically generated from the source code using the Doxygen tool. The following links point to the various versions of the documentation:

- API for the current release
- API for the development version (updated nightly, with error report showing errors in documentation)
- API for specific versions: 2.0, 2.1, 2.2, 2.3

The Open Babel toolkit uses a version numbering that indicates how the API has changed over time:

- Bug fix releases (e.g., 2.0.**0**, vs. 2.0.**1**) do not change API at all.
- Minor versions (e.g., 2.**0** vs. 2.**1**) will add function calls, but will be otherwise backwards-compatible.
- Major versions (e.g. **2** vs **3**) are not backwards-compatible, and have changes in the API.

Overall, our goal is for the Open Babel API to remain stable over as long a period as possible. This means that users can be confident that their code will continue to work despite the release of new versions with additional features, file formats and bug fixes. For example, at the time of writing we have been on the version 2 series for almost five years (since November 2005). In other words, a program written using Open Babel almost five years ago still works with the latest release.

11.2 C++

11.2.1 Quickstart example

Here's an example C++ program that uses the Open Babel toolkit to convert between two chemical file formats:

```
#include <iostream>
#include <openbabel/obconversion.h>
using namespace std;

int main(int argc,char **argv)
{
  if(argc<3)
  {
    cout << "Usage: ProgrameName InputFileName OutputFileName\n";
    return 1;
  }

  ifstream ifs(argv[1]);
  if(!ifs)
  {
    cout << "Cannot open input file\n";
    return 1;
  }
  ofstream ofs(argv[2]);
  if(!ofs)
  {
    cout << "Cannot open output file\n";
    return 1;
  }
  OpenBabel::OBConversion conv(&ifs, &ofs);
  if(!conv.SetInAndOutFormats("CML","MOL"))
  {
    cout << "Formats not available\n";
    return 1;
  }
  int n = conv.Convert();
  cout << n << " molecules converted\n";

  return 0;
}
```

Next, we'll look at how to compile this.

11.2.2 How to compile against the Open Babel library

Using Makefiles

The following Makefile can be used to compile the above example, assuming that it's saved as `example.cpp`. You need to have already installed Open Babel somewhere. If the include files or the library are not automatically found when running **make**, you can specify the location as shown by the commented out statements in CFLAGS and LDFLAGS below.

```
CC = g++
CFLAGS = -c # -I /home/user/Tools/openbabel/install/include/openbabel-2.0
LDFLAGS = -lopenbabel # -L /home/user/Tools/openbabel/install/lib

all: example

example: example.o
	$(CC) $(LDFLAGS) example.o -o example

example.o: example.cpp
	$(CC) $(CFLAGS) $(LDFLAGS) example.cpp

clean:
	rm -rf example.o example
```

Using CMake

Rather than create a Makefile yourself, you can get CMake to do it for you. The nice thing about using CMake is that it can generate not only Makefiles, but also project files for MSVC++, KDevelop and Eclipse (among others). The following `CMakeLists.txt` can be used to generate any of these. The commented out lines can be used to specify the location of the Open Babel library and include files if necessary.

```
cmake_minimum_required(VERSION 2.6)
add_executable(example example.cpp)
target_link_libraries(example openbabel)
# target_link_libraries(example /home/user/Tools/openbabel/install/lib/libopenbabel.
↪so)
# include_directories(/home/user/Tools/openbabel/install/include/openbabel-2.0)
```

11.2.3 Further examples

Output Molecular Weight for a Multi-Molecule SDF File

Let’s say we want to print out the molecular weights of every molecule in an SD file. Why? Well, we might want to plot a histogram of the distribution, or see whether the average of the distribution is significantly different (in the statistical sense) compared to another SD file.

```
#include <iostream>

#include <openbabel/obconversion.h>
#include <openbabel/mol.h>

int main(int argc,char **argv)
{
  OBConversion obconversion;
  obconversion.SetInFormat("sdf");
  OBMol mol;

  bool notatend = obconversion.ReadFile(&mol,"../xsaa.sdf");
  while (notatend)
  {
```

(continues on next page)

(continued from previous page)

```
    std::cout << "Molecular Weight: " << mol.GetMolWt() << std::endl;

    mol.Clear();
    notatend = obconversion.Read(&mol);
  }

  return(0);
}
```

Properties from SMARTS Matches

Let's say that we want to get the average bond length or dihedral angle over particular types of atoms in a large molecule. So we'll use SMARTS to match a set of atoms and loop through the matches. The following example does this for sulfur-carbon-carbon-sulfur dihedral angles in a polymer and the carbon-carbon bond lengths between the monomer units:

```
OBMol obMol;
OBBond *b1;
OBConversion obConversion;
OBFormat *inFormat;
OBSmartsPattern smarts;
smarts.Init("[#16D2r5][#6D3r5][#6D3r5][#16D2r5]");

string filename;
vector< vector <int> > maplist;
vector< vector <int> >::iterator matches;
double dihedral, bondLength;

for (int i = 1; i < argc; i++)
  {
    obMol.Clear();
    filename = argv[i];
    inFormat = obConversion.FormatFromExt(filename.c_str());
    obConversion.SetInFormat(inFormat);
    obConversion.ReadFile(&obMol, filename);

    if (smarts.Match(obMol))
      {
        dihedral = 0.0;
        bondLength = 0.0;
        maplist = smarts.GetUMapList();
        for (matches = maplist.begin(); matches != maplist.end(); matches++)
          {
            dihedral += fabs(obMol.GetTorsion((*matches)[0],
                                              (*matches)[1],
                                              (*matches)[2],
                                              (*matches)[3]));
            b1 = obMol.GetBond((*matches)[1], (*matches)[2]);
            bondLength += b1->GetLength();
          }
        cout << filename << ": Average Dihedral " << dihedral / maplist.size()
             << " Average Bond Length " << bondLength / maplist.size()
             << " over " << maplist.size() <<  " matches\n";
      }
  }
```

11.3 Python

11.3.1 Introduction

The Python interface to Open Babel is perhaps the most popular of the several languages that Open Babel supports. We provide two Python modules that can be used to access the functionality of Open Babel toolkit:

1. The *openbabel* module:

   This contains the standard Python bindings automatically generated using SWIG from the C++ API. See *The openbabel module*.

2. The *Pybel* module:

   This is a light-weight wrapper around the classes and methods in the *openbabel* module. Pybel provides more convenient and Pythonic ways to access the Open Babel toolkit. See *Pybel*.

You don't have to choose between them though - they can be used together.

11.3.2 Install Python bindings

Windows

Install the bindings

1. First you need to download and install the main Open Babel executable and library as described in *Install a binary package*.

2. Next, use `pip` to install the Python bindings:

   ```
   pip install -U openbabel
   ```

Note: Python is available as either a 32-bit or 64-bit version. You need to install the corresponding version of Open Babel in step 1.

Install Pillow (optional)

If you want to display 2D depictions using Pybel (rather than just write to a file), you need to install the Pillow library:

```
pip install -U pillow
```

Test the installation

Open a Windows command prompt, and type the following commands to make sure that everything is installed okay. If you get an error message, there's something wrong and you should email the mailing list with the output from these commands.

```
C:\Documents and Settings\Noel> obabel -V
Open Babel 3.0.0 -- Oct  7 2019 -- 20:18:16

C:\Documents and Settings\Noel> obabel -Hsdf
sdf  MDL MOL format
Reads and writes V2000 and V3000 versions
```

(continues on next page)

(continued from previous page)

```
Read Options, e.g. -as
 s  determine chirality from atom parity flags
...
...

C:\Documents and Settings\Noel> dir "%BABEL_DATADIR%"\mr.txt
 Volume in drive C has no label.
 Volume Serial Number is 68A3-3CC9

 Directory of C:\Users\Noel\AppData\Roaming\OpenBabel-3.0.0\data

06/10/2019  16:37             4,295 mr.txt
               1 File(s)          4,295 bytes
               0 Dir(s)  58,607,575,040 bytes free

C:\Documents and Settings\Noel> py
Python 2.7.16 (v2.7.16:413a49145e, Mar  4 2019, 01:37:19) [MSC v.1500 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from openbabel import pybel
>>> mol = pybel.readstring("smi", "CC(=O)Br")
>>> mol.make3D()
>>> print(mol.write("sdf"))

 OpenBabel01010918183D

  7  6  0  0  0  0  0  0  0  0999 V2000
    1.0166   -0.0354   -0.0062 C   0  0  0  0  0
    2.5200   -0.1269    0.0003 C   0  0  0  0  0
    3.0871   -1.2168    0.0026 O   0  0  0  0  0
    3.2979    1.4258    0.0015 Br  0  0  0  0  0
    0.6684    1.0007    0.0052 H   0  0  0  0  0
    0.6255   -0.5416    0.8803 H   0  0  0  0  0
    0.6345   -0.5199   -0.9086 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  4  1  0  0  0
  2  3  2  0  0  0
M  END
$$$$
>>> mol.draw() # If you installed PIL, this will display its structure
>>> (Hit CTRL+Z followed by Enter to exit)
```

Linux and MacOSX

See *Compile language bindings* for information on how to configure CMake to compile the Python bindings. This can be done either globally or locally.

You may need to add the location of `libopenbabel.so` (on my system, the location is `/usr/local/lib`) to the environment variable LD_LIBRARY_PATH if you get the following error when you try to import the OpenBabel library at the Python prompt:

```
$ python
>>> from openbabel import openbabel
Traceback (most recent call last):
  File "<stdin>", line 1, in
  File "/usr/lib/python2.4/site-packages/openbabel.py", line 9, in
   import _openbabel
ImportError: libopenbabel.so.3: cannot open shared object file: No such file or directory
```

Install Pillow (optional)

If you want to display 2D depictions using Pybel (rather than just write to a file), you need the Pillow library, and the Python Tkinter library (part of the standard library). These should be available through your package manager, e.g. on Ubuntu, Pillow is provided by ‘python-pil’ and ‘python-pil.imagetk’, while Tkinter is provided by ‘python-tk’.

11.3.3 The openbabel module

The **openbabel** module provides direct access to the C++ Open Babel library from Python. This binding is generated using the SWIG package and provides access to almost all of the Open Babel interfaces via Python, including the base classes OBMol, OBAtom, OBBond, and OBResidue, as well as the conversion framework OBConversion. As such, essentially any call in the C++ API is available to Python scripts with very little difference in syntax. As a result, the principal documentation is the *C++ API documentation*.

Examples

Here we give some examples of common Python syntax for the `openbabel` module and pointers to the appropriate sections of the API documentation.

The example script below creates atoms and bonds one-by-one using the **:obapi:‘OBMol‘**, **:obapi:‘OBAtom‘**, and **:obapi:‘OBBond‘** classes.

```
from openbabel import openbabel

mol = openbabel.OBMol()
print(mol.NumAtoms()) #Should print 0 (atoms)

a = mol.NewAtom()
a.SetAtomicNum(6)   # carbon atom
a.SetVector(0.0, 1.0, 2.0) # coordinates

b = mol.NewAtom()
mol.AddBond(1, 2, 1)   # atoms indexed from 1
print(mol.NumAtoms()) #Should print 2 (atoms)
print(mol.NumBonds()) Should print 1 (bond)

mol.Clear();
```

More commonly, Open Babel can be used to read in molecules using the **:obapi:‘OBConversion‘** framework. The following script reads in molecular information (a SMI file) from a string, adds hydrogens, and writes out an MDL file as a string.

```
from openbabel import openbabel

obConversion = openbabel.OBConversion()
obConversion.SetInAndOutFormats("smi", "mdl")

mol = openbabel.OBMol()
obConversion.ReadString(mol, "C1=CC=CS1")

print(mol.NumAtoms()) #Should print 5 (atoms)

mol.AddHydrogens()
print(mol.NumAtoms()) Should print 9 (atoms) after adding hydrogens

outMDL = obConversion.WriteString(mol)
```

The following script writes out a file using a filename, rather than reading and writing to a Python string.

```
from openbabel import openbabel

obConversion = openbabel.OBConversion()
obConversion.SetInAndOutFormats("pdb", "mol2")

mol = openbabel.OBMol()
obConversion.ReadFile(mol, "1ABC.pdb.gz")   # Open Babel will uncompress automatically

mol.AddHydrogens()

print(mol.NumAtoms())
print(mol.NumBonds())
print(mol.NumResidues())

obConversion.WriteFile(mol, '1abc.mol2')
```

Using iterators

A number of Open Babel toolkit classes provide iterators over various objects; these classes are identifiable by the suffix “Iter” in the list of toolkit classes in the API:

- OBAtomAtomIter and OBAtomBondIter - given an OBAtom, iterate over all neighboring OBAtoms or OBBonds
- OBMolAtomIter, OBMolBondIter, OBMolAngleIter, OBMolTorsionIter, OBMolRingIter - given an OBMol, iterate over all OBAtoms, OBBonds, OBAngles, OBTorsions or OBRings.
- OBMolAtomBFSIter - given an OBMol and the index of an atom, OBMolAtomBFSIter iterates over all the neighbouring atoms in a breadth-first manner. It differs from the other iterators in that it returns two values - an OBAtom, and the ‘depth’ of the OBAtom in the breadth-first search (this is useful, for example, when creating circular fingerprints)
- OBMolPairIter - given an OBMol, iterate over all pairs of OBAtoms separated by more than three bonds
- OBResidueIter - given an OBMol representing a protein, iterate over all OBResidues
- OBResidueAtomIter - given an OBResidue, iterate over all OBAtoms

These iterator classes can be used using the typical Python syntax for iterators:

```
for obatom in openbabel.OBMolAtomIter(obmol):
    print(obatom.GetAtomicMass())
```

Note that OBMolTorsionIter returns atom IDs which are off by one. That is, you need to add one to each ID to get the correct ID. Also, if you add or remove atoms, you will need to delete the existing TorsionData before using OBMolTorsionIter. This is done as follows:

```
mol.DeleteData(openbabel.TorsionData)
```

Calling a method requiring an array of C doubles

Some Open Babel toolkit methods, for example **:obapi:‘OBMol::Rotate() <OpenBabel::OBMol::Rotate>‘**, require an array of doubles. It’s not possible to directly use a list of floats when calling such a function from Python. Instead, you need to first explicitly create a C array using the *double_array()* function:

```
obMol.Rotate([1.0, -54.7, 3])
# Error!
myarray = openbabel.double_array([1.0, -54.7, 3])
obMol.Rotate(myarray)
# Works!
```

Accessing OBPairData, OBUnitCell and other OBGenericData

If you want to access any subclass of OBGenericData (such as **:obapi:‘OBPairData‘** or **:obapi:‘OBUnitCell‘**) associated with a molecule, you need to ‘cast’ the **:obapi:‘OBGenericData‘** returned by **:obapi:‘OBMol.GetData() <OpenBabel::OBMol::GetData>‘** using the *toPairData()*, *toUnitCell()* (etc.) functions:

```
pairdata = [openbabel.toPairData(x) for x in obMol.GetData()
            if x.GetDataType()==openbabel.PairData]
print(pairdata[0].GetAttribute(), pairdata[0].GetValue())

unitcell = openbabel.toUnitCell(obMol.GetData(openbabel.UnitCell))
print(unitcell.GetAlpha(), unitcell.GetSpaceGroup())
```

Using FastSearch from Python

Rather than use the **:obapi:‘FastSearch‘** class directly, it’s easiest to use the **:obapi:‘OpenInAndOutFiles() <OpenBabel::OBConversion::OpenInAndOutFiles>‘** method as follows:

```
>>> from openbabel import openbabel
>>> conv=openbabel.OBConversion()
>>> conv.OpenInAndOutFiles("1200mols.smi","index.fs")
True
>>> conv.SetInAndOutFormats("smi","fs")
True
>>> conv.Convert()
This will prepare an index of 1200mols.smi and may take some time...
It took 6 seconds
1192
>>> conv.CloseOutFile()
>>> conv.OpenInAndOutFiles("index.fs","results.smi")
True
```

(continues on next page)

(continued from previous page)

```
>>> conv.SetInAndOutFormats("fs","smi")
True
>>> conv.AddOption("s",conv.GENOPTIONS,"C=CC#N")
>>> conv.Convert()
10 candidates from fingerprint search phase
1202
>>> f=open("results.smi")
>>> f.read()
'OC(=O)C(=Cc1ccccc1)C#N\t298\nN#CC(=Cc1ccccc1)C#N\t490\nO=N(=O)c1cc(ccc1)C=C(C#N
)C#N\t491\nClc1ccc(cc1)C=C(C#N)C#N\t492\nClc1ccc(c(c1)Cl)C=C(C#N)C#N\t493\nClc1c
cc(cc1Cl)C=C(C#N)C#N\t494\nBrc1ccc(cc1)C=C(C#N)C#N\t532\nClc1ccccc1C=C(C#N)C#N\t
542\nN#CC(=CC=Cc1occc1)C#N\t548\nCCOC(=O)C(C#N)=C(C)C\t1074\n'
```

Combining numpy with Open Babel

If you are using the Python numerical extension, numpy, and you try to pass values from a numpy array to Open Babel, it may not work unless you convert the values to Python built-in types first:

```
import numpy
from openbabel import openbabel

mol = openbabel.OBMol()
atom = mol.NewAtom()

coord = numpy.array([1.2, 2.3, 4.6], "float32")
atom.SetVector(coord[0], coord[1], coord[2])
# Error

atom.SetVector(float(coord[0]), float(coord[1]), float(coord[2]))
# No error

coord = numpy.array([1.2, 2.3, 4.6], "float64")
atom.SetVector(coord[0], coord[1], coord[2])
# No error either - not all numpy arrays will cause an error
```

11.3.4 Pybel

Pybel provides convenience functions and classes that make it simpler to use the Open Babel libraries from Python, especially for file input/output and for accessing the attributes of atoms and molecules. The Atom and Molecule classes used by Pybel can be converted to and from the OBAtom and OBMol used by the `openbabel` module. These features are discussed in more detail below.

The rationale and technical details behind Pybel are described in O'Boyle et al [omh2008]. To support further development of Pybel, please cite this paper if you use Pybel to obtain results for publication.

Information on the Pybel API can be found at the interactive Python prompt using the `help()` function. The full API is also listed in the next section (see *Pybel API*).

To use Pybel, use `from openbabel import pybel`.

Atoms and Molecules

A *`Molecule`* can be created in any of three ways:

1. From an **:obapi:'OBMol'**, using `Molecule(myOBMol)`
2. By reading from a file (see *Input/Output* below)
3. By reading from a string (see *Input/Output* below)

An `Atom` be created in two different ways:

1. From an **:obapi:'OBAtom'**, using `Atom(myOBAtom)`
2. By accessing the `atoms` attribute of a `Molecule`

Using Pybel with `openbabel.py`

It is always possible to access the OBMol or OBAtom on which a Molecule or Atom is based, by accessing the appropriate attribute, either `.OBMol` or `.OBAtom`. In this way, it is easy to combine the convenience of `pybel` with the many additional capabilities present in `openbabel`. See *Combining Pybel with openbabel.py* below.

Molecules have the following attributes: `atoms`, `charge`, `data`, `dim`, `energy`, `exactmass`, `formula`, `molwt`, `spin`, `sssr`, `title` and `unitcell` (if crystal data). The `atoms` attribute provides a list of the Atoms in a Molecule. The `data` attribute returns a dictionary-like object for accessing and editing the data fields associated with the molecule (technically, it's a `MoleculeData` object, but you can use it like it's a regular dictionary). The `unitcell` attribute gives access to any unit cell data associated with the molecule (see **:obapi:'OBUnitCell'**). The remaining attributes correspond directly to attributes of OBMols: e.g. `formula` is equivalent to **:obapi:'OBMol::GetFormula() <OpenBabel::OBMol::GetFormula>'**. For more information on what these attributes are, please see the Open Babel C++ documentation for **:obapi:'OBMol'**.

For example, let's suppose we have an SD file containing descriptor values in the data fields:

```
>>> mol = next(readfile("sdf", "calculatedprops.sdf")) # (readfile is described below)
>>> print(mol.molwt)
100.1
>>> print(len(mol.atoms))
16
>>> print(mol.data.keys())
{'Comment': 'Created by CDK', 'NSC': 1, 'Hydrogen Bond Donors': 3,
 'Surface Area': 342.43, .... }
>>> print(mol.data['Hydrogen Bond Donors'])
3
>>> mol.data['Random Value'] = random.randint(0,1000) # Add a descriptor containing noise
```

Molecules have a `write()` method that writes a representation of a Molecule to a file or to a string. See *Input/Output* below. They also have a `calcfp()` method that calculates a molecular fingerprint. See *Fingerprints* below.

The `draw()` method of a Molecule generates 2D coordinates and a 2D depiction of a molecule. It uses the OASA library by Beda Kosata to do this. The default options are to show the image on the screen (`show=True`), not to write to a file (`filename=None`), to calculate 2D coordinates (`usecoords=False`) but not to store them (`update=False`).

The `addh()` and `removeh()` methods allow hydrogens to be added and removed.

If a molecule does not have 3D coordinates, they can be generated using the `make3D()` method. By default, this includes 50 steps of a geometry optimisation using the MMFF94 forcefield. The list of available forcefields is stored in the `forcefields` variable. To further optimise the structure, you can use the `localopt()` method, which by default carries out 500 steps of an optimisation using MMFF94. Note that hydrogens need to be added before calling `localopt()`.

The `calcdesc()` method of a Molecule returns a dictionary containing descriptor values for LogP, Polar Surface Area ("TPSA") and Molar Refractivity ("MR"). A list of the available descriptors is contained in the variable `descs`.

If only one or two descriptor values are required, you can specify the names as follows: `calcdesc(["LogP", "TPSA"])`. Since the `data` attribute of a Molecule is also a dictionary, you can easily add the result of `calcdesc()` to an SD file (for example) as follows:

```
mol = next(readfile("sdf", "without_desc.sdf"))
descvalues = mol.calcdesc()
# In Python, the update method of a dictionary allows you
# to add the contents of one dictionary to another
mol.data.update(descvalues)
output = Outputfile("sdf", "with_desc.sdf")
output.write(mol)
output.close()
```

For convenience, a Molecule provides an iterator over its Atoms. This is used as follows:

```
for atom in myMolecule:
   # do something with atom
```

Atoms have the following attributes: `atomicmass`, `atomicnum`, `coords`, `exactmass`, `formalcharge`, `heavyvalence`, `heterovalence`, `hyb`, `idx`, `implicitvalence`, `isotope`, `partialcharge`, `spin`, `type`, `valence`, `vector`. The `.coords` attribute provides a tuple (x, y, z) of the atom's coordinates. The remaining attributes are as for the *Get* methods of **:obapi:'OBAtom'**.

Input/Output

One of the strengths of Open Babel is the number of chemical file formats that it can handle (see *Supported File Formats and Options*). Pybel provides a dictionary of the input and output formats in the variables `informats` and `outformats` where the keys are the three-letter codes for each format (e.g. `pdb`) and the values are the descriptions (e.g. `Protein Data Bank format`).

Pybel greatly simplifies the process of reading and writing molecules to and from strings or files. There are two functions for reading Molecules:

1. `readstring()` reads a Molecule from a string
2. `readfile()` provides an iterator over the Molecules in a file

Here are some examples of their use. Note in particular the use of `next()` to access the first (and possibly only) molecule in a file:

```
>>> mymol = readstring("smi", "CCCC")
>>> print(mymol.molwt)
58
>>> for mymol in readfile("sdf", "largeSDfile.sdf")
... print(mymol.molwt)
>>> singlemol = next(readfile("pdb", "1CRN.pdb"))
```

If a single molecule is to be written to a molecule or string, the `write()` method of the Molecule should be used:

1. `mymol.write(format)` returns a string
2. `mymol.write(format, filename)` writes the Molecule to a file. An optional additional parameter, `overwrite`, should be set to `True` if you wish to overwrite an existing file.

For files containing multiple molecules, the `Outputfile` class should be used instead. This is initialised with a format and filename (and optional `overwrite` parameter). To write a Molecule to the file, the `write()` method of the Outputfile is called with the Molecule as a parameter. When all molecules have been written, the *close()* method of the Outputfile should be called.

Here are some examples of output using the Pybel methods and classes:

```
>>> print(mymol.write("smi"))
'CCCC'
>>> mymol.write("smi", "outputfile.txt")
>>> largeSDfile = Outputfile("sdf", "multipleSD.sdf")
>>> largeSDfile.write(mymol)
>>> largeSDfile.write(myothermol)
>>> largeSDfile.close()
```

Fingerprints

A *Fingerprint* can be created in either of two ways:

1. From a vector returned by the OpenBabel GetFingerprint() method, using `Fingerprint(myvector)`
2. By calling the *calcfp()* method of a Molecule

The *calcfp()* method takes an optional argument, `fptype`, which should be one of the fingerprint types supported by OpenBabel (see *Molecular fingerprints and similarity searching*). The list of supported fingerprints is stored in the variable *fps*. If unspecified, the default fingerprint (`FP2`) is calculated.

Once created, the Fingerprint has two attributes: `fp` gives the original OpenBabel vector corresponding to the fingerprint, and *bits* gives a list of the bits that are set.

The Tanimoto coefficient of two Fingerprints can be calculated using the `|` operator.

Here is an example of its use:

```
>>> from openbabel import pybel
>>> smiles = ['CCCC', 'CCCN']
>>> mols = [pybel.readstring("smi", x) for x in smiles] # Create a list of two␣
↪molecules
>>> fps = [x.calcfp() for x in mols] # Calculate their fingerprints
>>> print(fps[0].bits, fps[1].bits)
[261, 385, 671] [83, 261, 349, 671, 907]
>>> print(fps[0] | fps[1]) # Print the Tanimoto coefficient
0.3333
```

SMARTS matching

Pybel also provides a simplified API to the Open Babel SMARTS pattern matcher. A *Smarts* object is created, and the *findall()* method is then used to return a list of the matches to a given Molecule.

Here is an example of its use:

```
>>> mol = readstring("smi","CCN(CC)CC") # triethylamine
>>> smarts = Smarts("[#6][#6]") # Matches an ethyl group
>>> print(smarts.findall(mol))
[(1, 2), (4, 5), (6, 7)]
```

Combining Pybel with `openbabel.py`

It is easy to combine the ease of use of Pybel with the comprehensive coverage of the Open Babel toolkit that `openbabel.py` provides. Pybel is really a wrapper around `openbabel.py`, with the result that the OBAtom and OBMol used by `openbabel.py` can be interconverted to the Atom and Molecule used by Pybel.

The following example shows how to read a molecule from a PDB file using Pybel, and then how to use `openbabel.py` to add hydrogens. It also illustrates how to find out information on what methods and classes are available, while at the interactive Python prompt.

```
>>> from openbabel import pybel
>>> mol = next(pybel.readfile("pdb", "1PYB"))
>>> help(mol)
Help on Molecule in module pybel object:
...
 |  Attributes:
 |     atoms, charge, dim, energy, exactmass, flags, formula,
 |     mod, molwt, spin, sssr, title.
...
 |  The original Open Babel molecule can be accessed using the attribute:
 |     OBMol
...
>>> print(len(mol.atoms), mol.molwt)
3430 49315.2
>>> dir(mol.OBMol) # Show the list of methods provided by openbabel.py
['AddAtom', 'AddBond', 'AddConformer', 'AddHydrogens', 'AddPolarHydrogens', ... ]
>>> mol.OBMol.AddHydrogens()
>>> print(len(mol.atoms), mol.molwt)
7244 49406.0
```

The next example is an extension of one of the `openbabel.py` examples at the top of this page. It shows how a molecule could be created using `openbabel.py`, and then written to a file using Pybel:

```
from openbabel import openbabel, pybel

mol = openbabel.OBMol()
a = mol.NewAtom()
a.SetAtomicNum(6)    # carbon atom
a.SetVector(0.0, 1.0, 2.0) # coordinates
b = mol.NewAtom()
mol.AddBond(1, 2, 1)    # atoms indexed from 1

pybelmol = pybel.Molecule(mol)
pybelmol.write("sdf", "outputfile.sdf")
```

11.3.5 Pybel API

pybel - A Python module that simplifies access to the Open Babel API

Global variables: *informats*, *outformats*, *descs*, *fps*, *forcefields*, *operations*

Functions: *readfile()*, *readstring()*

Classes: *Atom*, *Molecule*, *Outputfile*, *Fingerprint*, *Smarts*, *MoleculeData*

Note: The `openbabel.py` module can be accessed through the `ob` global variable.

class pybel.**Atom**(*OBAtom*)

Represent an atom.

Required parameter: OBAtom – an Open Babel **:obapi:‘OBAtom‘**

Attributes: *atomicmass*, *atomicnum*, *coords*, *exactmass*, *formalcharge*, *heavyvalence*, *heterovalence*, *hyb*, *idx*, *implicitvalence*, *isotope*, *partialcharge*, *spin*, *type*, *valence*, *vector*.

The underlying Open Babel **:obapi:‘OBAtom‘** can be accessed using the attribute:

```
OBAtom
```

atomicmass

Atomic mass

atomicnum

Atomic number

coords

Coordinates of the atom

exactmass

Exact mass

formalcharge

Formal charge

heavyvalence

Number of non-hydrogen atoms attached

heterovalence

Number of heteroatoms attached

hyb

The hybridization of this atom: 1 for sp, 2 for sp2, 3 for sp3, . . .

For further details see **:obapi:‘OBAtom::GetHyb() <OpenBabel::OBAtom::GetHyb>‘**

idx

The index of the atom in the molecule (starts at 1)

implicitvalence

The maximum number of connections expected for this molecule

isotope

The isotope for this atom if specified; 0 otherwise.

partialcharge

Partial charge

spin

Spin multiplicity

type

Atom type

valence

Number of explicit connections

vector

Coordinates as a **:obapi:‘vector3‘** object.

class pybel.**Fingerprint**(*fingerprint*)

A molecular fingerprint.

Required parameters: **fingerprint** – a vector calculated by **:obapi:'OBFingerprint::FindFingerprint() <OpenBabel::OBFingerprint::FindFingerprint>'**

Attributes: *bits*

Methods: The | operator can be used to calculate the Tanimoto coefficient. For example, given two Fingerprints *a* and *b*, the Tanimoto coefficient is given by:

```
tanimoto = a | b
```

The underlying fingerprint object can be accessed using the attribute `fp`.

bits
A list of bits set in the fingerprint

class pybel.**Molecule**(*OBMol*)
Represent a Pybel Molecule.

Required parameter: **OBMol** – an Open Babel **:obapi:'OBMol'** or any type of Cinfony Molecule

Attributes: *atoms*, *charge*, *conformers*, *data*, *dim*, *energy*, *exactmass*, *formula*, *molwt*, *spin*, *sssr*, *title*, *unitcell*.

Methods: *addh()*, *calcfp()*, *calcdesc()*, *draw()*, *localopt()*, *make3D()*, *removeh()*, *write()*

The underlying Open Babel :obapi:'OBMol' can be accessed using the attribute: `OBMol`

An iterator (`__iter__()`) is provided that iterates over the atoms of the molecule. This allows constructions such as the following:

```
for atom in mymol:
    print(atom)
```

addh()
Add hydrogens.

atoms
A list of atoms of the molecule

calcdesc(*descnames=[]*)
Calculate descriptor values.

Optional parameter:

descnames – a list of names of descriptors See the *descs* variable for a list of available descriptors.

If *descnames* is not specified, all available descriptors are calculated.

calcfp(*fptype='FP2'*)
Calculate a molecular fingerprint.

Optional parameters:

fptype – the fingerprint type (default is FP2). See the *fps* variable for a list of of available fingerprint types.

charge
The charge on the molecule

conformers
Conformers of the molecule

data
Access the molecule's data through a dictionary-like object, `MoleculeData`.

dim
Are the coordinates 2D, 3D or 0D?

draw (*show=True*, *filename=None*, *update=False*, *usecoords=False*)
Create a 2D depiction of the molecule.

Optional parameters:

show – display on screen (default is `True`)

filename – write to file (default is `None`)

update – update the coordinates of the atoms This sets the atom coordinates to those determined by the structure diagram generator (default is `False`)

usecoords – use the current coordinates This causes the current coordinates to be used instead of calculating new 2D coordinates (default is `False`)

OASA is used for 2D coordinate generation and depiction. Tkinter and Python Imaging Library are required for image display.

energy
The molecule's energy

exactmass
The exact mass

formula
The molecular formula

localopt (*forcefield='mmff94'*, *steps=500*)
Locally optimize the coordinates.

Optional parameters:

forcefield – default is `mmff94`. See the `forcefields` variable for a list of available forcefields.

steps – default is 500

If the molecule does not have any coordinates, `make3D()` is called before the optimization. Note that the molecule needs to have explicit hydrogens. If not, call `addh()`.

make3D (*forcefield='mmff94'*, *steps=50*)
Generate 3D coordinates.

Optional parameters:

forcefield – default is `mmff94`. See the `forcefields` variable for a list of available forcefields.

steps – default is `50`

Once coordinates are generated, hydrogens are added and a quick local optimization is carried out with 50 steps and the MMFF94 forcefield. Call `localopt()` if you want to improve the coordinates further.

molwt
The molecular weight

removeh ()
Remove hydrogens.

spin
The spin multiplicity

sssr
The Smallest Set of Smallest Rings (SSSR)

title
The molecule title

unitcell
Access any unit cell data

write (*format='smi'*, *filename=None*, *overwrite=False*)
Write the molecule to a file or return a string.

Optional parameters:

format – chemical file format See the `outformats` variable for a list of available output formats (default is `smi`)

filename – default is `None`

overwrite – overwrite the output file if it already exists? Default is `False`.

If a *filename* is specified, the result is written to a file. Otherwise, a string is returned containing the result.

To write multiple molecules to the same file you should use the `Outputfile` class.

class pybel.**MoleculeData** (*obmol*)
Store molecule data in a dictionary-type object

Required parameters: *obmol* – an Open Babel **:obapi:'OBMol'**

Methods and accessor methods are like those of a dictionary except that the data is retrieved on-the-fly from the underlying **:obapi:'OBMol'**.

Example:

```
>>> mol = readfile("sdf", 'head.sdf').next() # Python 2
>>> # mol = next(readfile("sdf", 'head.sdf')) # Python 3
>>> data = mol.data
>>> print(data)
{'Comment': 'CORINA 2.61 0041  25.10.2001', 'NSC': '1'}
>>> print(len(data), data.keys(), data.has_key("NSC"))
2 ['Comment', 'NSC'] True
>>> print(data['Comment'])
CORINA 2.61 0041  25.10.2001
>>> data['Comment'] = 'This is a new comment'
>>> for k,v in data.items():
...     print(k, "-->", v)
Comment --> This is a new comment
NSC --> 1
>>> del data['NSC']
>>> print(len(data), data.keys(), data.has_key("NSC"))
1 ['Comment'] False
```

clear ()

has_key (*key*)

items ()

iteritems ()

keys ()

update (*dictionary*)

values()

class pybel.**Outputfile**(*format*, *filename*, *overwrite=False*, *opt=None*)

Represent a file to which *output* is to be sent.

Although it's possible to write a single molecule to a file by calling the *write()* method of a *Molecule*, if multiple molecules are to be written to the same file you should use the *Outputfile* class.

Required parameters:

format – chemical file format See the *outformats* variable for a list of available output formats

filename

Optional parameters:

overwrite – overwrite the output file if it already exists? Default is False

opt – a dictionary of format-specific options For format options with no parameters, specify the value as None.

Methods: *write()*, *close()*

close()

Close the output file to further writing.

write(*molecule*)

Write a molecule to the output file.

Required parameters: molecule – A *Molecule*

class pybel.**Smarts**(*smartspattern*)

A Smarts Pattern Matcher

Required parameters: smartspattern - A SMARTS pattern

Methods: *findall()*

Example:

```
>>> mol = readstring("smi","CCN(CC)CC") # triethylamine
>>> smarts = Smarts("[#6][#6]") # Matches an ethyl group
>>> print(smarts.findall(mol))
[(1, 2), (4, 5), (6, 7)]
```

The numbers returned are the indices (starting from 1) of the atoms that match the SMARTS pattern. In this case, there are three matches for each of the three ethyl groups in the molecule.

findall(*molecule*)

Find all matches of the SMARTS pattern to a particular molecule.

Required parameters: molecule - A *Molecule*

pybel.**descs = ['LogP', 'MR', 'TPSA']**

A list of supported descriptors

pybel.**forcefields = ['uff', 'mmff94', 'ghemical']**

A list of supported forcefields

pybel.**fps = ['FP2', 'FP3', 'FP4']**

A list of supported fingerprint types

pybel.**informats = {}**

A dictionary of supported input formats

`pybel.`**`operations = ['Gen3D']`**

A list of supported operations

`pybel.`**`outformats = {}`**

A dictionary of supported output formats

`pybel.`**`readfile`**(*format*, *filename*, *opt=None*)

Iterate over the molecules in a file.

Required parameters:

format – chemical file format See the *`informats`* variable for a list of available input formats

filename

Optional parameters:

opt – a dictionary of format-specific options For format options with no parameters, specify the value as None.

You can access the first molecule in a file using the next() method of the iterator (or the next() keyword in Python 3):

```
mol = readfile("smi", "myfile.smi").next() # Python 2
mol = next(readfile("smi", "myfile.smi"))  # Python 3
```

You can make a list of the molecules in a file using:

```
mols = list(readfile("smi", "myfile.smi"))
```

You can iterate over the molecules in a file as shown in the following code snippet:

```
>>> atomtotal = 0
>>> for mol in readfile("sdf", "head.sdf"):
...     atomtotal += len(mol.atoms)
...
>>> print(atomtotal)
43
```

`pybel.`**`readstring`**(*format*, *string*, *opt=None*)

Read in a molecule from a string.

Required parameters:

format – chemical file format See the *`informats`* variable for a list of available input formats

string

Optional parameters:

opt – a dictionary of format-specific options For format options with no parameters, specify the value as None.

Example:

```
>>> input = "C1=CC=CS1"
>>> mymol = readstring("smi", input)
>>> len(mymol.atoms)
5
```

11.3.6 Examples

Output Molecular Weight for a Multi-Molecule SDF File

Let's say we want to print out the molecular weights of every molecule in an SD file. Why? Well, we might want to plot a histogram of the distribution, or see whether the average of the distribution is significantly different (in the statistical sense) compared to another SD file.

openbabel.py

```python
from openbabel import openbabel as ob

obconversion = ob.OBConversion()
obconversion.SetInFormat("sdf")
obmol = ob.OBMol()

notatend = obconversion.ReadFile(obmol,"../xsaa.sdf")
while notatend:
    print(obmol.GetMolWt())
    obmol = ob.OBMol()
    notatend = obconversion.Read(obmol)
```

Pybel

```python
from openbabel import pybel

for molecule in pybel.readfile("sdf","../xsaa.sdf"):
    print(molecule.molwt)
```

Find information on all of the atoms and bonds connected to a particular atom

First of all, look at all of the classes in the *Open Babel API* that end with "Iter". You should use these whenever you need to do something like iterate over all of the atoms or bonds connected to a particular atom, iterate over all the atoms in a molecule, iterate over all of the residues in a protein, and so on.

As an example, let's say we want to find information on all of the bond orders and atoms connected to a particular OBAtom called 'obatom'. The idea is that we iterate over the neighbouring atoms using OBAtomAtomIter, and then find the bond between the neighbouring atom and 'obatom'. Alternatively, we could have iterated over the bonds (OBAtomBondIter), but we would need to look at the indices of the two atoms at the ends of the bond to find out which is the neighbouring atom:

```python
for neighbour_atom in ob.OBAtomAtomIter(obatom):
   print(neighbour_atom.GetAtomicNum())
   bond = obatom.GetBond(neighbour_atom)
   print(bond.GetBondOrder())
```

Examples from around the web

- Noel O'Blog - Hack that SD file, Just How Unique are your Molecules Part I and Part II, Calculate circular fingerprints with Pybel, Molecular Graph-ics with Pybel, and Generating InChI's Mini-Me, the InChIKey.
- Filter erroneous structures from the ZINC database

- Quantum Pharmaceuticals - Investigation of datasets for hERG binding
- cclib - Given the coordinates, charge, and multiplicity, how to create the corresponding OBMol
- Florian Nigsch wrote an implementation of Murcko fragments using Pybel
- Andrew Dalke's Chemical Toolkit Rosetta contains several examples of Python code using openbabel.py and pybel

Split an SDF file using the molecule titles

The following was a request on the CCL.net list:

> Hi all, Does anyone have a script to split an SDFfile into single sdfs named after each after each individual molecule as specified in first line of parent multi file?

The solution is simple…

```
from openbabel import pybel
for mol in pybel.readfile("sdf", "bigmol.sdf"):
   mol.write("sdf", "%s.sdf" % mol.title)
```

An implementation of RECAP

TJ O'Donnell (of gNova) has written an implementation of the RECAP fragmentation algorithm in 130 lines of Python. The code is at [1].

TJ's book, "Design and Use of Relational Databases in Chemistry", also contains examples of Python code using Open Babel to create and query molecular databases (see for example the link to Open Babel code in the Appendix).

11.4 Java

The `openbabel.jar` file in the Open Babel distribution allows you to use the Open Babel C++ library from Java or any of the other JVM languages (Jython, JRuby, BeanShell, etc.).

11.4.1 Quickstart Example

Let's begin by looking at an example program that uses Open Babel. The following program carries out file format conversion, iteration over atoms and SMARTS pattern matching:

```
import org.openbabel.*;

public class Test {

   public static void main(String[] args) {
        // Initialise
        System.loadLibrary("openbabel_java");

        // Read molecule from SMILES string
        OBConversion conv = new OBConversion();
        OBMol mol = new OBMol();
        conv.SetInFormat("smi");
        conv.ReadString(mol, "C(Cl)(=O)CCC(=O)Cl");
```

(continues on next page)

(continued from previous page)

```
       // Print out some general information
       conv.SetOutFormat("can");
       System.out.print("Canonical SMILES: " +
         conv.WriteString(mol));
       System.out.println("The molecular weight is "
         + mol.GetMolWt());
       for(OBAtom atom : new OBMolAtomIter(mol))
           System.out.println("Atom " + atom.GetIdx()
             + ": atomic number = " + atom.GetAtomicNum()
             + ", hybridisation = " + atom.GetHyb());

       // What are the indices of the carbon atoms
       // of the acid chloride groups?
       OBSmartsPattern acidpattern = new OBSmartsPattern();
       acidpattern.Init("C(=O)Cl");
       acidpattern.Match(mol);

       vectorvInt matches = acidpattern.GetUMapList();
       System.out.println("There are " + matches.size() +
                          " acid chloride groups");
       System.out.print("Their C atoms have indices: ");
       for(int i=0; i<matches.size(); i++)
           System.out.print(matches.get(i).get(0) + " ");
   }
}
```

Output:

```
Canonical SMILES: ClC(=O)CCC(=O)Cl
The molecular weight is 154.9793599
Atom 1: atomic number = 6, hybridisation = 2
Atom 2: atomic number = 17, hybridisation = 0
Atom 3: atomic number = 8, hybridisation = 2
Atom 4: atomic number = 6, hybridisation = 3
Atom 5: atomic number = 6, hybridisation = 3
Atom 6: atomic number = 6, hybridisation = 2
Atom 7: atomic number = 8, hybridisation = 2
Atom 8: atomic number = 17, hybridisation = 0
There are 2 acid chloride groups
Their C atoms have indices: 1 6
```

11.4.2 Installation

Windows

`openbabel.jar` is installed along with the OpenBabelGUI on Windows, typically in `C:/Program Files (x86)/OpenBabel-2.3.2`. As an example of how to use `openbabel.jar`, download OBTest.java and compile and run it as follows:

```
C:\> set CLASSPATH=C:\Program Files (x86)\OpenBabel-2.3.2\openbabel.jar;.
C:\> "C:\Program Files\Java\jdk1.5.0_16\bin\javac.exe" OBTest.java
C:\> "C:\Program Files\Java\jdk1.5.0_16\bin\java.exe" OBTest
Running OBTest...
```

(continues on next page)

(continued from previous page)

```
Benzene has 6 atoms.
C:\>
```

MacOSX and Linux

The following instructions describe how to compile and use these bindings on MacOSX and Linux:

1. `openbabel.jar` is included in the Open Babel source distribution in `scripts/java`. To compile a Java application that uses this (e.g. the example program shown above), use a command similar to the following:

   ```
   javac Test.java -cp ../openbabel-2.3.2/scripts/java/openbabel.jar
   ```

2. To run the resulting `Test.class` on MacOSX or Linux you first need to compile the Java bindings as described in the section *Compile language bindings*. This creates `lib/libopenbabel_java.so` in the build directory.

3. Add the location of `openbabel.jar` to the environment variable CLASSPATH, not forgetting to append the location of `Test.class` (typically "."):

   ```
   export CLASSPATH=/home/user/Tools/openbabel-2.3.2/scripts/java/openbabel.jar:.
   ```

4. Add the location of `libopenbabel_java.so` to the environment variable LD_LIBRARY_PATH. Additionally, if you have not installed Open Babel globally you should set BABEL_LIBDIR to the location of the Open Babel library and BABEL_DATADIR to the `data` directory.

5. Now, run the example application. The output should be as shown above.

11.4.3 API

`openbabel.jar` provides direct access to the C++ Open Babel library from Java through the namespace **org.openbabel**. This binding is generated using the SWIG package and provides access to almost all of the Open Babel interfaces from Java, including the base classes **:obapi:‘OBMol‘**, **:obapi:‘OBAtom‘**, **:obapi:‘OBBond‘**, and **:obapi:‘OBResidue‘**, as well as the conversion framework **:obapi:‘OBConversion‘**.

Essentially any call in the C++ API is available to Java programs with very little difference in syntax. As a result, the principal documentation is the *Open Babel C++ API documentation*. A few differences exist, however:

- Global variables, global functions and constants in the C++ API can be found in **org.openbabel.openbabel_java**. The variables are accessible through get methods.
- When accessing various types of **:obapi:‘OBGenericData‘**, you will need to cast them to the particular subclass using the global functions, *toPairData*, *toUnitCell*, etc.
- The Java versions of the iterator classes in the C++ API (that is, all those classes ending in *Iter*) implement the *Iterator* and *Iterable* interfaces. This means that the following *foreach* loop is possible:

  ```
  for(OBAtom atom : new OBMolAtomIter(mol)) {
      System.out.println(atom.GetAtomicNum());
  }
  ```

- To facilitate use of the **:obapi:‘OBMolAtomBFSIter‘**, *OBAtom* has been extended to incorporate a *CurrentDepth* value, accessible through a get method:

```
for(OBAtom atom : new OBMolAtomBFSIter(mol)) {
    System.out.println(atom.GetCurrentDepth());
}
```

11.5 Perl

11.5.1 Installation

The Perl bindings are available only on MacOSX and Linux. (We could not get them to work on Windows.) See *Compile language bindings* for information on how to configure CMake to compile and install the Perl bindings.

11.5.2 Using Chemistry::OpenBabel

The Chemistry::OpenBabel module is designed to allow Perl scripts to use the C++ Open Babel library. The bindings are generated using the SWIG package and provides access to almost all of the Open Babel interfaces via Perl, including the base classes OBMol, OBAtom, OBBond, and OBResidue, as well as the conversion framework OBConversion.

PerlMol

For developing chemistry in Perl, you should also look at the PerlMol project.

As such, essentially any call in the C++ API is available to Perl access with very little difference in syntax. This guide is designed to give examples of common Perl syntax for Chemistry::OpenBabel and pointers to the appropriate sections of the *API documentation*.

The example script below creates atoms and bonds one-by-one using the OBMol, OBAtom, and OBBond classes.

```
#!/usr/bin/perl

use Chemistry::OpenBabel;

my $obMol = new Chemistry::OpenBabel::OBMol;

$obMol->NewAtom();
$numAtoms = $obMol->NumAtoms(); # now 1 atom

my $atom1 = $obMol->GetAtom(1); # atoms indexed from 1
$atom1->SetVector(0.0, 1.0, 2.0);
$atom1->SetAtomicNum(6); # carbon atom

$obMol->NewAtom();
$obMol->AddBond(1, 2, 1); # bond between atoms 1 and 2 with bond order 1
$numBonds = $obMol->NumBonds(); # now 1 bond

$obMol->Clear();
```

More commonly, Open Babel can be used to read in molecules using the OBConversion framework. The following script reads in molecular information from a SMILES string, adds hydrogens, and writes out an MDL file as a string.

```
#!/usr/bin/perl
```

(continues on next page)

(continued from previous page)

```
use Chemistry::OpenBabel;

my $obMol = new Chemistry::OpenBabel::OBMol;
my $obConversion = new Chemistry::OpenBabel::OBConversion;
$obConversion->SetInAndOutFormats("smi", "mdl");
$obConversion->ReadString($obMol, "C1=CC=CS1");

$numAtoms = $obMol->NumAtoms(); # now 5 atoms

$obMol->AddHydrogens();
$numAtoms = $obMol->NumAtoms(); # now 9 atoms

my $outMDL = $obConversion->WriteString($obMol);
```

The following script writes out a file using a filename, rather than reading and writing to a Perl string.

```
#!/usr/bin/perl

use Chemistry::OpenBabel;

my $obMol = new Chemistry::OpenBabel::OBMol;
my $obConversion = new Chemistry::OpenBabel::OBConversion;
$obConversion->SetInAndOutFormats("pdb", "mol2");
$obConversion->ReadFile($obMol, "1ABC.pdb");

$obMol->AddHydrogens();

print "# of atoms: $obMol->NumAtoms()";
print "# of bonds: $obMol->NumBonds()";
print "# of residues: $obMol->NumResidues()";

$obConversion->WriteFile($obMol, "1abc.mol2");
```

11.5.3 Examples

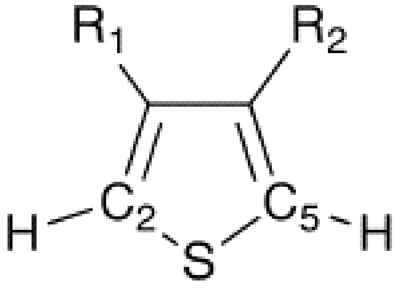
Output Molecular Weight for a Multi-Molecule SDF File

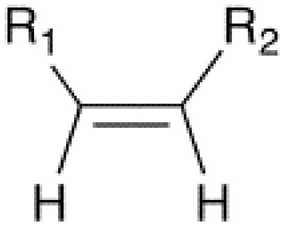
Let's say we want to print out the molecular weights of every molecule in an SD file. Why? Well, we might want to plot a histogram of the distribution, or see whether the average of the distribution is significantly different (in the statistical sense) compared to another SD file.

```
use Chemistry::OpenBabel;

my $obconversion = new Chemistry::OpenBabel::OBConversion;
$obconversion->SetInFormat("sdf");
my $obmol = new Chemistry::OpenBabel::OBMol;

my $notatend = $obconversion->ReadFile($obmol, "../xsaa.sdf");
while ($notatend) {
    print $obmol->GetMolWt(), "\n";
    $obmol->Clear();
    $notatend = $obconversion->Read($obmol);
}
```

Add and Delete Atoms

This script shows an example of deleting and modifying atoms to transform one structure to a related one. It operates on a set of substituted thiophenes, deletes the sulfur atom (note that R1 and R2 may contain sulfur, so the SMARTS pattern is designed to constrain to the ring sulfur), etc. The result is a substituted ethylene, as indicated in the diagrams.





```
use Chemistry::OpenBabel;

my $obMol = new Chemistry::OpenBabel::OBMol;
my $obConversion = new Chemistry::OpenBabel::OBConversion;
my $filename = shift @ARGV;

$obConversion->SetInAndOutFormats("xyz", "mol");
$obConversion->ReadFile($obMol, $filename);

for (1..$obMol->NumAtoms()) {
    $atom = $obMol->GetAtom($_);
    # look to see if this atom is a thiophene sulfur atom
    if ($atom->MatchesSMARTS("[#16D2]([#6D3H1])[#6D3H1]")) {
        $sulfurIdx = $atom->GetIdx();
    # see if this atom is one of the carbon atoms bonded to a thiophene sulfur
    } elsif ($atom->MatchesSMARTS("[#6D3H1]([#16D2][#6D3H1])[#6]") ) {
        if ($c2Idx == 0) { $c2Idx = $atom->GetIdx(); }
        else {$c5Idx = $atom->GetIdx(); }
    }
}

# Get the actual atom objects -- indexing will change as atoms are added and deleted!
$sulfurAtom = $obMol->GetAtom($sulfurIdx);
$c2Atom = $obMol->GetAtom($c2Idx);
$c5Atom = $obMol->GetAtom($c5Idx);

$obMol->DeleteAtom($sulfurAtom);

$obMol->DeleteHydrogens($c2Atom);
$obMol->DeleteHydrogens($c5Atom);
```

(continues on next page)

(continued from previous page)

```
$c2Atom->SetAtomicNum(1);
$c5Atom->SetAtomicNum(1);

$obConversion->WriteFile($obMol, "$filename.mol");
```

11.6 CSharp and OBDotNet

OBDotNet is a compiled assembly that allows Open Babel to be used from the various .NET languages (e.g. Visual Basic, C#, IronPython, IronRuby, and J#) on Windows, Linux and MacOSX. The current version is OBDotNet 0.4.

11.6.1 Installation

Windows

The `OBDotNet.dll` assembly provided on Windows was compiled using the .NET framework v3.5 for the x86 platform. To use it, you will need to compile your code using .NET v3.5 or newer and you will also need to target x86 (`/platform:x86`).

The following instructions describe how to compile a simple C# program that uses OBDotNet:

1. First you need to download and install the **`OpenBabelGUI version 2.3.2`**
2. Next create an example CSharp program that uses the Open Babel API (see below for one or use this link). Let's call this `example.cs`.
3. Copy `OBDotNet.dll` from the Open Babel installation into the same folder as `example.cs`.
4. Open a command prompt at the location of `example.cs` and compile it as follows:

```
C:\Work> C:\Windows\Microsoft.NET\Framework\v3.5\csc.exe
                             /reference:OBDotNet.dll /platform:x86 example.cs
```

5. Run the created executable, **`example.exe`**, to discover the molecule weight of propane:

```
C:\Work> example.exe
44.09562
```

If you prefer to use the MSVC# GUI, note that the Express edition does not have the option to choose x86 as a target. This will be a problem if you are using a 64-bit operating system. There's some information at Coffee Driven Development on how to get around this.

MacOSX and Linux

On Linux and MacOSX you need to use Mono, the open source implementation of the .NET framework, to compile the bindings. The following instructions describe how to compile and use these bindings:

1. `OBDotNet.dll` is included in the Open Babel source distribution in `scripts/csharp`. To compile a CSharp application that uses this (e.g. the example program shown below), use a command similar to the following:

```
gmcs example.cs /reference:../openbabel-2.3.2/scripts/csharp/OBDotNet.dll
```

2. To run this on MacOSX or Linux you need to compile the CSharp bindings as described in the section *Compile language bindings*. This creates `lib/libopenbabel_csharp.so` in the build directory.
3. Add the location of `OBDotNet.dll` to the environment variable MONO_PATH. Add the location of `libopenbabel_csharp.so` to the environment variable LD_LIBRARY_PATH. Additionally, if you have not installed Open Babel globally you should set BABEL_LIBDIR to the location of the Open Babel library and BABEL_DATADIR to the `data` directory.
4. Run `example.exe`:

```
$ ./example.exe
44.09562
```

11.6.2 OBDotNet API

The API is almost identical to the Open Babel *C++ API*. Differences are described here.

Using iterators

In OBDotNet, iterators are provided as methods of the relevant class. The full list is as follows:

- **OBMol** has `.Atoms()`, `.Bonds()`, `.Residues()`, and `.Fragments()`. These correspond to **:obapi:‘OBMolAtomIter‘**, **:obapi:‘OBMolBondIter‘**, **:obapi:‘OBResidueIter‘** and **:obapi:‘OBMolAtomDFSIter‘** respectively.
- **OBAtom** has `.Bonds()` and `.Neighbours()`. These correspond to **:obapi:‘OBAtomBondIter‘** and **:obapi:‘OBAtomAtomIter‘** respectively.

Such iterators are used as follows:

```
foreach (OBAtom atom in myobmol.Atoms())
    System.Console.WriteLine(atom.GetAtomType());
```

Other iterators in the C++ API not listed above can still be used through their IEnumerator methods.

Handling OBGenericData

To cast **:obapi:‘OBGenericData‘** to a specific subclass, you should use the `.Downcast <T>` method, where `T` is a subclass of **OBGenericData**.

Open Babel Constants

Open Babel constants are available in the class `openbabelcsharp`.

11.6.3 Examples

The following sections show how the same example application would be programmed in C#, Visual Basic and IronPython. The programs print out the molecular weight of propane (represented by the SMILES string “CCC”).

C#

```
using System;
using OpenBabel;

namespace MyConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            OBConversion obconv = new OBConversion();
            obconv.SetInFormat("smi");
            OBMol mol = new OBMol();
            obconv.ReadString(mol, "CCC");
            System.Console.WriteLine(mol.GetMolWt());
        }
    }
}
```

Visual Basic

```
Imports OpenBabel

Module Module1

    Sub Main()
        Dim OBConv As New OBConversion()
        Dim Mol As New OBMol()

        OBConv.SetInFormat("smi")
        OBConv.ReadString(Mol, "CCC")
        System.Console.Write("The molecular weight of propane is " & Mol.GetMolWt())
    End Sub

End Module
```

IronPython

```
import clr
clr.AddReference("OBDotNet.dll")

import OpenBabel as ob

conv = ob.OBConversion()
conv.SetInFormat("smi")
mol = ob.OBMol()
conv.ReadString(mol, "CCC")
print(mol.GetMolWt())
```

11.7 Ruby

As with the other language bindings, just follow the instructions at *Compile language bindings* to build the Ruby bindings.

Like any Ruby module, the Open Babel bindings can be used from a Ruby script or interactively using **irb** as follows:

```
$ irb
irb(main):001:0> require 'openbabel'
=> true
irb(main):002:0> c=OpenBabel::OBConversion.new
=> #<OpenBabel::OBConversion:0x2acedbadd020>
irb(main):003:0> c.set_in_format 'smi'
=> true
irb(main):004:0> benzene=OpenBabel::OBMol.new
=> #<OpenBabel::OBMol:0x2acedbacfa10>
irb(main):005:0> c.read_string benzene, 'c1ccccc1'
=> true
irb(main):006:0> benzene.num_atoms
=> 6
```

11.8 Updating to Open Babel 3.0 from 2.x

Open Babel 3.0 breaks the API in a number of cases, and introduces some new behavior behind-the-scenes. These changes were necessary to fix some long standing issues impacting chemical accuracy as well as performance.

Here we describe the main changes, and how to change existing code to adapt.

11.8.1 Removal of babel

The `babel` executable has been removed, and `obabel` should be used instead. Essentially **obabel** is a modern version of **babel** with additional capabilities and a more standard interface. Typically the only change needed is to place `-O` before the output filename:

```
$ babel -ismi tmp.smi -omol out.mol
$ obabel -ismi tmp.smi -omol -O out.mol
```

Specifically, the differences are as follows:

- **obabel** requires that the output file be specified with a `-O` option. This is closer to the normal Unix convention for commandline programs, and prevents users accidentally overwriting the input file.
- **obabel** is more flexible when the user needs to specify parameter values on options. For instance, the `--unique` option can be used with or without a parameter (specifying the criteria used). With **babel**, this only works when the option is the last on the line; with **obabel**, no such restriction applies. Because of the original design of **babel**, it is not possible to add this capability in a backwards-compatible way.
- **obabel** has a shortcut for entering SMILES strings. Precede the SMILES by -: and use in place of an input file. The SMILES string should be enclosed in quotation marks. For example:

  ```
  obabel -:"O=C(O)c1ccccc1OC(=O)C" -ocan
  ```

  More than one can be used, and a molecule title can be included if enclosed in quotes:

```
obabel -:"O=C(O)c1ccccc1OC(=O)C aspirin" -:"Oc1ccccc1C(=O)O salicylic acid"
       -ofpt
```

- **`obabel`** cannot use concatenated single-character options.

11.8.2 Python module

In OB 3.x, both `openbabel.py` and `pybel.py` live within the `openbabel` module:

```
# OB 2.x
import openbabel as ob
import pybel

# OB 3.0
from openbabel import openbabel as ob
from openbabel import pybel
```

While more verbose, the new arrangement is in line with standard practice and helps avoid conflict with a different Python project, PyBEL.

11.8.3 Handling of elements and related information

The API for interconverting atomic numbers and element symbols has been replaced for performance reasons. The `OBElementTable` class has been removed and its associated functions are now available through the `OBElements` namespace:

```
// OB 2.x
OBElementTable etab;
const char *elem = etab.GetSymbol(6);
unsigned int atomic_num = etab.GetAtomicNum(elem);

// OB 3.0
#include <openbabel/elements.h>
const char *elem = OBElements::GetSymbol(6);
unsigned int atomic_num = OBElements::GetAtomicNum(elem);
```

Furthermore, the OBAtom API convenience functions for testing for particular elements (e.g. `IsHydrogen()`) have been removed. Instead, `OBAtom::GetAtomicNum()` should be used along with an element constant or atomic number:

```
// OB 2.x
if (atom->IsCarbon()) {...

// OB 3.0
if (atom->GetAtomicNum() == OBElements.Carbon) {...
//   or
if (atom->GetAtomicNum() == 6) {...
```

Handling of isotope information now longer uses `OBIsotopeTable` but is accessed through the `OBElements` namespace:

```
// OB 2.x
OBIsotopeTable isotab;
isotab.GetExactMass(6, 14);
```

(continues on next page)

(continued from previous page)

```
// OB 3.0
double exact = OBElements.GetExactMass(OBElements.Carbon, 14);
```

11.8.4 Atom classes

In OB 2.x, atom class information was stored as part of an `OBAtomClassData` object attached to an `OBMol` and accessed via `OBMol.GetData("Atom Class")`. In OB 3.0, atom class information is instead stored as an `OBPairInteger` associated with an `OBAtom` and accessed via `OBAtom.GetData("Atom Class")`.

11.8.5 OBAtom valence and degree methods

OB 2.x referred to the function that returned the explicit degree of an atom as `GetValence()`. This was confusing, at best. To find the explicit valence, the `BOSum()` method was required. OB 3.0 avoids this confusion by renaming methods associated with degree or valence:

- `OBAtom::GetExplicitValence()` (OB 2.x `BOSum()`)
- `OBAtom::GetExplicitDegree()` (OB 2.x `GetValence()`)
- `OBAtom::GetHvyDegree()` (OB 2.x `GetHvyValence()`)
- `OBAtom::GetHeteroDegree()` (OB 2.x `GetHeteroValence()`)

11.8.6 Molecule, atom and bond flags

The “Unset” methods for molecule, atom and bond flags have been removed. Instead, a value of `false` should be passed to the corresponding “Set” method. For example, `OBMol::UnsetAromaticPerceived()` in OB 2.x is now `OBMol::SetAromaticPerceived(false)`.

11.8.7 Removal of deprecated methods

Several deprecated methods have been removed. For the most part, an equivalent function with a different name is present in the API:

- `OBBond::GetBO()`/`SetBO()` removed. `OBBond::GetBondOrder()`/`SetBondOrder()` should be used instead.
- `OBAtom::GetCIdx()` removed. `OBAtom::GetCoordinateIdx()` should be used instead.
- `OBBitVec::Empty()` removed. `OBBitVec::IsEmpty()` should be used instead.
- `OBBitVec::BitIsOn()` removed. `OBBitVec::BitIsSet()` should be used instead.

11.8.8 Handling of implicit hydrogens

With OB 3.0, the number of implicit hydrogens is stored as a property of the atom. This value can be interrogated and set independently of any other property of the atom. This is how other mature cheminformatics toolkits handle implicit hydrogens. In contrast, in OB 2.x this was a derived property worked out from valence rules and some additional flags set on an atom to indicate non-standard valency.

From the point of view of the user, the advantage of the 2.x approach was that the user never needed to consider the implicit hydrogens; their count was calculated based on the explicit atoms (a behavior known as “floating valence”).

The disadvantage was that it was difficult for the user to specify non-standard valencies, may have papered-over problems with the data, gave rise to subtle bugs which were not easily addressed and had poorer performance.

As an example of how the behavior has changed, let's look at creating a bond. If we read the SMILES string `C.C`, create a bond between the two atoms and write out the SMILES string, we get different answers for OB 2.x (`CC`) versus OB 3.0 (`[CH4][CH4]`). OB 2.x just works out the count based on standard valence rules. With OB 3.0, there were four implicit hydrogens on each carbon before we made the bond, and there still are four - they didn't go anywhere and weren't automatically adjusted.

While this may seem like a major change, adapting code to handle the change should be straightforward: adding or removing a bond should be accompanied by incrementing or decrementing the implicit hydrogen count by the bond order. This also applies to deleting an atom, since this deletes any bonds connected to it. Note that care should be taken not to set the hydrogen count to a negative value when decrementing.

```
unsigned int bondorder = 1;
mol->AddBond(1, 2, bondorder);
OBAtom* start = mol->GetAtom(1);
unsigned int hcount = start->GetImplicitHCount();
start->SetImplicitHCount(bondorder >= hcount ? 0 : hcount - bondorder);
OBAtom* end = mol->GetAtom(2);
hcount = end->GetImplicitHCount();
end->SetImplicitHCount(bondorder >= hcount ? 0 : hcount - bondorder);
```

For the particular case of creating a new atom, it is worth noting that the implicit hydrogen count defaults to zero and that users must set it themselves if necessary. To help with this situation a convenience function has been added to OBAtom that sets the implicit hydrogen count to be consistent with normal valence rules. TODO

Regarding specific API functions, the following have been removed:

- `OBAtom::SetImplicitValence()`, `GetImplicitValence()`
- `OBAtom::IncrementImplicitValence()`, `DecrementImplicitValence()`
- `OBAtom::ForceNoH()`, `HasNoHForce()`, `ForceImplH()`, `HasImplHForced()`
- `OBAtom::ImplicitHydrogenCount()`

The following have been added:

- `OBAtom::SetImplicitHCount()`, `GetImplicitHCount()`

11.8.9 Handling of aromaticity

Molecule modification no longer clears the aromaticity perception flag. If the user wishes to force reperception after modification, then they should call `OBMol::SetAromaticPerceived(false)`.

Chapter 12

Cheminformatics 101

An introduction to the computer science and chemistry of chemical information systems

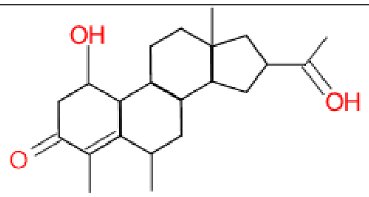
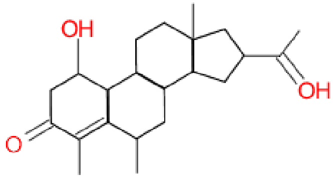
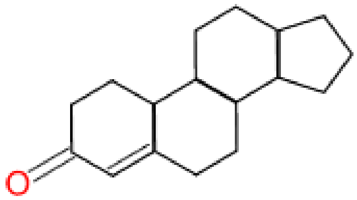
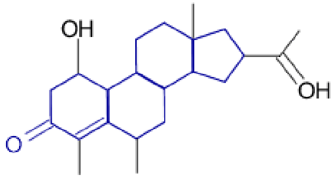
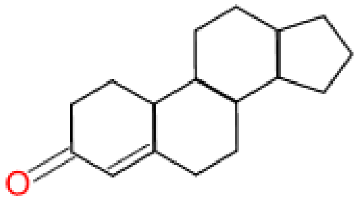
The original version of this introduction to cheminformatics can be found on the eMolecules website. It is included here with the permission of the author.

12.1 Cheminformatics Basics

12.1.1 What is Cheminformatics?

Cheminformatics is a cross between Computer Science and Chemistry – the process of storing and retrieving information about chemical compounds.

Information Systems are concerned with storing, retrieving, and searching information, and with storing *relationships* between bits of data. For example:

Operation	Classical Information System		Chemical Information System	
Store	Name = 'Jimmy Carter'	Stores text, numbers, dates, . . .		Stores chemical compounds and information about them
Retrieve	Find record #13282	Retrieves 'Jimmy Carter'	Find CC(=O)C4CC3C2CC(C)C1=C(C). . . C(=O)CC(O)C1C2CCC3(C)C4	Retrieves: 
Search	Find Presidents named 'Bush'	George Bush and George W. Bush	Find molecules containing 	Retrieves: 
Relationship	Year Carter was elected	Answer: Elected in 1976	What's the logP(o/w) of 	Answer: logP(o/W) = 2.62

12.1.2 How is Cheminformatics Different?

There are four key problems a cheminformatics system solves:

1. **Store a Molecule**

   Computer scientists usually use the *valence model* of chemistry to represent compounds. The next section *Representing Molecules*, discusses this at length.

2. **Find exact molecule**

   If you ask, "Is Abraham Lincoln in the database?" it's not hard to find the answer. But, given a specific molecule, is it in the database? What do we know about it? This may seem seem simple at first glance, but it's not, as we'll see when we discuss tautomers, stereochemistry, metals, and other "flaws" in the valence model of chemistry.

3. **Substructure search**

   If you ask, "Is anyone named Lincoln in the database?" you usually expect to find the former President and a number of others - this is called a *search* rather than a *lookup*. For a chemical informatics system, we have a *substructure search*: Find all molecules containing a partial molecule (the "substructure") drawn by the user. The substructure is usually a functional group, “scaffold”, or core structure representing a class of molecules. This too is a hard problem, *much* harder than most text searches, for reasons that go to the very root of mathematics and the theory of computability.

4. **Similarity search**

   Some databases can find similar-sounding or misspelled words, such as “Find Lincon” or “find Cincinati”, which respectively might find Abraham Lincoln and Cincinnati. Many chemical information systems can find molecules similar to a given molecule, ranked by similarity. There are several ways to measure molecular similarity, discussed further in the section on *Molecular Similarity*.

12.2 Representing Molecules

12.2.1 What is a Molecule?

One of the greatest achievements in chemistry was the development of the *valence model* of chemistry, where a molecule is represented as *atoms* joined by semi-rigid *bonds* that can be single, double, or triple. This simple mental model has little resemblance to the underlying quantum-mechanical reality of electrons, protons and neutrons, yet it has proved to be a remarkably useful approximation of how atoms behave in close proximity to one another, and has been the foundation of chemical instruction for well over a century.

The valence model is also the foundation of modern chemical information systems. When a Computer Scientist approaches a problem, the first task is to figure out a *datamodel* that represents the problem to be solved as *information*. To the Computer Scientist, the valence model naturally transforms into a *graph*, where the *nodes* are atoms and the *edges* are bonds. Computer Scientists know how to manipulate graphs - mathematical graph theory and computer science have been closely allied since the invention of the digital computer.

> *There are atoms and space. Everything else is opinion.*
>
> —Democritus

However, the valence model of chemistry has many shortcomings. The most obvious is aromaticity, which quickly required adding the concept of a non-integral “aromatic” distributed bond, to the single/double/triple bonds of the simple valence model. And that was just the start - tautomers, ferrocenes, charged molecules and a host of other common molecules simply don’t fit the valence model well.

This complicates life for the computer scientist. As we shall see, they are the source of most of the complexity of modern cheminformatics systems.

12.2.2 Older systems: Connection Tables

Most of the early (and some modern) representations of molecules were in a *connection table*, literally, a table enumerating the atoms, and a table enumerating the bonds and which atoms each bond connected. Here is an example of connection-table (CTAB) portion of an MDL “SD” file (the data portion is not shown here):

```
MOLCONV

3 2  0  0  1  0              1 V2000
5.9800   -0.0000   -0.0000 Br  0  0  0  0  0  0
4.4000   -0.6600    0.8300 C   0  0  0  0  0  0
3.5400   -1.3500   -0.1900 C   0  0  0  0  0  0
1  2  1  0
2  3  1  0
```

This simple example illustrates most of the key features. The molecule has three atoms, two bonds, and is provided with three-dimensional (x,y,z) coordinates. MDL provides extensive documentation for their various CTFile formats if you are interested in the details.

Connection tables can capture the valence model of chemistry fairly well, but they suffer from three problems:

1. They are very inefficient, taking on the order of a dozen or two of bytes of data per atom and per bond. Newer line notations (discussed below) represent a molecules with an average of 1.2 to 1.5 bytes per atom, or 6-8 bytes per atom if coordinates are added.

2. Many suffered from lack of specificity. For example, since hydrogens are often not specified, there can be ambiguity as to the electronic state of some molecules, because the connection-table format does not explicitly state the valence assumptions.

3. Most mix the concept of *connectivity* (what the atoms are and how they are connected) with other data such as 2D and 3D coordinates. For example, if you had two different conformers of a molecule, most connection tables would require you to specify the entire molecule twice, even though the connection table is identical in both.

12.2.3 Line Notations: InChI, SMILES, WLN and others

A *line notation* represents a molecule as a single-line string of characters.

WLN - Wisswesser Line Notation WLN, invented by William J. Wisswesser in the early 1950's, was the first comprehensive line notation, capable of representing arbitrarily complex molecules correctly and compactly.

```
1H = CH4 Methane
2H = CH3-CH3 Ethane
3H = CH3-CH2-CH3 Propane
QVR BG CG DG EG FG = C7HCl5O2 Pentachlorbenzoate
```

WLN was the first line notation to feature a *canonical form*, that is, the rules for WLN meant there was only one "correct" WLN for any particular molecule. Those versed in WLN were able to write molecular structure in a line format, communicate molecular structure to one another and to computer programs. Unfortunately, WLN's complexity prevented widespread adoption. The rules for correct specification of WLN filled a small book, encoding those rules into a computer proved difficult, and the rules for the *canonicalization* were computationally intractable.

SMILES - Simplified Molecular Input Line Entry System The best-known line notation today is SMILES. It was by Arthur and David Weininger in response to a need for a simpler, more "human accessible" notation than WLN. While SMILES is not trivial to learn and write, most chemists can create correct SMILES with just a few minutes training, and the entire SMILES language can be learned in an hour or two. You can read more details here. Here are some examples:

```
C          methane
CC         ethane
C=C        ethene
Oc1ccccc1  phenol
```

SMILES, like WLN, has a *canonical form*, but unlike WLN, Weininger relied on the computer, rather than the chemist, to convert a non-canonical SMILES to a canonical SMILES. This important separation of duties was key to making SMILES easy to enter. (Read more about canonicalization below.)

InChI InChI is the latest and most modern of the line notations. It resolves many of the chemical ambiguities not addressed by SMILES, particularly with respect to stereo centers, tautomers and other of the "valence model problems" mentioned *above*.

You can read more about InChI at the Official Web Site, or on the Unofficial InChI FAQ page.

12.2.4 Canonicalization

A critical feature of line notations is *canonicalization* - the ability to choose one "blessed" representation from among the many. Consider:

```
OCC   ethanol
CCO   ethanol
```

Both of these SMILES represent the same molecule. If we could all agree that one of these was the "correct" or "canonical" SMILES for ethanol, then we would *always store it the same way* in our database. More importantly, if we want to ask, "Is ethanol in our database" we know that it will only be there once, and that we can generate the canonical SMILES for ethanol and look it up.

(Note that in theory one can create a canonical connection table, too, but it's not as useful since informatics systems usually have trouble indexing BLOBs - large objects.)

12.2.5 Line Notation versus Connection Tables: A practical matter

Why are line notations preferred over connection-table formats? In theory, either could express the same information. But there are practical difference, mostly related to the complexity of "parsing" a connection table. If you know that the whole molecule is on one line of a file, it's easy to parse.

Line notations are also very nice for database applications. Relational databases have datatypes that, roughly speaking, are divided into numbers, text, and "everything else", also known as "BLOBs" (Binary Large OBjects). You can store line notations in the "text" fields much more easily than connection tables.

Line notations also have pragmatic advantages. Modern Unix-like systems (such as UNIX, Linux and Cygwin) have a number of very powerful "filter" text-processing programs that can be "piped" together (connected end-to-end) to perform important tasks. For example, to count the number of molecules containing aliphatic nitrogen in a SMILES file, I can simply:

```
grep N file.smi | wc
```

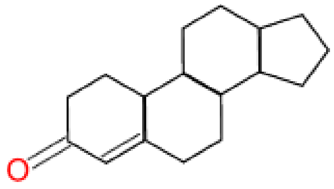
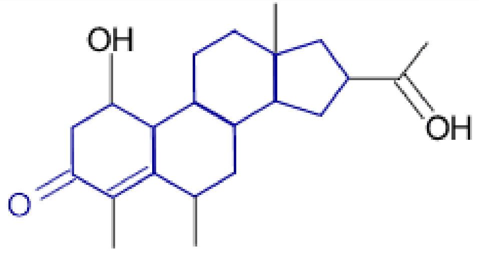
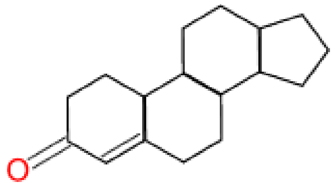
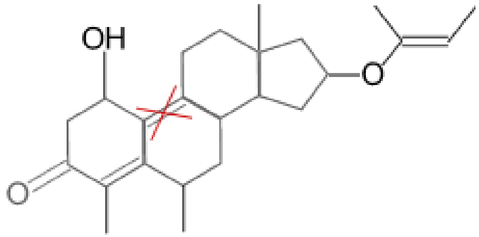
(**grep** looks for a particular expression, in this case `N`, and prints any line that contains it, and **wc** ("word count") counts the number of words and lines.)

This is just a simple example of the power available via "script" programs using "filters" on Unix-like systems. Unix filters are much less useful for connection-table formats, because each molecule is spread over many lines.

12.2.6 Query Languages: SMARTS

In addition to a typographical way to represent molecules, we also need a way to enter *queries* about molecules, such as, "Find all molecules that contain a phenol."

With text, we're familiar with the concept of typing a partial word, such as "ford" to find "Henry Ford" as well as "John Hartford". For chemistry, we can also specify partial structures, and find anything that contains them. For example:

Query	Database	Matches?
		**YES** (matched portion highlighted in blue)
		**NO** (double bond indicated doesn't match)

eMolecules, Inc.

eMolecules is a one-stop shop for suppliers and information for over 8 million chemical compounds. Under the hood is a chemical registration technology based on Open Babel.

The simplest query language for chemistry is SMILES itself: Just specify a structure, such as `Oc1ccccc1`, and search. This is how eMolecules' basic searching works (see Sidebar). It's simple and, because of the high-performance indexes in eMolecules, is also very fast.

However, for general-purpose cheminformatics, one needs more power. What if the substructure you're looking for isn't a valid molecule? For example `ClccBr` (1,2- substitution on an aromatic ring) isn't a whole molecule, since the concept of aromaticity is only sensible in the context of a whole ring system.

Or what if the thing we're looking for isn't a simple atom such as Br, but rather a concept like "Halogen"? Or, "A terminal methyl"?

To address this, cheminformatics systems have special *query languages*, such as SMARTS (SMiles ARbitrary Target Specification). SMARTS is a close cousin to SMILES, but it has *expressions* instead of simple atoms and bonds. For example, `[C,N]` will find an atom that is either carbon or nitrogen.

12.2.7 IUPAC Names, Trade Names, Common Names

Chemistry also has three other important name systems:

IUPAC Names IUPAC (the International Union of Pure and Applied Chemistry) established a naming convention that is widely used throughout chemistry. Any chemical can be named, and all IUPAC names are unambiguous. This textual representation is aimed at humans, not computers: Chemists versed in IUPAC nomenclature (which is widely taught) can read an IUPAC name and visualize or draw the molecule.

Trade Names Names such as Tylenol™ and Valium™ are given to compounds and formulations by manufacturers for marketing and sales purposes, and for regulatory purposes.

Common names Names such as “aspirin” or “alcohol” for substances that are in widespread use.

12.3 Substructure Searching with Indexes

12.3.1 What is Indexing?

Indexing is pre-computing the answers to portions of expected questions *before* they’re asked, so that when the question comes, it can be answered quickly.

Take your favorite search engine (AOL, Yahoo!, Google, MSN, . . .) for example. Without indexing, they might wait until you ask for “John Hartford Bluegrass”, then start searching the web, and in a year or two find all the web pages about the deceased banjo/fiddle player and steamboat captain. That would probably not impress you.

Instead, these search engines search the web *before* you ask your question, and build an *index* of the words they find. When you type in “Bluegrass John Hartford”, they already know all of the pages that have “John”, all of the pages with “Hartford”, and all of the pages with “Bluegrass”. Instead of searching, they examine their index, and find pages that are on *all three* lists, and quickly find your results. (NB: It’s actually a lot more complex, but this illustrates the main idea of indexing.)

12.3.2 Indexes for Chemicals

Instead of indexing words, cheminformatics systems index *substructures*. Although there are many schemes for doing this, cheminformatics systems all use the same fundamental principle: they *decompose the molecule* into smaller bits, and index those.

Roughly speaking, a cheminformatics system will index each of the substructures (fragments) above, so that every molecule that contains each fragment is known.

When a query is entered, the cheminformatics system breaks apart the query using the same technique, to find all of the fragments in the query. It then checks its index for each fragment, and combines the lists it finds to get only those molecules that have *all* of those fragments.

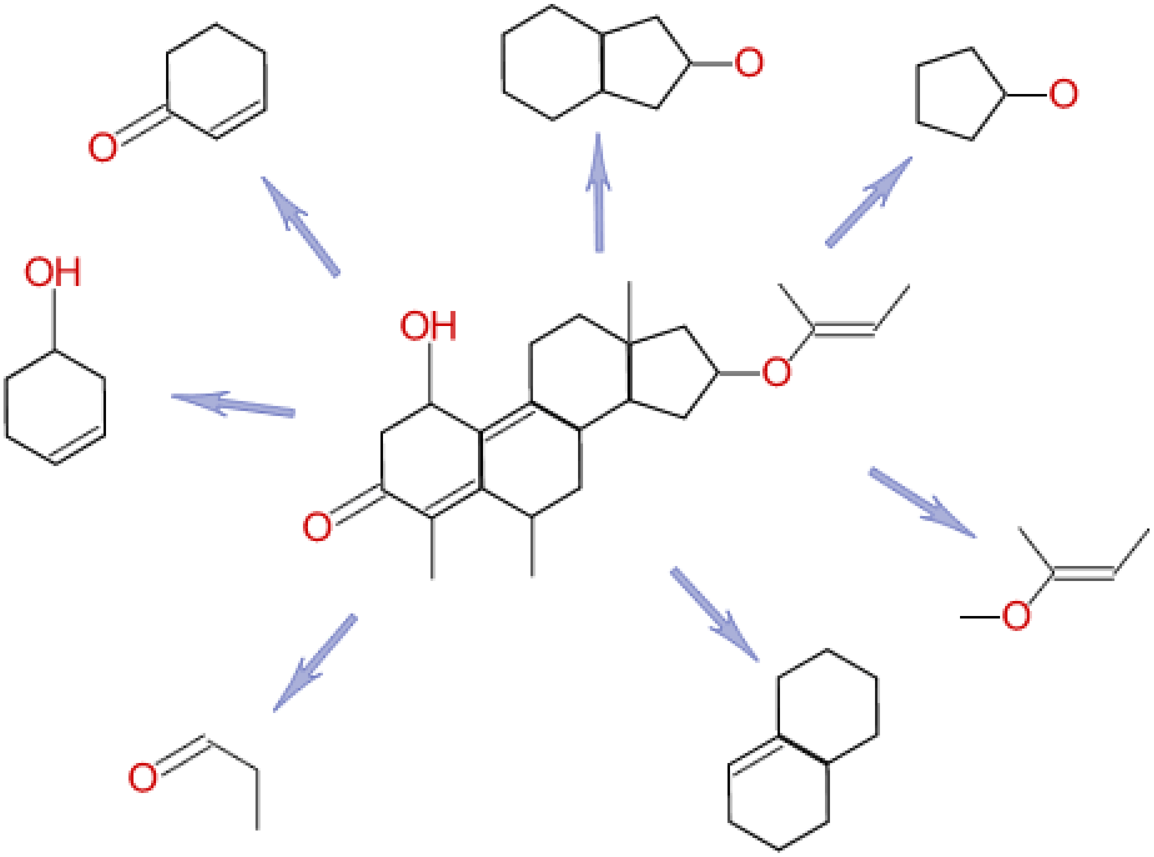
This doesn’t mean that all molecules returned by the index actually are matches. In the language of databases, we say the index will return *false positives*, candidate molecules that don’t actually match the substructure search.

Consider our example of searching for “John Hartford” - the index might return many pages that have both “John” and “Hartford”, yet have nothing to do with bluegrass music or steamboats. For example, it might return a page containing, “President John F. Kennedy visited Hartford, Connecticut today. . . ”. To confirm that the search system has found something relevant, it must check the pages return from the index to ensure that the specific phrase “John Hartford” is present. However, notice that this is *much* faster than searching every page, since the overwhelming majority of web pages were instantly rejected because they have neither “John” nor “Hartford” on them.

Similarly, a chemical fragment index serves to find only the most *likely* molecules for our substructure match - anything that the index didn’t find is definitely not a match. But we still have to examine each of the molecules returned by the indexing system and verify that the complete substructure for which we are searching is present.

12.3.3 NP-Complete - A Little about Computability

Searching through a page of text for the words “John Hartford” is pretty easy for a modern computer. Although false positives returned by the index are a nuisance and impair performance, they are not a catastrophe. Not so for


O
O
O
OH
OH
O
O
O
O

substructure matching. Unfortunately, substructure matching falls into a category of “hard” mathematical problems, which means false positives from the index are a big problem.

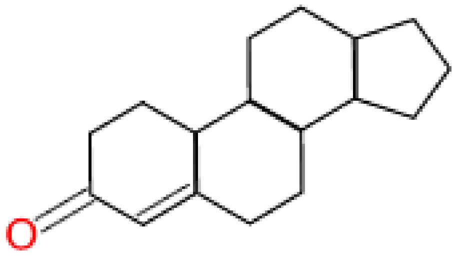
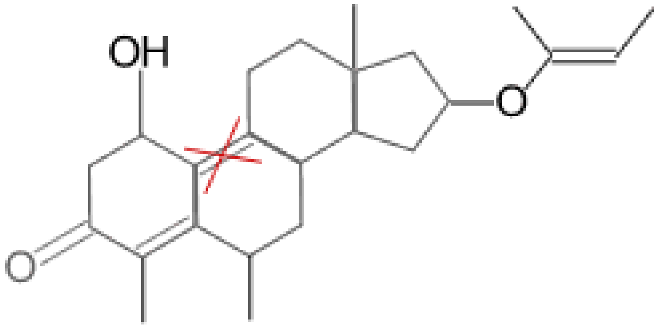
Substructure matching (finding a certain functional group within a molecule) is an example of what mathematicians call graph isomorphism, and is in a class of problems called NP Complete. Roughly speaking, this means the time it takes to do a substructure search is non-polynomial, i.e. exponential in the number of atoms and bonds. To see why this is a computational disaster, compare two tasks, one that takes polynomial time, k_1*N^2, versus one that takes exponential time k_2*2^N. Our polynomial task is bad enough: If we double N, it takes *four times* as long to solve. But the exponential task is worse: *Every time we add an atom it doubles*. So going from one atom to two doubles the time, and going from 100 atoms to 101 atoms doubles the time. Even if we can get k_2 down to a millionth of k_1, we’re still in trouble - a million is just 2^{20} or twenty atoms away.

It has been mathematically proven that substructure searching is in the set of NP Complete problems, so there’s no point wasting our time searching for a polynomial algorithm. The good news is that most molecules have “low connectivity”, meaning most atoms have fewer than four bonds, unlike the weird and twisted graphs that mathematicians consider. In practice, most substructure matching can be done in polynomial time around N^2 or N^3. But even with this improvement, substructure matching is an “expensive” time-consuming task for a computer.

The key point is that indexing is particularly important for cheminformatics systems. The typical modern computer can only examine a few thousand molecules per second, so examining millions of molecules one-by-one is out of the question. The indexing done by a modern cheminformatics system is the key to its performance.

12.4 Molecular Similarity

Substructure searching is a very powerful technique, but sometimes it misses answers for seemingly trivial differences. We saw this earlier with the following:

Query	Target
	
We’re looking for steroids. . .	But we don’t find this one because of the double bond

It is somewhat like searching for “221b Baker Street” and finding nothing because the database contains “221B Baker Street” and the system doesn’t consider “b” and “B” a match.

A good similarity search would find the target structure shown above, because even though it is not a substructure match, it is highly similar to our query.

There are many ways to measure similarity.

2D topology The best-known and most widely used similarity metrics compare the two-dimensional topology, that is, they only use the molecule’s atoms and bonds without considering its shape.

Tanimoto similarity is perhaps the best known as it is easy to implement and fast to compute. An excellent summary of 2D similarity metrics can be found in section 5.3 of the Daylight Theory Manual.

3D configuration One of the most important uses of similarity is in the discovery of new drugs, and a molecule's shape is critical to it's medicinal value (see QSAR).

3D similarity searches compare the configuration (also called the "conformation") of a molecule to other molecules. The "electronic surface" of the molecule is the important bit - the part that can interact with other molecules. 3D searches compare the surfaces of two molecules, and how polarized or polarizable each bit of the surface is.

3D similarity searches are uncommon, for two reasons: It's difficult and it's slow. The difficulty comes from the complexity of molecular interactions - a molecule is not a fixed shape, but rather a dynamic object that changes according to its environment. And the slowness comes from the difficulty: To get better results, scientists employ more and more complex programs.

Physical Properties The above 2D and 3D similarity are based on the molecule's structure. Another technique compares the properties - either computed or measured or both - and declares that molecules with many properties in common are likely to have similar structure. It is the idea of QSAR taken to the database.

Clustering "Clustering" is the process of differentiating a set of things into groups where each group has common features. Molecules can be clustered using a variety of techniques, such as common 2D and/or 3D features.

Note that clustering is not a similarity metric *per se* (the topic of this section), but it may use various similarity metrics when computing clusters. It is included here because it can be used as a "cheap substitute". That is, when someone wants to find compounds similar to a known compound, you can show them the group (the cluster) to which the compound belongs. It allows you to pre-compute the clusters, spending lots of computational time up front, and then give answers very quickly.

Many cheminformatics databases have one or more similarity searches available.

12.5 Chemical Registration Systems

Chemical Registration is the "big brother" of cheminformatics.

A cheminformatics system is primarily devoted to recording chemical structure. Chemical Registration systems are additionally concerned with:

- Structural novelty - ensure that each compound is only registered once
- Structural normalization - ensure that structures with alternative representations (such as nitro groups, ferrocenes, and tautomers) are entered in a uniform way.
- Structure drawing - ensure that compounds are drawn in a uniform fashion, so that they can be quickly recognized "by eye".
- Maintaining relationships among related compounds. For example, all salt forms of a compound should be recognized as being related to one another, and compounds in different solvates are also related.
- Registering mixtures, formulations and alternative structures.
- Registering compounds the structure of which is unknown.
- Roles, responsibilities, security, and company workflow.
- Updates, amendments and corrections, and controlling propagation of changes (e.g. does changing a compound change a mixture containing that compound?)

The scope of Chemical Registration Systems is far beyond the goals of this brief introduction to cheminformatics. However, to illustrate just one of the points above, let's consider structural novelty. In real life, chemical structure can be very ambiguous. Imagine you have five bottles of a particular compound that has a stereo center:

1. The contents of the first bottle were carefully analyzed, and found to be a single stereoisomer.

2. The contents of the second bottle were carefully analyzed and found to contain a racemic mixture of the stereoisomers.
3. The stereoisomers of the third bottle are unknown. It may be pure, or have one predominant form, or be a racemic mixture.
4. The fourth bottle was obtained by running part of the contents of bottle #2 through a chromatographic separation. It is isotopically pure, but you don't know which stereoisomer.
5. The fifth bottle is the other fraction from the same separation of #4. It is also isotopically pure, but you don't know which stereoisomer, *but you know it's the opposite of #4*.

Which of these five bottles contain the same compound, and which are different? That is the essential task of a chemical registry system, which would consider all five to be different. After all, you probably have data about each bottle (that's why you have them), and you must be able to record it and not confuse it with the other bottles.

In this example above, consider what is known and not known:

Bottle	Known	Not Known
1	Everything	Nothing
2	Everything	Nothing
3	Compound is known	Stereochemistry
4	Compound and purity known, stereochemistry is opposite of #5	Specific stereochemistry
5	Compound and purity known, stereochemistry is opposite of #4	Specific stereochemistry

A cheminformatics system has no way to record the contents of the five bottles; it is only concerned with structure. By contrast, a chemical registration system can record both *what is known* as well as *what is not known*. This is the critical difference between the two.

Chapter 13

Stereochemistry

Open Babel stores stereochemistry as the relative arrangement of a set of atoms in space. For example, for a tetrahedral stereocenter, we store information like “looking from atom 2, atoms 4, 5 and 6 are arranged clockwise around atom 3”. This section describes how a user can work with or manipulate this information. This might be useful to invert a particular center, replace a substituent at a stereocenter, enumerate stereoisomers or determine the number of unspecified stereocenters.

Although Open Babel has data structures to support a variety of forms of stereochemistry, currently little use is made of any stereochemistry other than tetrahedral and cis/trans (and square planar to a certain degree).

We will look first of all at how stereochemistry information is stored, accessed, and modified. Then we describe how this information is deduced from the chemical structure. This chapter should be read in combination with the API documentation (see the Stereochemistry overview page found under “Modules”).

13.1 Accessing stereochemistry information

Each record of stereochemistry information around an atom or bond is stored as StereoData associated with the OBMol. First of all, let’s look at direct access to the StereoData. The following code counts the number of tetrahedral centers with specified stereochemistry, as well as the number of double bonds with specified cis/trans stereochemistry:

```
num_cistrans = 0
num_tetra = 0

mol = pybel.readstring("smi", "F/C=C/C[C@@H](Cl)Br")
m = mol.OBMol

for genericdata in m.GetAllData(ob.StereoData):
    stereodata = ob.toStereoBase(genericdata)
    stereotype = stereodata.GetType()

    if stereotype == ob.OBStereo.CisTrans:
        cistrans = ob.toCisTransStereo(stereodata)
        if cistrans.IsSpecified():
            num_cistrans += 1

    elif stereotype == ob.OBStereo.Tetrahedral:
        tetra = ob.toTetrahedralStereo(stereodata)
```

(continues on next page)

(continued from previous page)

```
        if tetra.IsSpecified():
            num_tetra += 1
```

Atom and Bond Ids

All of the stereo handling code uses Ids to reference atoms and bonds, rather than indices. An Open Babel atom has an index (`OBAtom::GetIdx()`) and an Id (`OBAtom::GetId()`). The former runs from 1 to the number of atoms. The latter can be anything (don't assume it's the Idx-1), but is unique within the molecule. If you delete an atom, the indices change, but the Ids do not. For this reason, all stereo is stored using Ids so that stereo information is not invalidated by changes to the molecule. When the stereochemistry description involves implicit hydrogens or lone pairs, the special atom Id `OBStereo::ImplicitRef` (numerically, `-2` or `4294967294` if accessed via the bindings) is used.

The code above is quite verbose, and requires iteration through all of the stereo data. To make it simpler to access stereo data for a particular atom or bond, a facade class OBStereoFacade can instead be used, which provides convenience functions for these operations:

```
num_cistrans = 0
num_tetra = 0

mol = pybel.readstring("smi", "F/C=C/C[C@@H](Cl)Br")
m = mol.OBMol

facade = ob.OBStereoFacade(m)

for atom in ob.OBMolAtomIter(m):
    mid = atom.GetId()
    if facade.HasTetrahedralStereo(mid):
        tetra = facade.GetTetrahedralStereo(mid)
        if tetra.IsSpecified():
            num_tetra += 1

for bond in ob.OBMolBondIter(m):
    mid = bond.GetId()
    if facade.HasCisTransStereo(mid):
        cistrans = facade.GetCisTransStereo(mid)
        if cistrans.IsSpecified():
            num_cistrans += 1
```

Note that every time you create a new OBStereoFacade, a certain amount of work is done building up the correspondance between atoms/bonds and stereo data. For this reason, a single OBStereoFacade should be created for a molecule and reused.

13.2 The Config() object

The description of the stereochemical configuration is accessed via a Config() object associated with each StereoData. The contents of this object will be different depending on the specific type of stereochemistry, e.g. `OBCisTransStereo::Config` (`OBCisTransConfig` from Python) records the begin and end Ids of the associated bond, the Ids of the attached atoms, the spatial relationship of those atoms, and whether stereo is specified.

Let's read the SMILES string `F[C@@](Cl)(Br)I` and access the stereo. When we read this SMILES string, the tetrahedral center will be the second atom, that with Idx 2.:

```
smi = "F[C@@](Cl)(Br)I"
mol = pybel.readstring("smi", smi).OBMol
secondatom = mol.GetAtom(2)
atomid = secondatom.GetId()

stereofacade = ob.OBStereoFacade(mol)
print("Does this atom have tet stereo info?", stereofacade.
↪HasTetrahedralStereo(atomid))
tetstereo = stereofacade.GetTetrahedralStereo(atomid)
config = tetstereo.GetConfig()
print("The stereocenter is at atom Id {}".format(config.center))
print("Is the configuration specified? {}".format("Yes" if config.specified else "No
↪"))
print("Looking from atom Id {0}, the atoms Ids {1} are arranged clockwise".
↪format(config.from_or_towards, config.refs))
```

Which prints:

```
Does this atom have tet stereo info? True
The stereocenter is at atom Id 1
Is the configuration specified? Yes
Looking from atom Id 0, the atoms Ids (2, 3, 4) are arranged clockwise
```

How do I know that I'm looking from atom Id 0, and that the atom Ids are arranged clockwise? From the documentation for `OBTetrahedralStereo::GetConfig`, which states that this is the default. You may be used to thinking "How are these atoms arranged looking from here?". With GetConfig(), you are instead making the request "Give me the atoms in clockwise order looking from here". It follows from this that you should never need to test the value of the winding, the direction, or the from/towards atom; you provide these, and their values will be whatever you provided. For example, you could instead ask for the anticlockwise arrangement of atoms looking *towards* the atom with Id 0:

```
configB = tetstereo.GetConfig(0, ob.OBStereo.AntiClockwise, ob.OBStereo.ViewTowards)
print("Looking towards atom Id {0}, the atoms Ids {1} are arranged anticlockwise".
↪format(config.from_or_towards, config.refs))
```

Which prints:

```
Looking towards atom Id 0, the atoms Ids (2, 3, 4) are arranged anticlockwise
```

To check whether two Configs represent the same stereo configuration, use the equality operator:

```
assert config == configB
```

It should be noted that the Config objects returned by GetConfig() are *copies* of the stereo configuration. That is, modifying them has no affect on the stereochemistry of the molecule (see the next section). As a result, it is straightforward to keep a copy of the stereo configuration, modify the molecule, and then check whether the modification has altered the stereochemistry using the equality operator of the Config.

13.3 Modifying the stereochemistry

We discuss below the interaction between 2D and 3D structural information and how stereochemistry is perceived. For now, let's avoid these issues by using a 0D structure and modifying its stereochemistry:

```
from openbabel import pybel
ob = pybel.ob
```

(continues on next page)

(continued from previous page)

```
mol = pybel.readstring("smi", "C[C@@H](Cl)F")
print(mol.write("smi", opt={"nonewline": True}))

# Invert the stereo
m = mol.OBMol
facade = ob.OBStereoFacade(m)
tetstereo = facade.GetTetrahedralStereo(m.GetAtom(2).GetId())
config = tetstereo.GetConfig()
config.winding = ob.OBStereo.AntiClockwise
tetstereo.SetConfig(config)
print(mol.write("smi", opt={"nonewline": True}))

config.specified = False
tetstereo.SetConfig(config)
print(mol.write("smi", opt={"nonewline": True}))
```

which prints:

```
C[C@@H](Cl)F
C[C@H](Cl)F
CC(Cl)F
```

How did I know that setting the relative arrangement to anti-clockwise would invert the stereo? Again, as described above, by default GetConfig() returns the atoms in clockwise order. Another way to invert the stereo would be to swap two of the refs, or to set the direction from ‘from’ to ‘towards’.

13.4 Stereo perception

Until now we have not mentioned where this stereo information came from; we have read a SMILES string and somehow the resulting molecule has stereo data associated with it.

Stereo perception is the identification of stereo centers from the molecule and its associated data, which may include 3D coordinates, stereobonds and existing stereo data. Passing an OBMol to the global function `PerceiveStereo` triggers stereo perception, and sets a flag marking stereo as perceived (`OBMol::SetChiralityPerceived(true)`). If, in the first place, stereo was already marked as perceived then stereo perception is not performed. Any operations that require stereo information should call PerceiveStereo before accessing stereo information.

Behind the scenes, the code for stereo perception is quite different depending on the dimensionality (`OBMol::GetDimension()`) of the molecule.

3D structures

Perhaps the most straightforward is when the structure has 3D coordinates. In this case, a symmetry analysis identifies stereogenic centers whose stereoconfigurations are then perceived from the coordinates. Some file formats such as the MOL file allow atoms and double bonds to be marked as have unspecified stereochemistry, and this information is applied to the detected stereocenters. For the specific case of the MOL file, the flag in the atom block that marks this is ignored by default (as required by the specification) but an option (`s`) is provided to read it:

```
$ obabel -:"I/C=C/C[C@@](Br)(Cl)F" --gen3d -omol | obabel -imol -osmi
I/C=C/C[C@@](Br)(Cl)F
$ obabel -:"IC=CC(Br)(Cl)F" --gen3d -omol | obabel -imol -osmi
```

(continues on next page)

(continued from previous page)

```
IC=CC[C@@](Br)(Cl)F
$ obabel -:"IC=CCC(Br)(Cl)F" --gen3d -omol | obabel -imol -as -osmi
IC=CCC(Br)(Cl)F
```

As just described, the flow of information is from the 3D coordinates to Open Babel's internal record of stereo centers, and this flow is triggered by calling stereo perception (which does nothing if the stereo is marked as already perceived). It follows from this that altering the coordinates *after* stereo perception (e.g. by reflecting through an axis, thereby inverting chirality) has no affect on the internal stereo data. If operations are performed on the molecule that require stereo is be reperceived, then `OBMol::SetChiralityPerceived(false)` should be called.

It should also be clear from the discussion above that changing the stereo data (e.g. using SetConfig() to invert a tetrahedral stereocenter) has no affect on the molecule's coordinates (though it may affect downstream processing, such as the information written to a SMILES string). If this is needed, the user will have to manipulate the coordinates themselves, or generate coordinates for the whole molecule using the associated library functions (e.g. the `--gen3d` operation).

2D structures

2D structures represent a depiction of a molecule, where stereochemistry is usually indicated by wedge or hash bonds. It is sometimes indicated by adopting particular conventions (e.g. the Fischer or Haworth projection of monosaccharides). It should be noted that Open Babel does not support any of these conventions, nor does it support the use of wedge or hash bonds for perspective drawing (e.g. where a thick bond is supported by two wedges). This may change in future, of course, but it's worth noting that Open Babel is not the only toolkit with these limitations and so what you think you are storing in your database may not be what the 'computer' thinks it is.

Stereo centers are identified based on a symmetry analysis, and their configuration inferred either from the geometry (for cis/trans bonds) or from bonds marked as wedge/hash (tetrahedral centers). File format readers record information about which bonds were marked as wedges or hashes and this can be accessed with OBBond:IsWedge/IsHash, where the Begin atom of the bond is considered the origin of the wedge/hash. Similar to the situation with 3D perception, changing a bond from a wedge to a hash (or vice versa) has no affect on the stereo objects once stereo has been perceived, but triggering reperception will regenerate the desired stereo data.

It should also be noted that the file writers regenerate the wedges or hashes from the stereo data at the point of writing; in other words, the particular location of the wedge/hash (or even whether it is present) may change on writing. This was done to ensure that the written structure accurately represents Open Babel's internal view of the molecule; passing wedges/hashes through unchanged may not represent this (consider the case where a wedge bond is attached to a tetrahedral center which cannot be a stereocenter).

0D structures

A SMILES string is sometimes referred to as describing a 0.5D structure, as it can describe the relative arrangement of atoms around stereocenters. The SMILES reader simply reads and records this information as stereo data, and then the molecule is marked as having stereo perceived (unless the `S` option is passed - see below).

Being able to skip the symmetry analysis associated with stereo perception means that SMILES strings can be read quickly - a useful feature if dealing with millions of molecules. However, if you wish to identify additional stereocenters whose stereo configuration is unspecified, or if the SMILES strings come from an untrusted source and stereo may have been incorrectly specified (e.g. on a tetrahedral center with two groups the same), then you may wish to trigger perception.

Without any additional information, stereo cannot be perceived from a structure that has neither 2D nor 3D coordinates. Triggering stereo perception on such a structure will generate stereo data if stereogenic centers are present, but their configuration will be marked as unspecified. However, where existing stereo data is present (e.g. after reading a SMILES string), that data will be retained if the stereocenter is identified by the perception routine as a true stereocenter. This can be illustrated using the S option to the SMILES reader, which tells it not to mark the stereo as perceived on reading; as a result, reperception will occur if triggered by a writer yielding different results in the case of an erroneously specified stereocenter:

```
$ obabel -:"F[C@@](F)(F)[C@@H](I)Br" -osmi
F[C@@](F)(F)[C@@H](I)Br
$ obabel -:"F[C@@](F)(F)[C@@H](I)Br" -aS -osmi
FC(F)(F)[C@@H](I)Br
```

13.5 Miscellaneous stereo functions in the API

- OBAtom::IsChiral - this is a convenience function that checks whether there is any tetrahedral stereo data associated with a particular atom. OBStereoFacade should be used in preference to this.

Chapter 14

Handling of aromaticity

The purpose of this section is to give an overview of how Open Babel handles aromaticity. Given that atoms can be aromatic, bonds can be aromatic, and that molecules have a flag for aromaticity perceived, it's important to understand how these all work together.

14.1 How is aromaticity information stored?

Like many other toolkits, Open Babel stores aromaticity information separate from bond order information. This means that there isn't a special bond order to indicate aromatic bond. Instead, aromaticity is stored as a flag on an atom as well as a flag on a bond. You can access and set this information using the following API functions:

- OBAtom::IsAromatic(), OBAtom::SetAromatic(), OBBond::UnsetAromatic()
- OBBond::IsAromatic(), OBBond::SetAromatic(), OBBond::UnsetAromatic()

There is a catch though, or rather a key point to note. OBMols have a flag to indicate whether aromaticity has been perceived. This is set via the following API functions:

- OBMol::SetAromaticPerceived(), OBMol::UnsetAromaticPerceived()

The value of this flag determines the behaviour of the OBAtom and OBBond IsAromatic() functions.

- If the flag is set, then IsAromatic() simply returns the corresponding value of the atom or bond flag.
- If unset, then IsAromatic() triggers aromaticity perception (from scratch), and then returns the value of the flag.

14.2 Perception of aromaticity

It's convenient to use the term "perception", but what we mean is to apply an aromaticity model. Currently Open Babel only has a single aromaticity model, which is close to the Daylight aromaticity model. An aromaticity model describes how many pi electrons are contributed by each atom; if this sums to 4n+2 within a cycle, then all atoms and bonds in that cycle will be marked as aromatic.

Applying a model involves creating an instance of OBAromaticTyper(), and calling AssignAromaticFlags() passing an OBMol as a parameter. This wipes any existing flags, sets the atom and bond flags according to the model, and marks the aromaticity as perceived.

If you wish (and know what you are doing), you can apply your own aromaticity model by setting various atoms and bonds as aromatic and then marking the molecule as having aromaticity perceived. Naturally, not all models will make sense chemically. Even more problematic is the situation where no Kekulé form exists that corresponds to the aromatic form. And finally, there is the philosophical question of the meaning of an aromatic atom without aromatic bonds, and vice versa.

14.3 SMILES reading and writing

Putting the pieces together, let's look at the interaction between SMILES reading/writing and the handling of aromaticity.

Writing SMILES

Unless Kekulé SMILES are requested (via the `k` output option), the SMILES writer will always write an aromatic SMILES string. IsAromatic() will be called on atoms and bonds to determine whether to use lowercase letters. As described earlier, this will trigger aromaticity perception according to the default model if the molecules is not marked as having its aromaticity perceived.

Reading SMILES

The situation when reading SMILES is a bit more involved. If the SMILES string contains lowercase characters and aromatic bonds, this information is used to mark atoms and bonds as aromatic. The molecule is then kekulized to assign bond orders to aromatic bonds. Next, unless the `a` option is supplied, the molecule is marked as having its aromaticity unperceived.

That last step might seem strange. Why, after going to the trouble of reading the aromaticity and using it to kekulize, do we then effectively ignore it?

The reason is simply this: when writing an aromatic SMILES, we usually want to use our own aromaticity model and not that present in the input SMILES string. Otherwise, SMILES strings for the same molecule from different sources (that may use different aromaticity models) would not yield the same canonical SMILES string.

Of course, if the SMILES string came from Open Babel in the first place, we are doing unnecessary work when we keep reapplying the same aromaticity model. In this case, you can speed things up by using the `a` option, so that the aromaticity information present in the input is retained. The following examples show this in action:

```
$ obabel -:cc -osmi
C=C
$ obabel -:cc -osmi -aa
cc
```

14.4 Effect of modifying the structure

Perhaps surprisingly, modifying the structure has no effect on the existing aromaticity flags; deleting an atom does not mark aromaticity as unperceived, nor indeed does any other change to the structure such as changing the atomic number of an atom or setting its charge; nor does the use of Begin/EndModify() affect the aromaticity flags. The only way to ensure that aromaticity is reperceived after modifying the structure is to explicitly mark it as unperceived.

The rationale for this is that an efficient toolkit should avoid unnecessary work. The toolkit does not know if a particular modification invalidates any aromaticity already perceived, or even if it did know, it cannot know whether the user actually wishes to invalidate it. It's up to the user to tell the toolkit. This places more responsibility in the hands of the user, but also more power.

To illustrate, let's consider what happens when the user reads benzene from the SMILES string `c1ccccc1`, and then modifies the structure by deleting an aromatic atom.

As this is an aromatic SMILES string, the SMILES reader will mark all atoms and bonds as aromatic. Next, the molecule itself is marked as not having aromaticity perceived (see previous section). After reading, we can trigger aromaticity perception by calling IsAromatic() on an atom; now, in addition to the atoms and bonds being marked as aromatic, the molecule itself will be marked as having aromaticity perceived.

If at this point we delete a carbon and write out a SMILES string, what will the result be? You may expect something like `[CH]=CC=C[CH]` (or `C=CC=CC` if we also adjust the hydrogen count on the neighbor atoms) but instead it will be `[cH]ccc[cH]` (or `ccccc` if hydrogens were adjusted).

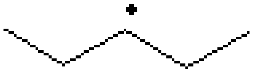
This follows from the discussion above - structural modifications have no effect on aromaticity flags. If instead the user wishes the SMILES writer to reperceive aromaticity, all that is necessary is to mark the molecule as not having aromaticity perceived, in which case the Kekulé form will instead be obtained.

Chapter 15

Radicals and SMILES extensions

15.1 The need for radicals and implicit hydrogen to coexist

Hydrogen deficient molecules, radicals, carbenes, etc., are not well catered for by chemical software aimed at pharmaceuticals. But radicals are important reaction intermediates in living systems as well as many other fields, such as polymers, paints, oils, combustion and atmospheric chemistry. The examples given here are small molecules, relevant to the last two applications.

Chemistry software to handle radicals is complicated by the common use of implicit hydrogen when describing molecules. How is the program to know when you type "O" whether you mean an oxygen atom or water? This ambiguity leads some to say that hydrogens should always be explicit in any chemical description. But this is not the way that most chemists work. A straight paraffinic chain from which a hydrogen had been abstracted might commonly be represented by something like: 

This uses implicit hydrogens and an explicit radical centre. But sometimes the hydrogens are explicit and the radical centre implicit, as when $CH_3$is used to represent the methyl radical.

15.2 How Open Babel does it

Open Babel accepts molecules with explicit or implicit hydrogens and can convert between the two. It will also handle radicals (and other hydrogen-deficient species) with implicit hydrogen by using internally a property of an atom, *_spinmultiplicity*, modelled on the RAD property in MDL MOL files and also used in CML. This can be regarded in the present context as a measure of the hydrogen deficiency of an atom. Its value is:

- 0 for normal atoms,
- 2 for radical (missing one hydrogen) and
- 1 or 3 for carbenes and nitrenes (missing two hydrogens).

It happens that for some doubly deficient species, like carbene CH_2 and oxygen atoms, the singlet and triplet species are fairly close in energy and both may be significant in certain applications such as combustion, atmospheric or preparative organic chemistry, so it is convenient that they can be described separately. There are of course an infinity of other electronic configurations of molecules but Open Babel has no special descriptors for them. However, even more hydrogen-deficient atoms are indicated by the highest possible value of spinmultiplicity (C atom has spin multiplicity of 5). (This extends MDL's RAD property which has a maximum value of 3.)

If the spin multiplicity of an atom is not input explicitly, it is set (in **:obapi:‘OBMol::AssignSpinMultiplicity() <OpenBabel::OBMol::AssignSpinMultiplicity>‘**) when the input format is MOL, SMI, CML or Therm. This routine is called after all the atoms and bonds of the molecule are known. It detects hydrogen deficiency in an atom and assigns spin multiplicity appropriately. But because hydrogen may be implicit it only does this for atoms which have at least one explicit hydrogen or on atoms which have had **:obapi:‘ForceNoH() <OpenBabel::OBAtom::ForceNoH>‘** called for them - which is effectively zero explicit hydrogens. The latter is used, for instance, when SMILES inputs `[O]` to ensure that it is seen as an oxygen atom (spin multiplicity=3) rather than water. Otherwise, atoms with no explicit hydrogen are assumed to have a spin multiplicity of 0, i.e with full complement of implicit hydrogens.

In deciding which atoms should be have spin multiplicity assigned, hydrogen atoms which have an isotope specification (D,T or even 1H) do not count. So SMILES `N[2H]` is NH_2D (spin multiplicity left at 0, so with a full content of implicit hydrogens), whereas `N[H]` is NH (spin multiplicity=3). A deuterated radical like NHD is represented by `[NH][2H]`.

15.3 In radicals either the hydrogen or the spin multiplicity can be implicit

Once the spin multiplicity has been set on an atom, the hydrogens can be implicit even if it is a radical. For instance, the following mol file, with explicit hydrogens, is one way of representing the ethyl radical:

```
ethyl radical
 OpenBabel04010617172D
Has explicit hydrogen and implicit spin multiplicity
  7  6  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 H   0  0  0  0  0
    0.0000    0.0000    0.0000 H   0  0  0  0  0
    0.0000    0.0000    0.0000 H   0  0  0  0  0
    0.0000    0.0000    0.0000 H   0  0  0  0  0
    0.0000    0.0000    0.0000 H   0  0  0  0  0
  1  2  1  0  0  0
  1  3  1  0  0  0
  1  4  1  0  0  0
  1  5  1  0  0  0
  2  6  1  0  0  0
  2  7  1  0  0  0
M  END
```

When read by Open Babel the spinmultiplicity is set to 2 on the C atom 2. If the hydrogens are made implicit, perhaps by the `-d` option, and the molecule output again, an alternative representation is produced:

```
ethyl radical
 OpenBabel04010617192D
Has explicit spin multiplicity and implicit hydrogen
  2  1  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
  1  2  1  0  0  0
M  RAD  1   2   2
M  END
```

15.4 SMILES extensions for radicals

Although radical structures can be represented in SMILES by specifying the hydrogens explicitly, e.g. `[CH3]` is the methyl radical, some chemists have apparently felt the need to devise non-standard extensions that represent the radical centre explicitly. Open Babel will recognize `C[O.]` as well as `C[O]` as the methoxy radical CH_3O during input, but the non-standard form is not supported in output.

By default, radical centres are output in explict hydrogen form, e.g. `C[CH2]` for the ethyl radical. All the atoms will be in explict H form, i.e. `[CH3][CH2]`, if **:obapi:‘AddHydrogens() <OpenBabel::OBMol::AddHydrogens>‘** or the `-h` option has been specified. The output is always standard SMILES, although other programs may not interpret radicals correctly.

Open Babel supports another SMILES extension for both input and output: the use of lower case atomic symbols to represent radical centres. (This is supported on the ACCORD Chemistry Control and maybe elsewhere.) So the ethyl radical is `Cc` and the methoxy radical is `Co`. This form is input transparently and can be output by using the `-xr` option “radicals lower case”. It is a useful shorthand in writing radicals, and in many cases is easier to read since the emphasis is on the radical centre rather than the number of hydrogens which is less chemically significant.

In addition, this extension interprets multiple lower case `c` without ring closure as a conjugated carbon chain, so that `cccc` is input as 1,3-butadiene. Lycopene (the red in tomatoes) is `Cc(C)cCCc(C)cccc(C)cccc(C)ccccc(C)cccc(C)cccc(C)CCcc(C)C` (without the stereochemical specifications). This conjugated chain form is not used on output - except in the standard SMILES aromatic form, `c1ccccc1` benzene.

It is interesting to note that the lower case extension actually improves the chemical representation in a few cases. The allyl radical C_3H_5 would be conventionally `[CH2]=[CH][CH2]` (in its explict H form), but could be represented as `ccc` with the extended syntax. The latter more accurately represents the symmetry of the molecule caused by delocalisation.

This extension is not as robust or as carefully considered as standard SMILES and should be used with restraint. A structure that uses `c` as a radical centre close to aromatic carbons can be confusing to read, and Open Babel’s SMILES parser can also be confused. For example, it recognizes `c1ccccc1c` as the benzyl radical, but it doesn’t like `c1cc(c)ccc1`. Radical centres should not be involved in ring closure: for cyclohexyl radical `C1cCCCC1` is ok, but `c1CCCCC1` is not.

15.5 Other Supported Extensions

Open Babel supports quadruple bonds `$`, e.g. `[Rh-](Cl)(Cl)(Cl)(Cl)$[Rh-](Cl)(Cl)(Cl)Cl` and aromatic `[te]`, e.g. `Cc1[te]ccc1`. In addition, ring closures up to 5 digits `%(N)` are supported, e.g. `C%(100)CC%(100)`.

Chapter 16

Contributing to Open Babel

16.1 Overview

Open Babel is developed using open, community-oriented development made possible by an active community – developers, testers, writers, implementers and most of all users. No matter which ‘er’ you happen to be, or how much time you can provide, you can make valuable contributions.

Not sure where to start? This section aims to give you some ideas.

Provide input

You can help us by:

- helping to answer questions on our mailing list
- suggesting new features or file formats
- reporting bugs

Spread the word

If you find Open Babel useful, there’s a chance that others will also. You can help us by:

- promoting and citing Open Babel in talks and publications
- writing blog posts about Open Babel
- helping with documentation and our website
- building your own software on Open Babel

To get started, just send an email to our mailing list.

Code a storm

As an open source project, Open Babel has a very open development process. This means that many contributors have helped with the project with a variety of help – some for long periods of time, and some with small, single changes. All types of assistance has been valuable to the growth of the project over the years.

New developers are always very welcome to OpenBabel so if you're interested, just send an email to the developer list (join here) about what you would like to work on, or else we can come up with some ideas for areas where you could contribute. Here are some possibilities:

- Implement the latest algorithms described in the literature
- Add a new file format (see *How to add a new file format*)
- Perform 'software archaeology' (see *Software Archaeology*)
- Fix some bugs
- Add a requested feature
- Implement a feature from our roadmap

16.2 Developing Open Babel

Due to its open nature of its development, Open Babel contains code contributed by a wide variety of developers (see *Thanks*). This section describes some general guidelines and "best practices" for code developers.

16.2.1 Developer Resources

For new and existing developers here are some useful resources:

- GitHub project page
- Development version API documentation
- Development version Sphinx documentation

16.2.2 Working with the Development Code

To download and update the latest version of the Open Babel source code, you need Git. Git is the name of the project used to maintain the Open Babel version control repository. There are many clients for Git, including command-line and GUI applications.

Keeping up to date with Git

(1) Check out the latest development version:

```
git clone https://github.com/openbabel/openbabel.git openbabel-dev
```

This creates a directory called `openbabel-dev`, which contains the latest source code from Open Babel.

(2) Configure and compile this using CMake (see *Compiling Open Babel*).

(3) After some time passes, and you want the latest bug fixes or new features, you may want to update your source code. To do this, go into the `openbabel-dev` directory you created above, and type:

```
git pull -u
```

(4) Do step (2) again.

(5) If, after updating, the compilation fails please report it to the Open Babel mailing list. In the meanwhile, if you want to go back to a particular revision (that is, if you don't want to use the latest one), just use `git log` to find the checksum of the current revision, and update to an earlier revision using this:

$ git log … commit 1c2916cc5e6ed31a23291524b08291c904506c3f Author: Noel O'Boyle <baoilleach@gmail.com> Date: Mon Apr 30 07:33:17 2018 +0100

$ git checkout 1c2916cc5

16.2.3 Modular design of code base

Since version 2.0, Open Babel has had a modular structure. Particularly for the use of Open Babel as a chemical file format converter, it aims to:

- separate the chemistry, the conversion process and the user interfaces, reducing, as far as possible, the dependency of one on another.
- put all the code for each chemical format in one place (usually a single cpp file) and make the addition of new formats simple.
- allow the format conversion of not just molecules, but also any other chemical objects, such as reactions.

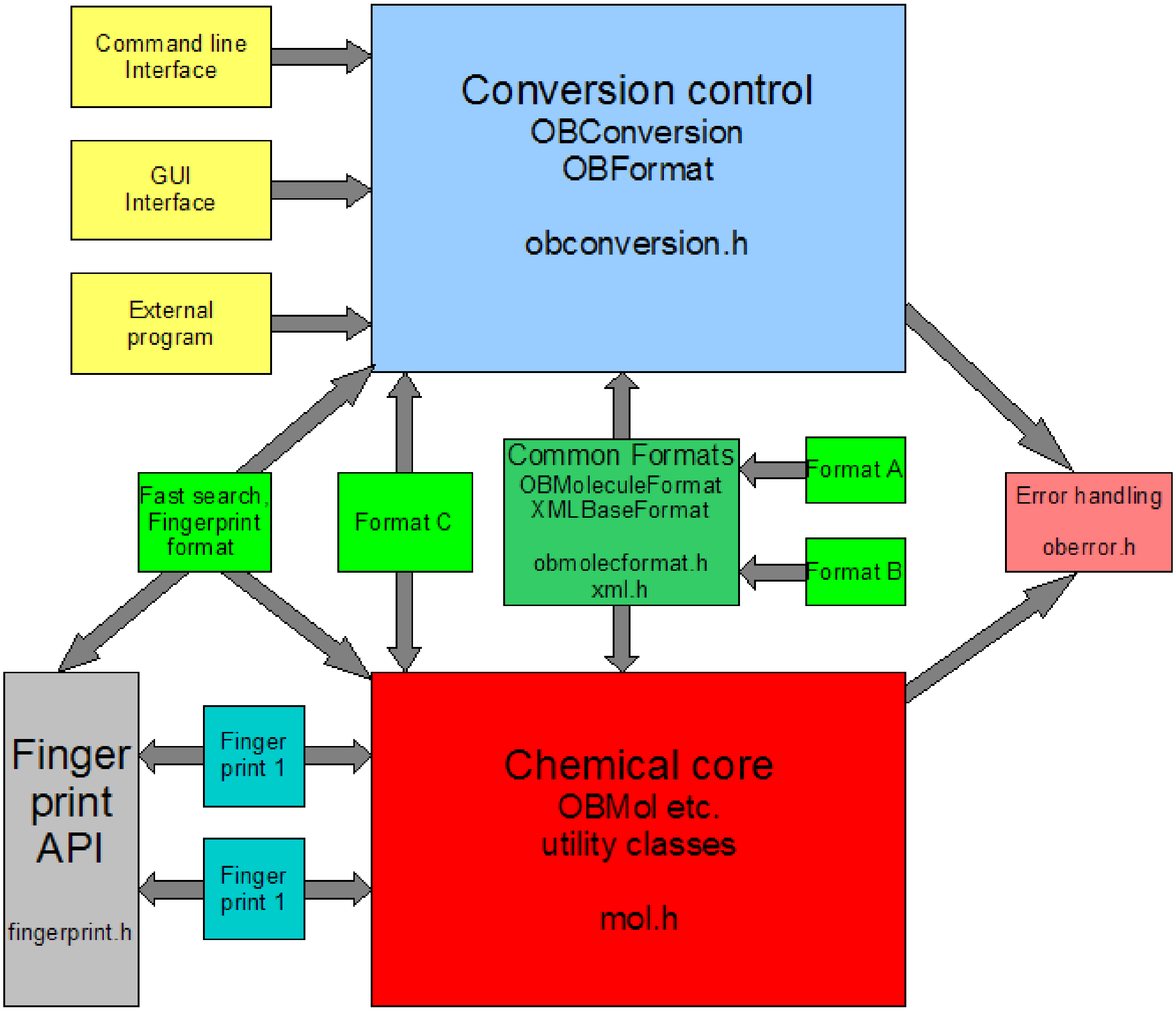


Fig. 1: The structure of the Open Babel codebase broken down into modules

The separate parts of the OpenBabel program are:

- The **Chemical** core, which contains OBMol etc. and has all the chemical structure description and manipulation. This bit is the heart of the application and its API can be used as a chemical toolbox. It has no input/output capabilities.
- The **Formats**, which read and write to files of different types. These classes are derived from a common base class, OBFormat, which is in the Conversion Control module. They also make use of the chemical routines in the Chemical Core module. Each format file contains a global object of the format class. When the format is loaded the class constructor registers the presence of the class with OBConversion. This means the formats are plugins - new formats can be added without changing any framework code.
- **Common Formats** include OBMoleculeFormats and XMLBaseFormat from which most other formats (like Format A and Format B in the diagram) are derived. Independent formats like Format C are also possible.
- The **Conversion** control, which also keeps track of the available formats, the conversion options and the input and output streams. It can be compiled without reference to any other parts of the program. In particular, it knows nothing of the Chemical core: mol.h is not included.
- The **User interface**, which may be a command line (in main.cpp), a Graphical User Interface(GUI), especially suited to Windows users and novices, or may be part of another program which uses OpenBabel's input and output facilities. This depends only on the Conversion control module (obconversion.h is included), but not on the Chemical core or on any of the Formats.
- The **Fingerprint API**, as well as being usable in external programs, is employed by the fastsearch and fingerprint formats.
- The **Fingerprints**, which are bit arrays which describe an object and which facilitate fast searching. They are also built as plugins, registering themselves with their base class OBFingerprint which is in the Fingerprint API.
- The **Error handling** can be used throughout the program to log and display errors and warnings (see below).

It is possible to build each box in the diagram as a separate DLL or shared library and the restricted dependencies can help to limit the amount of recompilation. For the formats or the fingerprints built in this way it may be possible to use only those whose DLL or so files are present when the program starts. Several formats or fingerprints may be present in a single dynamic library.

Alternatively, and most commonly, the same source code can be built into a single executable. The restricted dependencies still provide easier program maintenance.

This segregation means that a module can directly call code only in other modules connected to it by forward arrows. So some discipline is needed when adding new code, and sometimes non-obvious work-arounds are necessary. For instance, since the user interface doesn't know about the Chemical Core, if it were necessary to set any parameters in it, then this would have to be done through a pseudo format OBAPIInterface.

Sometimes one format needs to use code from another format, for example, rxnformat needs to read mol files with code from mdlformat. The calling format should not use the code directly but should do it through a OBConversion object configured with the appropriate helper format.

The objects passed between the modules in the diagram above are polymorphic **:obapi:'OBBase'** pointers. This means that the conversion framework can be used by any object derived from OBBase (which essentially means anything - chemical or not). Most commonly these refer to OBMol objects, less commonly to OBReaction objects, but could be extended to anything else without needing to change any existing code.

16.2.4 Error Handling and Warnings

The general philosophy of the Open Babel project is to attempt to gracefully recover from error conditions. Depending on the severity of the error, a message may or may not be sent to the user – users can filter out developer debugging messages and minor errors, but should be notified of significant problems.

Errors and warnings in Open Babel are handled internally by a flexible system motivated by a few factors:

- End users often do not wish to be deluged by debugging or other messages during operation.
- Other developers may wish to redirect or filter error/warning output (e.g., in a GUI).
- The operation of Open Babel should be open to developers and users alike to monitor an "audit trail" of operations on files and molecules, and debug the program and library itself when the need arises.

Multiple error/warning levels exist and should be used by code. These are defined in the **:obapi:'obMessageLevel'** enum as follows:

- `obError` – for critical errors (e.g., cannot read a file)
- `obWarning` – for non-critical problems (e.g., molecule appears empty)
- `obInfo` – for informative messages (e.g., file is a non-standard format)
- `obAuditMsg` – for messages auditing methods which destroy or perceive molecular data (e.g., kekulization, atom typing, etc.)
- `obDebug` – for messages only useful for debugging purposes

The default filter level is set to `obWarning`, which means that users are told of critical errors, but not non-standard formatting of input files.

A global error handler **:obapi:'obErrorLog'** (an instance of **:obapi:'OBMessageHandler'**) is defined and should be used as shown in the API documentation for the **:obapi:'OBMessageHandler'** class.

16.2.5 Lazy Evaluation

The **:obapi:'OBMol::BeginModify() <OpenBabel::OBMol::BeginModify>'** and **:obapi:'OBMol::EndModify() <OpenBabel::OBMol::EndModify>'** calls are part of Open Babel's lazy evaluation mechanism.

In some cases, code may desire to make a large number of changes to an OBMol object at once. Ideally, this should all happen without triggering unintended perception routines. Therefore, the `BeginModify()` call marks the beginning of such code, and `EndModify()` triggers any needed updates of lazy evaluation methods.

For example:

```
mol.BeginModify();
double x,y,z;
OBAtom *atom;
vector<string> vs;

for (i = 1; i <= natoms; i ++)
{
    if (!ifs.getline(buffer,BUFF_SIZE))
        return(false);
    tokenize(vs,buffer);
    if (vs.size() != 4)
        return(false);

    atom = mol.NewAtom();
    x = atof((char*)vs[1].c_str());
    y = atof((char*)vs[2].c_str());
    z = atof((char*)vs[3].c_str());

    atom->SetVector(x,y,z); //set coordinates
    atom->SetAtomicNum(atoi(vs[0].c_str())); // set atomic number
}
mol.ConnectTheDots();
```

(continues on next page)

(continued from previous page)

```
mol.PerceiveBondOrders();
mol.EndModify();
```

This code reads in a list of atoms with XYZ coordinates and the atomic number in the first column (`vs[0]`). Since hundreds or thousands of atoms could be added to a molecule, followed by creating bonds, the code is enclosed in a `BeginModify()`/`EndModify()` pair.

16.3 Documentation

Documenting Open Babel is an important and ongoing task. As an open source project, code must be documented, both for other developers to use the API and for others to follow your code. This includes clear documentation on the interfaces of particular classes and methods (that is, the API documentation) but also tutorials and examples of using the Open Babel library to accomplish clear tasks.

Beyond the documentation described above, as an open-source project involving many, many contributors, the internal code should be clearly commented and easy to read (in English, preferably, since this is the common language of developers on the project).

16.3.1 Adding New Code

The golden rule is **write the documentation, then code to the specs**.

You should never, ever start writing code unless you've specified, clearly and exactly, what your code will do. This makes life easier for you (i.e., you know exactly what the code should do), and for others reading your code.

This mantra also facilitates writing tests (see *Adding a new test*).

16.3.2 Modifying Old Code

When modifying old code, please take a little time to improve the documentation of the function.

Even an "obvious" function must be documented, if for no other reason than to say, "This function does what you think, and has no side effects."

Take **:obapi:'OBAtom::SetAtomicNum() <OpenBabel::OBAtom::SetAtomicNum>'** - should be "obvious", right? Wrong.

- Does it affect the charge?
- The spin multiplicity?
- The implicit valence?
- The hybridization?
- What happens if I do SetHybridization(3) and then SetAtomicNum(1)?
- Does the molecule have to be in the modify state?
- If the molecule is not in the modify state, is it put into the modify state by SetAtomicNum()?
- Does SetAtomicNum() cause a recomputation of aromaticity?

16.3.3 User documentation and tutorials

There's no point spending time adding new features to Open Babel unless you describe how to use them and give examples. The best place to do this is in the user documentation… which you're reading right now.

This documentation is automatically generated from text files in a simple markup language (*reStructuredText*) using the Sphinx documentation system. This allows us to generate web pages, PDF files, and even ePub eBooks all from the same source (which is currently maintained at BitBucket).

If you notice any errors or feel like adding a section, please let us know at openbabel-devel.

16.4 Adding a new test

Tests allow us to maintain code quality, ensure that code is working, prevent regressions, and facilitate refactoring. Personally, I find that there is no better motivation for writing tests than knowing that that bug I fixed will stay fixed, and that feature I implemented will not be broken by others. As an open source developer, I never have enough time; tests ensure that what time I have is not wasted.

We can divide the existing tests into three classes, based on how they test the Open Babel codebase:

1. Tests written in C++ that test the public API
2. Tests written in Python that use the SWIG bindings to test the public API
3. Tests written in Python that use the command-line executables for testing

Which type of test should you write? It doesn't really matter - it's more important that you write *some* type of test. Personally, I can more quickly test more if I write the test in Python, so generally I write and check-in tests of type (2) above; when I need to run a testcase in a debugger, I write a short test of type (1) so that I can step through and see what's happening.

16.4.1 Running tests

To begin with, we need to configure CMake to enable tests: `-DENABLE_TESTS=ON`. This adds the `make test` target and builds the C++ tests. For tests of type 3 (above), you will also need to enable the Python bindings: `-DPYTHON_BINDINGS=ON -DRUN_SWIG=ON`. Some tests are dependent on optional dependencies; if you don't build with support for these, then the corresponding tests will not be run.

To actually run the tests, you can run the entire test suite in one go or run individual tests. To run the entire suite, use `make test` or `ctest` (note that you can use the `-j` option to speed up ctest). The ctest command also allows a single test or a list of tests to be specified, and in combination with `-VV` (verbose) may be useful to run an individual test. However, I find it more useful to run individual tests directly. Here is an example of how to run an individual test for each of the three types discussed earlier:

1. `test_runner regressionstest 1`

   This will run test number 1 in `regressionstest.cpp`. Nothing will happen… unless the test fails. (test_runner is a testing harness generated by CMake.)

2. `python test\testbindings.py TestSuite.testAsterisk`

   This will run the testAsterisk test in `testbindings.py`. This will write out a single dot, and some summary information.

3. `python test\testbabel.py testOBabel.testSMItoInChI`

   This will run the testSMItoInChI test in `testbabel.py`.

The next few sections describe adding a new test of types 1 to 3. The same test will be added, a test to ensure that the molecular weight of ethanol is reported as 46.07.

16.4.2 Test using C++

The easiest place to add new tests is into `test/regressionstest.cpp`. Look at the switch statement at the end of the file and pick a number for the test. Let's say 260. Add the following:

```
case 260:
  test_Ethanol_MolWt();
  break;
```

Now add the value of 260 to `test/CMakeLists.txt` so that it will be run as part of the testsuite.:

```
set (regressions_parts 1 221 222 223 224 225 226 227 260)
```

Now let's add the actual test somewhere near the top of the file:

```
void test_Ethanol_MolWt()
{
  OBMol mol;
  OBConversion conv;
  OB_REQUIRE(conv.SetInFormat("smi"));
  conv.ReadString(&mol, "CCO");
  double molwt = mol.GetMolWt();
  OB_ASSERT(molwt - 46.07 < 0.01);
}
```

The various assert methods are listed in `obtest.h` and are as follows:

- **OB_REQUIRE(exp)** - This must evaluate to `true` or else the test will be marked as failing and will terminate. This is useful for conditions that *must* be true or else the remaining tests cannot be run (e.g. was the necessary OBFormat found?).
- **OB_ASSERT(exp)** - This must evaluate to `true` or else the test will be marked as failing. In contrast to OB_REQUIRE, the test does not terminate in this case, but continues to run. This feature can be useful because it lets you know (based on the output) how many and which OB_ASSERT statements failed.
- **OB_COMPARE(expA, expB)** - Expressions A and B must be equal or else the test fails (but does not terminate).

It is often useful to write a test that uses a checked-in testfile. Let's do this for our example testcase. If you place a file `ethanol.smi` into `test/files`, then the following will read it using a convenience function provided by `obtest.h`.:

```
void test_Ethanol_MolWt()
{
  OBMolPtr mol = OBTestUtil::ReadFile("ethanol.smi")
  double molwt = mol.GetMolWt();
  OB_ASSERT(molwt - 46.07 < 0.01);
}
```

As well as `ReadFile` (which is convenient for single molecules), the `OBTestUtil` struct provides `GetFilename` which will return the full path to the testfile, if you wish to open it yourself.

16.4.3 Test using a command-line executable

At the command-line we can calculate the molecular weight of ethanol as shown below. We are going to do something similar using the Python test framework:

```
> obabel -:CCO --append MW -otxt
46.0684
```

Open test/testbabel.py in an editor. I have grouped tests related to the obabel executable into a class testOBabel, so let's add a new test there. Somewhere in that class (for example, at the end), add a function such as the following (note: it must begin with the word "test"):

```
def testMolWtEthanol(self):
    """Check the molecular weight of ethanol"""
    self.canFindExecutable("obabel")
    answers = [
            ("CCO", 46.07),
            ("[H]", 1.01),
            ("[2H]", 2.01),
            ]
    for smi, molwt in answers:
        output, error = run_exec('obabel -:%s --append mw -otxt' % smi)
        my_molwt = round(float(output), 2)
        self.assertEqual(my_molwt, molwt)
```

We provide a few convenience functions to help write these tests. The most important of these is run_exec(command) which runs the commandline executable returns a tuple of stdout and stderr. Behind the scenes, it adds the full path to the named executable. In the example above, run_exec(stdin, command) took a single argument; the next example will show its use with two arguments - the additional argument is a string which is treated as stdin, and piped through the executable.

In the previous example, each SMILES string was passed in one-at-a-time. However, it is more efficient to do them all in one go as in the following example:

```
def testMolWtEthanol(self):
    """Check the molecular weight of ethanol"""
    self.canFindExecutable("obabel")
    smifile = """CCO
    [H]
    [2H]
    """
    answers = [46.07, 1.01, 2.01]
    output, error = run_exec(smifile, 'obabel -ismi --append mw -otxt')
    for ans, my_ans in zip(answers, output.split("\n")):
        self.assertEqual(ans, round(float(my_ans), 2))
```

To use a testfile placed in test/files, the getTestFile() member function is provided:

```
def testMolWtEthanol(self):
    """Check the molecular weight of ethanol"""
    self.canFindExecutable("obabel")
    answers = [46.07, 1.01, 2.01]
    smifile = self.getTestFile("ethanol.smi")
    output, error = run_exec('obabel %s --append mw -otxt' % smifile)
    for ans, my_ans in zip(answers, output.split("\n")):
        self.assertEqual(ans, round(float(my_ans), 2))
```

The full list of provided convenience functions is:

- **run_exec(command)**, **run_exec(stdin, command)** - see above
- **BaseTest.getTestFile(filename)** - returns the full path to a testfile
- **BaseTest.canFindExecutable(executable)** - checks whether the executable exists in the expected location
- **BaseTest.assertConverted(stderr, N)** - An assert statement that takes the stderr from run_exec and will check whether the number of molecules reported as converted matches N

16.4.4 Test the API using Python

The easiest place to add new tests is into `test/testbindings.py`. Classes are used to organise the tests, but for a single ‘miscellaneous’ test a good place is the TestSuite class. Somewhere in that class add the following function:

```
def testMolWtEthanol(self):
    """Check the molecular weight of ethanol"""
    answers = [
            ("CCO", 46.07),
            ("[H]", 1.01),
            ("[2H]", 2.01),
            ]
    for smi, molwt in answers:
        my_molwt = round(pybel.readstring("smi", smi).molwt, 2)
        self.assertEqual(my_molwt, molwt)
```

The variable `here` is defined in `testbindings.py` and may be used find the path to testfiles. For example, given the `test/ethanol.smi`, the following may be used to read it:

```
def testMolWtEthanol(self):
    """Check the molecular weight of ethanol"""
    answers = [46.07, 1.01, 2.01]
    testfile = os.path.join(here, "test", "ethanol.smi")
    for mol, answer in zip(pybel.readfile("smi", testfile), answers):
        my_molwt = round(mol.molwt, 2)
        self.assertEqual(my_molwt, molwt)
```

The tests use the standard unittest framework. One thing to note, which is not obvious, is how to test for exceptions. A typical case is checking that a dodgy SMILES is rejected on reading; in this instance, `Pybel.readstring()` will raise an IOError. To assert that this is the case, rather than use try/except, the following syntax is required:

```
self.assertRaises(IOError, pybel.readstring, "smi", "~*&*($")
```

If you have multiple tests to add on a single ‘topic’, you will probably want to add your own class either into `testbindings.py` or a new Python file. Note that if you create a new Python file, it should start with the word `test` and you will need to add the rest of the name to the `pybindtest` list in `test/CMakeLists.txt`.

16.4.5 Some final comments

Some thoughts on the topic of the perfect test:

- When adding a regression test for a bug fix, the test should fail without the fix, but pass afterwards.
- When adding a test for a new feature, the test should provide complete coverage for all code paths.
- Test not just perfect input, but all sorts of dodgy input like molecules with no atoms, empty strings, and so forth.
- Don’t be afraid to add tests for things which you already (think you) know will pass; such tests may surprise you, and even if they don’t they will prevent regressions.

Potential problems/gotchas:

- Why isn't your Python test being run? Test functions name must begin with the word `test`.
- If your new test passes first time, check that it is actually running correctly, by changing your asserts so that they should fail.
- The C++ tests will be marked as failing if the test writes any of the following to stdout: `ERROR`, `FAIL`, `Test failed`. This is actually how the assert methods work.
- It's best to avoid writing to disk, and instead write to a variable or stdout and capture it (as in the examples above).

16.5 Software Archaeology

In any large software project, some parts of the code are revised and kept up-to-date more than others.

Conversely, some parts of the code begin to fall behind – the code may be poorly tested, poorly documented, and not always up to best practices.

With that in mind, the following sections describe the important task of software archeology – diving in to older parts of code and bringing them up to date. Whenever editing a file, please keep these in mind.

16.5.1 Documentation and Code Readability

- Add clear documentation for every public function (see *Documentation*).
- Add clear comments on the internal operation of functions so that anyone can read through the code quickly.
  - If you're not sure what a function does, e-mail the openbabel-devel list and it can be worked out.
- Mark functions which should be publicly visible and functions which are only useful internally. Many methods are not particularly useful except inside the library itself.
- Improve code indentation
  - It seems like a minor point, but the format of your code is important. As open source software, your code is read by many, many people.
  - Different contributions have often had different indentation styles. Simply making the code indentation consistent across an entire file makes the code easier to read.
  - The current accepted scheme for Open Babel is a default indent of two spaces, and use of spaces instead of tabs.
  - For tips on changing your editor to use this indentation style, please e-mail the openbabel-devel list.
- Delete code which is commented out. The SVN version control system maintains history, so if we need it later, we can go back and get that code. Dead code like this simply makes it harder to read the important code!
- Marking areas of code which use **:obapi:‘OBAtom::GetIdx() <OpenBabel::OBAtom::GetIdx>‘** or other accesses to atom indexes, which may break when atom indexing changes.

16.5.2 Code Maintenance

- Minimize `#if`/`#endif` conditional compilation. Some is required for portability, but these should be minimized where possible. If there seems to be some magic #define which accesses parts of the file, it's probably dead code. As above, dead code makes it harder to maintain and read everything else.

- Removing calls to cout, cerr, STDOUT, perror etc. These should use the global error reporting code.
- Minimize warnings from compilers (e.g., GCC flags `-Wextra -Wall`). Sometimes these are innocuous, but it's usually better to fix the problems before they become bugs.
- Use static code analysis tools to find potential bugs in the code and remove them.
- Insure proper use of atom and bond iterators, e.g., `FOR_ATOMS_OF_MOL` rather than atom or bond index access, which will break if indexing changes.

Patches and contributions towards any of these tasks will be greatly appreciated.

Chapter 17

Adding plugins

Open Babel uses a plugin architecture for file formats, ‘operations’, charge models, forcefields, fingerprints and descriptors. The general idea behind plugins is described on Wikipedia. When you start an application that uses the Open Babel library, it searches for available plugins and loads them. This means, for example, that plugins could be distributed separately to the Open Babel distribution.

In fact, even the plugin types are themselves plugins; this makes it easy to add new categories of plugin. The different types of plugins can be listed using:

```
C:\>babel -L
charges
descriptors
fingerprints
forcefields
formats
loaders
ops
```

To list the plugins of a particular type, for example, charge models, just specify the plugin type:

```
C:\>babel -L charges
gasteiger    Assign Gasteiger-Marsili sigma partial charges
mmff94       Assign MMFF94 partial charges
qeq    Assign QEq (charge equilibration) partial charges (Rappe and Goddard, 199
1)
qtpie    Assign QTPIE (charge transfer, polarization and equilibration) partial
charges (Chen and Martinez, 2007)
```

To add a new plugin of any type, the general method is very simple:

1. Make a copy of an existing plugin .cpp file
2. Edit it so that it does what you want
3. Add the name of the .cpp file to the appropriate `CMakeLists.txt`.

The following sections describe in depth how to add support for a new file format or operation to Open Babel. Remember that if you do add a new plugin, please contribute the code back to the Open Babel project.

17.1 How to add a new file format

Adding support for a new file format is a relatively easy process, particularly with Open Babel 2.3 and later. Here are several important steps to remember when developing a format translator:

1. Create a file for your format in `src/formats/` or `src/formats/xml/` (for XML-based formats). Ideally, this file is self-contained although several formats modules are compiled across multiple source code files.

2. Add the name of the new .cpp file to an appropriate place in `src/formats/CMakeLists.txt`. It will now be compiled as part of the build process.

3. Take a look at other file format code, particularly `exampleformat.cpp`, which contains a heavily-annotated description of writing a new format. XML formats need to take a different approach; see the code in `xcmlformat.cpp` or `pubchemformat.cpp`.

4. When reading in molecules (and thus performing a lot of molecular modifications) call **:obapi:‘OBMol::BeginModify() <OpenBabel::OBMol::BeginModify>‘** at the beginning and **:obapi:‘OBMol::EndModify() <OpenBabel::OBMol::EndModify>‘** at the end. This will ensure that perception routines do not run while you read in a molecule and are reset after your code finishes (see *Lazy Evaluation*).

5. Currently, lazy perception does not include connectivity and bond order assignment. If your format does not include bonds, make sure to call **:obapi:‘OBMol::ConnectTheDots() <OpenBabel::OBMol::ConnectTheDots>‘** and **:obapi:‘OBMol::PerceiveBondOrders() <OpenBabel::OBMol::PerceiveBondOrders>‘** after **:obapi:‘OBMol::EndModify() <OpenBabel::OBMol::EndModify>‘** to ensure bonds are assigned.

6. Consider various input and output options that users can set from the command-line or GUI. For example, many quantum mechanics formats (as well as other formats which do not recognize bonds) offer the following options:

   `-as` Call only **:obapi:‘OBMol::ConnectTheDots() <OpenBabel::OBMol::ConnectTheDots>‘** (single bonds only)

   `-ab` No bond perception

7. Make sure to use generic data classes like **:obapi:‘OBUnitCell‘** and others as appropriate. If your format stores any sort of common data types, consider adding a subclass of **:obapi:‘OBGenericData‘** for use by other formats and user code.

8. Please make sure to add several example files to the test set repository. Ideally, these should work several areas of your import code – in the end, the more robust the test set, the more stable and useful Open Babel will be. The test files should include at least one example of a correct file and one example of an invalid file (i.e., something which will properly be ignored and not crash **`babel`**).

9. Make sure to document your format using the string returned by `Description().` At the minimum this should include a description of all options, along with examples. However, the more information you add (e.g. unimplemented features, applications of the format, and so forth) the more confident users will be in using it.

10. That’s it! Contact the openbabel-discuss mailing list with any questions, comments, or to contribute your new format code.

17.2 Adding new operations and options

The **`babel`** command line has the form:

```
babel inputfile [outputfile] [options]
```

There are several types of options:

Options that control the conversion process For example `-i`, `-o` and `-m`

Options specific to particular input or output formats These are specified with the `-a` and `-x` prefixes

General options These usually operate on a molecule after it has been read by the input format and before it has been written by the output format.

The ones of interest here are the general options. These can be single letter options like `-c` (which centers coordinates), or multi-character options like `--separate` (which makes separate molecules from disconnected fragments). The ones mentioned are hardwired into the code, but it is possible to define new options that work in a similar way. This is done using the **:obapi:‘OBOp‘** class.

17.2.1 The OBOp class

The name **:obapi:‘OBOp‘** is intended to imply an operation as well as an option. This is a plugin class, which means that new ops are easily added without a need to alter any existing code.

The ops that are installed can be found using:

```
babel -L ops
```

or in the plugins menu item in the GUI. An example is the `--gen3D` option, which adds 3D coordinates to a molecule:

```
1   class OpGen3D : public OBOp
2   {
3   public:
4     OpGen3D(const char* ID) : OBOp(ID, false){};
5     const char* Description(){ return "Generate 3D coordinates"; }
6
7     virtual bool WorksWith(OBBase* pOb)const
8       { return dynamic_cast<OBMol*>(pOb)!=NULL; }
9     virtual bool Do(OBBase* pOb, OpMap* pmap, const char* OptionText);
10  };
11
12  OpGen3D theOpGen3D("gen3D");
13
14  bool OpGen3D::Do(OBBase* pOb, OpMap* pmap, const char* OptionText)
15  {
16    OBMol* pmol = dynamic_cast<OBMol*>(pOb);
17    if(!pmol)
18      return false;
19
20    OBBuilder builder;
21    builder.Build(*pmol);
22    pmol->SetDimension(3);
23
24    return true;
25  }
```

The real work is done in the *Do* function, but there is a bit of boilerplate code that is necessary.

Line **4**: The constructor calls the base class constructor, which registers the class with the system. There could be additional parameters on the constructor if necessary, provided the base constructor is called in this way. (The `false` parameter value is to do with setting a default instance which is not relevant here.)

Line **5**: It is necessary to provide a description. The first line is used as a caption for the GUI checkbox. Subsequent lines are shown when listed with the verbose option.

Line **7**: *WorksWith()* identifies the type of object. Usually this is a molecule (*OBMol*) and the line is used as shown. The function is used by the GUI to display the option only when it is relevant.

> The *OBOp* base class doesn't know about *OBMol* or *OBConversion* and so it can be used with any kind of object derived from *OBBase* (essentially anything). Although this means that the dependencies between one bit of the program and another are reduced, it does lead to some compromises, such as having to code *WorksWith()* explicitly rather than as a base class default.

Line **12**: This is a global instance which defines the Id of the class. This is the option name used on the command line, preceded by `--`.

Line **14**: The *Do()* function carries out the operation on the target object. It should normally return `true`. Returning `false` prevents the molecule being sent to the output format. Although this means that it is possible to use an *OBOp* class as a filter, it is better to do this using the `--filter` option.

Any other general options specified on the command line (or the GUI) can be accessed by calling *find* on the parameter *pmap*. For example, to determine whether the `-c` option was also specified:

```
OpMap::const_iterator iter = pmap->find("c");
if(iter!=pmap->end())
  do something;
```

17.3 How to add a new descriptor

[Some text here]

17.3.1 Add a new group contribution descriptor

Group contribution descriptors are a common type of molecular descriptor whose value is a sum of contributions from substructures of the molecule. Such a descriptor can easily be added to Open Babel without the need to recompile the code. All you need is a set of SMARTS strings for each group, and their corresponding contributions to the descriptor value.

The following example shows how to add a new descriptor, *hellohalo*, whose value increments by 1, 2, 3 or 4 for each F, Cl, Br, and I (respectively) in the molecule.

1. Create a working directory, for example `C:\Work`.
2. Copy the plugin definition file, `plugindefines.txt` to the working directory. This file can be found in the Open Babel data directory (typically in `/usr/share/openbabel` on Linux systems, or `C:\Users\username\AppDataRoaming\OpenBabel-2.3.2\data` on Windows).
3. For the *hellohalo* descriptor, add the following to the end of `plugindefines.txt` (make sure to include a blank line between it and other descriptor definitions):

   ```
   OBGroupContrib
   hellohalo              # name of descriptor
   hellohalo_smarts.txt   # data file
   Count up the number of halogens (sort of)\n     # brief description
   This descriptor is not correlated with any\n    # longer description
   known property, living or dead.
   ```

4. Now create a file `hellohalo_smarts.txt`, again in the working directory, containing the following SMARTS definitions and contribution values:

```
# These are the SMARTS strings and contribution values
# for the 'hellohalo' group contribution descriptor.
;heavy
F     1   # This is for fluorines
Cl    2   # And this is for chlorines
Br    3   # Etc.
I     4   # Ditto
```

That's it!

Now let's test it. Open a command prompt, and change directory to the working directory. We can find information on the new descriptor using **obabel**'s `-L` option:

```
C:\Work>obabel -L descriptors
abonds    Number of aromatic bonds
atoms    Number of atoms
...
hellohalo    Count up the number of halogens (sort of)
...
title    For comparing a molecule's title
TPSA    topological polar surface area

C:\Work>obabel -L hellohalo
One of the descriptors
hellohalo    Count up the number of halogens (sort of)
This descriptor is not correlated with any
known property, living or dead.
 Datafile: hellohalo_smarts.txt
OBGroupContrib is definable
```

An easy way to test the descriptor is to use the title output format, and append the descriptor value to the title:

```
C:\Work>obabel -:C(Cl)(Cl)I -otxt --append hellohalo
8
1 molecule converted
```

There are a couple of points to note about the pattern file:

1. Although a SMARTS string may match a substructure of a molecule, the descriptor contribution is only assigned to the first atom of the match.

2. Where several SMARTS strings assign values to the same atom, only the final assignment is retained. As an example, the following set of patterns will assign a contribution of 0.4 to all atoms except for carbon atoms, which have a value of 1.0:

   ```
   ;heavy
   [*]     0.4    # All atoms
   [#6]    1.0    # All carbon atoms
   ```

3. If you wish to take into account contributions from hydrogen atoms, you should precede the `;heavy` section by a `;hydrogen` section. The values for the contributions in the latter section are multiplied by the number of hydrogens attached to the matching atom. For example, consider the following set of patterns:

   ```
   ;hydrogen
   [*]  0.2   # Hydrogens attached to all atoms
   C    1.0   # Hydrogens attached to an aliphatic carbon
   ;heavy
   C   10.0   # An aliphatic carbon
   ```

> For ethanol, this gives a value of 25.2: two carbons (20.0), five hydrogens attached to a carbon (5.0), and one other hydrogen (0.2).

For further inspiration, check out `psa.txt`, `mr.txt` and `logp.txt` in the `data` directory. These are the group contribution descriptions for Polar Surface Area, Molar Refractivity and LogP.

Chapter 18

Supported File Formats and Options

Chemists are a very imaginative group. They keep thinking of new file formats.

Indeed, these are not just simple differences in how chemical data is stored, but often completely different views on molecular representations. For example, some file formats ignore hydrogen atoms as "implicit," while others do not store bonding information. This is, in fact, a key reason for Open Babel's existence.

OpenBabel has support for 146 formats in total. It can read 108 formats and can write 107 formats. These formats are identified by a name (for example, `ShelX format`) and one or more short codes (in this case, `ins` or `res`). The titles of each section provide this information (for example, *ShelX format (ins, res)*).

The short code is used when using **`obabel`** or **`babel`** to convert files from one format to another:

```
obabel -iins myfile.ins -ocml
```

converts from ShelX format to Chemical Markup Language (in this case, no output file is specified and the output will be written to screen [stdout]). In fact, if the filename extension is the same as the file format code, then there is no need to specify the code. In other words, the following command will behave identically:

```
babel myfile.ins -ocml
```

As well as the general conversion options described elsewhere (see *Options*), each format may have its own options for either reading or writing. For example, the ShelX format has two options that affect reading of files, `s` and `b`. To set a file format option:

- For **Read Options**, precede the option with `-a` at the command line
- For **Write Options**, precede the option with `-x`

Mnemonic

To remember the correct switch for read or write options, think of "raw eggs": **r**ead is **a**, **w**rite is **x** ("eggs").

For example, if we wanted to set all bonds to single bonds when reading a ShelX format file, we could specify the `s` option:

```
babel -iins myfile.ins -ocml -as
```

More than one read (or write) option can be specified (e.g. `-ax -ay -az`). **`babel`** (but not **`obabel`**) also allows you to specify several options together (e.g. as `-axyz`).

Developer Note To set the file formats for an `OBConversion` object, use `SetInAndOutFormat(InCode, OutCode)`. To set a Read Option `s`, use `SetOptions("s", OBConversion::INOPTIONS)`.

18.1 Common cheminformatics formats

18.1.1 Canonical SMILES format (can)

A canonical form of the SMILES linear text format

The SMILES format is a linear text format which can describe the connectivity and chirality of a molecule. Canonical SMILES gives a single 'canonical' form for any particular molecule.

See also:

The "regular" *SMILES format (smi, smiles)* gives faster output, since no canonical numbering is performed.

Write Options

-a *Output atomclass like [C:2], if available*

-h *Output explicit hydrogens as such*

-i *Do not include isotopic or chiral markings*

-n *No molecule name*

-r *Radicals lower case eg ethyl is Cc*

-t *Molecule name only*

-F <atom numbers> *Generate Canonical SMILES for a fragment*

The atom numbers should be specified like "1 2 4 7".

-f <atomno> *Specify the first atom*

This atom will be used to begin the SMILES string.

-l <atomno> *Specify the last atom*

The output will be rearranged so that any additional SMILES added to the end will be attached to this atom. See the *SMILES format (smi, smiles)* for more information.

18.1.2 Chemical Markup Language (cml, mrv)

An XML format for interchange of chemical information.

This format writes and reads CML XML files. To write CML1 format rather than the default CML2, use the `-x1` option. To write the array form use `-xa` and to specify all hydrogens using the hydrogenCount attribute on atoms use `-xh`.

Crystal structures are written using the <crystal>, <xfract> (,...etc.) elements if the OBMol has a OBGenericDataType::UnitCell data.

All these forms are handled transparently during reading. Only a subset of CML elements and attributes are recognised, but these include most of those which define chemical structure, see below.

The following are read:

- Elements:
  - molecule, atomArray, atom, bondArray, bond, atomParity, bondStereo
  - name, formula, crystal, scalar (contains crystal data)
  - string, stringArray, integer, integerArray, float floatArray, builtin
- Attributes:
  - On <molecule>: id, title, ref(in CMLReact)
  - On <atom>: id, atomId, atomID, elementType, x2, y2, x3, y3, z3, xy2, xyz3, xFract, yFract, zFract, xyzFract, hydrogenCount, formalCharge, isotope, isotopeNumber, spinMultiplicity, radical(from Marvin), atomRefs4 (for atomParity)
  - On <bond>: atomRefs2, order, CML1: atomRef, atomRef1, atomRef2

Atom classes are also read and written. This is done using a specially formed atom id. When reading, if the atom id is of the form aN_M (where N and M are positive integers), then M is interpreted as the atom class. Such atom ids are automatically generated when writing an atom with an atom class.

Read Options

-2	*read 2D rather than 3D coordinates if both provided*

Write Options

-1	*write CML1 (rather than CML2)*
-a	*write array format for atoms and bonds*
-A	*write aromatic bonds as such, not Kekule form*
-m	*write metadata*
-x	*omit XML and namespace declarations*
-c	*continuous output: no formatting*
-p	*write properties*
-N <prefix>	*add namespace prefix to elements*

Comments

In the absence of hydrogenCount and any explicit hydrogen on an atom, implicit hydrogen is assumed to be present appropriate to the radical or spinMultiplicity attributes on the atom or its normal valency if they are not present.

The XML formats require the XML text to be well formed but generally interpret it fairly tolerantly. Unrecognised elements and attributes are ignored and there are rather few error messages when any required structures are not found. This laxity allows, for instance, the reactant and product molecules to be picked out of a CML React file using CML. Each format has an element which is regarded as defining the object that OpenBabel will convert. For CML this is <molecule>. Files can have multiple objects and these can be treated the same as with other multiple object formats like SMILES and MDL Molfile. So conversion can start at the nth object using the `-fn` option and finish before the end using the `-ln` option. Multiple object XML files also can be indexed and searched using FastSearch, although this has not yet been extensively tested.

18.1.3 InChI format (inchi)

IUPAC/NIST molecular identifier

Read Options

-X <Option string> *List of InChI options*

-n *molecule name follows InChI on same line*

-a *add InChI string to molecule name*

Write Options

Standard InChI is written unless certain InChI options are used

-K *output InChIKey only*

-t *add molecule name after InChI*

-w *ignore less important warnings*

These are: 'Omitted undefined stereo' 'Charges were rearranged' 'Proton(s) added/removed' 'Metal was disconnected'

-a *output auxiliary information*

-l *display InChI log*

-r *recalculate InChI; normally an input InChI is reused*

-s *recalculate wedge and hash bonds(2D structures only)*

Uniqueness options (see also `--unique` and `--sort` which are more versatile)

-u *output only unique molecules*

-U *output only unique molecules and sort them*

-e *compare first molecule to others*

This can also be done with *InChICompare format*:

```
babel first.smi second.mol third.cml -ok
```

-T <param> *truncate InChI according to various parameters*

See below for possible truncation parameters.

-X <Option string> *Additional InChI options*

See InChI documentation. These options should be space delimited in a single quoted string.

- Structure perception (compatible with stdInChI): `NEWPSOFF`, `DoNotAddH`, `SNon`
- Stereo interpretation (produces non-standard InChI): `SRel`, `SRac`, `SUCF`, `ChiralFlagON`, `ChiralFlagOFF`
- InChI creation options (produces non-standard InChI): `SUU`, `SLUUD`, `FixedH`, `RecMet`, `KET`, `15T`

	The following options are for convenience, e.g. `-xF` but produce non-standard InChI.
-F	*include fixed hydrogen layer*
-M	*include bonds to metal*

Comments

Truncation parameters used with `-xT`:

/formula	formula only
/connect	formula and connectivity only
/nostereo	ignore E/Z and sp3 stereochemistry
/nosp3	ignore sp3 stereochemistry
/noEZ	ignore E/Z steroeochemistry
/nochg	ignore charge and protonation
/noiso	ignore isotopes

Note that these can also be combined, e.g. `/nochg/noiso`

18.1.4 InChIKey (inchikey)

A hashed representation of the InChI.

The InChIKey is a fixed-length (27-character) condensed digital representation of an InChI, developed to make it easy to perform web searches for chemical structures.

An InChIKey consists of 14 characters (derived from the connectivity layer in the InChI), a hyphen, 9 characters (derived from the remaining layers), a character indicating the InChI version, a hyphen and a final checksum character. Contrast the InChI and InChIKey of the molecule represented by the SMILES string *CC(=O)Cl*:

```
obabel -:CC(=O)Cl -oinchi
InChI=1S/C2H3ClO/c1-2(3)4/h1H3

obabel -:CC(=O)Cl -oinchikey
WETWJCDKMRHUPV-UHFFFAOYSA-N
```

This is the same as using `-oinchi -xK` and can take the same options as the InChI format (see *InChI format (inchi)*):

```
obabel -:CC(=O)Cl -oinchi -xK
WETWJCDKMRHUPV-UHFFFAOYSA-N
```

Note that while a molecule with a particular InChI will always give the same InChIKey, the reverse is not true; there may exist more than one molecule which have different InChIs but yield the same InChIKey.

Note: This is a write-only format.

18.1.5 MDL MOL format (mdl, mol, sd, sdf)

Reads and writes V2000 and V3000 versions

Open Babel supports an extension to the MOL file standard that allows cis/trans and tetrahedral stereochemistry to be stored in 0D MOL files. The tetrahedral stereochemistry is stored as the atom parity, while the cis/trans stereochemistry is stored using Up and Down bonds similar to how it is represented in a SMILES string. Use the `S` option when reading or writing if you want to avoid storing or interpreting stereochemistry in 0D MOL files.

Read Options

-s *determine chirality from atom parity flags*

The default setting for 2D and 3D is to ignore atom parity and work out the chirality based on the bond stereochemistry (2D) or coordinates (3D). For 0D the default is already to determine the chirality from the atom parity.

-S *do not read stereochemistry from 0D MOL files*

Open Babel supports reading and writing cis/trans and tetrahedral stereochemistry to 0D MOL files. This is an extension to the standard which you can turn off using this option.

-T *read title only*

-P *read title and properties only*

When filtering an sdf file on title or properties only, avoid lengthy chemical interpretation by using the `T` or `P` option together with the *copy format*.

Write Options

-3 *output V3000 not V2000 (used for >999 atoms/bonds)*

-a *write atomclass if available*

-m *write no properties*

-w *use wedge and hash bonds from input (2D only)*

-v *always specify the valence in the valence field*

The default behavior is to only specify the valence if it is not consistent with the MDL valence model. So, for CH4 we don’t specify it, but we do for CH3. This option may be useful to preserve the correct number of implicit hydrogens if a downstream tool does not correctly implement the MDL valence model (but does honor the valence field).

-S *do not store cis/trans stereochemistry in 0D MOL files*

-A *output in Alias form, e.g. Ph, if present*

-E *add an ASCII depiction of the molecule as a property*

-H *use HYD extension (always on if mol contains zero-order bonds)*

18.1.6 Protein Data Bank format (ent, pdb)

Read Options

-s *Output single bonds only*

-b *Disable bonding entirely*

-c *Ignore CONECT records*

Write Options

-n *Do not write duplicate CONECT records to indicate bond order*

-o *Write origin in space group label (CRYST1 section)*

18.1.7 SMILES format (smi, smiles)

A linear text format which can describe the connectivity and chirality of a molecule

Open Babel implements the OpenSMILES specification.

It also implements an extension to this specification for radicals.

Note that the `l <atomno>` option, used to specify a "last" atom, is intended for the generation of SMILES strings to which additional atoms will be concatenated. If the atom specified has an explicit H within a bracket (e.g. `[nH]` or `[C@@H]`) the output will have the H removed along with any associated stereo symbols.

See also:

The *Canonical SMILES format (can)* produces a canonical representation of the molecule in SMILES format. This is the same as the `c` option below but may be more convenient to use.

Read Options

-a *Preserve aromaticity present in the SMILES*

This option should only be used if reading aromatic SMILES generated by the same version of Open Babel. Any other use will lead to undefined behavior. The advantage of this option is that it avoids aromaticity perception, thus speeding up reading SMILES.

-S *Clean stereochemistry*

By default, stereochemistry is accepted as given. If you wish to clean up stereochemistry (e.g. by removing tetrahedral stereochemistry where two of the substituents are identical) then specifying this option will reperceive stereocenters.

Write Options

-a *Output atomclass like [C:2], if available*

-c *Output in canonical form*

-U *Universal SMILES*

-I *Inchified SMILES*

-h *Output explicit hydrogens as such*

-i *Do not include isotopic or chiral markings*

-k *Create Kekule SMILES instead of aromatic*

-n *No molecule name*

-r *Radicals lower case eg ethyl is Cc*

-t *Molecule name only*

-x *append X/Y coordinates in canonical-SMILES order*

-C *'anti-canonical' random order (mostly for testing)*

-o <ordering> *Output in user-specified order*

Ordering should be specified like 4-2-1-3 for a 4-atom molecule. This gives canonical labels 1,2,3,4 to atoms 4,2,1,3 respectively, so that atom 4 will be visited first and the remaining atoms visited in a depth-first manner following the lowest canonical labels.

-O *Store the SMILES atom order as a space-separated string*

The string is stored as an OBPairData wth the name 'SMILES Atom Order'.

-F <atom numbers> *Generate SMILES for a fragment*

The atom numbers should be specified like "1 2 4 7".

-R *Do not reuse bond closure symbols*

-f <atomno> *Specify the first atom*

This atom will be used to begin the SMILES string.

-l <atomno> *Specify the last atom*

The output will be rearranged so that any additional SMILES added to the end will be attached to this atom.

-T <max seconds> *Specify the canonicalization timeout*

Canonicalization can take a while for symmetric molecules and a timeout is used. The default is 5 seconds.

18.1.8 SMILES format using Smiley parser (smy)

The Smiley parser presents an alternative to the standard SMILES parser (*SMILES format (smi, smiles)*). It was written to be strictly compatible with the OpenSMILES standard (http://opensmiles.org). In comparison, the standard parser is more forgiving to erroneous input, and also supports some extensions such as for radicals.

In addition, the Smiley parser returns detailed error messages when problems arise parsing or validating the SMILES, whereas the standard parser seldom describes the specific problem. For a detailed description of the OpenSMILES semantics, the specification should be consulted. In addition to syntactical and grammatical correctness, the Smiley parser also verifies some basic semantics.

Here are some examples of the errors reported:

```
SyntaxError: Bracket atom expression contains invalid trailing characters.
F.FB(F)F.[NH2+251][C@@H](CP(c1ccccc1)c1ccccc1)C(C)(C)C 31586112
               ^^
SyntaxError: Unmatched branch opening.
```

(continues on next page)

(continued from previous page)

```
CC(CC
  ^^^
SyntaxError: Unmatched branch closing.
CC)CC
^^^
SemanticsError: Unmatched ring bond.
C1CCC
^
SemanticsError: Conflicing ring bonds.
C-1CCCCC=1
```

Hydrogen with Hydrogen Count

Hydrogen atoms can not have a hydrogen count. Hydrogen bound to a hydrogen atom should be specified by two bracket atom expressions.

Examples:

```
[HH]         invalid
[HH1]        invalid (same as [HH]
[HH3]        invalid
[HH0]        valid (same as [H])
[H][H]       valid
```

Unmatched Ring Bond

Report unmatched ring bonds.

Example:

```
C1CCC
```

Conflicting Ring Bonds

When the bond type for ring bonds are explicitly specified at both ends, these should be the same.

Example:

```
C-1CCCCCC=1
```

Invalid Ring Bonds

There are two types of invalid ring bonds. The first is when two atoms both have the same two ring bonds. This would mean adding a parallel edge in the graph which is not allowed. The second type is similar but results in a self-loop by having a ring bond number twice.

Examples:

```
C12CCCC12      parallel bond
C11            self-loop bond
```

Invalid Chiral Valence

When an atom is specified as being chiral, it should have the correct number of neighboring atoms (possibly including an implicit H inside the bracket.

The valid valences are:

```
Tetrahedral (TH)          : 4
Allene (AL)               : 4 (*)
Square Planar (SP)        : 4
Trigonal Bypiramidal (TB) : 5
Octahedral(OH)            : 6

(*) The chiral atom has only 2 bonds but the neighbor's neighbors are
    counted: NC(Br)=[C@AL1]=C(F)I
```

Invalid Chiral Hydrogen Count

Chiral atoms can only have one hydrogen in their bracket since multiple hydrogens would make them not chiral.

Example:

```
C[C@H2]F
```

Note: This is a read-only format.

18.1.9 Sybyl Mol2 format (ml2, mol2, sy2)

Read Options

-c *Read UCSF Dock scores saved in comments preceding molecules*

Write Options

-l *Output ignores residue information (only ligands)*

-c *Write UCSF Dock scores saved in comments preceding molecules*

-u *Do not write formal charge information in UNITY records*

18.2 Utility formats

18.2.1 Compare molecules using InChI (k)

A utility format that allows you to compare molecules using their InChIs

The first molecule is compared with the rest, e.g.:

```
babel first.smi second.mol third.cml -ok
```

This is the same as using `-oinchi -xet` and can take the same options as InChI format (see *InChI format (inchi)*).

Note: This is a write-only format.

18.2.2 Confab report format (confabreport)

Assess performance of a conformer generator relative to a set of reference structures

Once a file containing conformers has been generated by *Confab*, the result can be compared to the original input structures or a set of reference structures using this output format.

Conformers are matched with reference structures using the molecule title. For every conformer, there should be a reference structure (but not necessarily vice versa).

Further information is available in the section describing *Confab*.

Note: This is a write-only format.

Write Options

-f <filename> *File containing reference structures*

-r <rmsd> *RMSD cutoff (default 0.5 Angstrom)*

The number of structures with conformers within this RMSD cutoff of the reference will be reported.

18.2.3 Copy raw text (copy)

A utility format for exactly copying the text of a chemical file format

This format allows you to filter molecules from multimolecule files without the risk of losing any additional information they contain, since no format conversion is carried out.

Warning: Currently not working correctly for files with Windows line endings.

Example:

Extract only structures that include at least one aromatic carbon (by matching the SMARTS pattern `[c]`):

```
babel -s '[c]' database.sdf -ocopy new.sd
```

Note: XML files may be missing non-object elements at the start or end and so may no longer be well formed.

Note: This is a write-only format.

18.2.4 General XML format (xml)

Calls a particular XML format depending on the XML namespace.

This is a general XML “format” which reads a generic XML file and infers its format from the namespace as given in a xmlns attribute on an element. If a namespace is recognised as associated with one of the XML formats in Open Babel, and the type of the object (e.g. a molecule) is appropriate to the output format then this is used to input a single object. If no namespace declaration is found the default format (currently CML) is used.

The process is repeated for any subsequent input so that it is possible to input objects written in several different schemas from the same document. The file `CMLandPubChem.xml` illustrates this and contains molecules in both CML and PubChem formats.

This implementation uses libxml2.

Note: This is a read-only format.

Read Options

-n	*Read objects of first namespace only*

18.2.5 Generic Output file format (dat, log, out, output)

Automatically detect and read computational chemistry output files

This format can be used to read ADF, Gaussian, GAMESS, PWSCF, Q-Chem, MOPAC, ORCA etc. output files by automatically detecting the file type.

Note: This is a read-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.2.6 MolPrint2D format (mpd)

An implementation of the circular fingerprint MolPrint2D

MolPrint2D is an atom-environment fingerprint developed by Bender et al [bmg2004] which has been used in QSAR studies and for measuring molecular similarity.

The format of the output is as follows:

```
[Molec_name]\t[atomtype];[layer]-[frequency]-[neighbour_type];
```

Example for the SMILES string `CC(=O)Cl`:

```
acid chloride   1;1-1-2;2-1-9;2-1-15;   2;1-1-1;1-1-9;1-1-15;
                9;1-1-2;2-1-1;2-1-15;   15;1-1-2;2-1-1;2-1-9;
```

Note: This is a write-only format.

Write Options

-n	*prefix molecule names with name of file*
-c	*use XML style separators instead*
-i	*use IDX atom types of babel internal*

18.2.7 Multilevel Neighborhoods of Atoms (MNA) (mna)

Iteratively generated 2D descriptors suitable for QSAR

Multilevel Neighborhoods of Atoms (MNA) descriptors are 2D molecular fragments suitable for use in QSAR modelling [fpbg99]. The format outputs a complete descriptor fingerprint per molecule. Thus, a 27-atom (including hydrogen) molecule would result in 27 descriptors, one per line.

MNA descriptors are generated recursively. Starting at the origin, each atom is appended to the descriptor immediately followed by a parenthesized list of its neighbours. This process iterates until the specified distance from the origin, also known as the depth of the descriptor.

Elements are simplified into 32 groups. Each group has a representative symbol used to stand for any element in that group:

Type	Elements
H	H
C	C
N	N
O	O
F	F
Si	Si
P	P
S	S
Cl	Cl
Ca	Ca
As	As
Se	Se
Br	Br
Li	Li, Na
B	B, Re
Mg	Mg, Mn
Sn	Sn, Pb
Te	Te, Po
I	I, At
Os	Os, Ir
Sc	Sc, Ti, Zr
Fe	Fe, Hf, Ta
Co	Co, Sb, W
Sr	Sr, Ba, Ra
Pd	Pd, Pt, Au

Continued on next page

Table 1 – continued from previous page

Type	Elements
Be	Be, Zn, Cd, Hg
K	K, Rb, Cs, Fr
V	V, Cr, Nb, Mo, Tc
Ni	Ni, Cu, Ge, Ru, Rh, Ag, Bi
In	In, La, Ce, Pr, Nd, Pm, Sm, Eu
Al	Al, Ga, Y, Gd, Tb, Dy, Ho, Er, Tm, Yb, Lu, Tl
R	R, He, Ne, Ar, Kr, Xe, Rn, Ac, Th, Pa, U, Np, Pu, Am, Cm, Bk, Cf, Es, Fm, Md, No, Lr, Db, Jl

Acyclic atoms are preceded by a hyphen “-” mark.

Here’s the multi-level neighborhood for the molecule represented by the SMILES string CC(=O)Cl:

```
# The contents of this file were derived from
# Title = acid chloride
-C(-H(-C)-H(-C)-H(-C)-C(-C-O-Cl))
-C(-C(-H-H-H-C)-O(-C)-Cl(-C))
-O(-C(-C-O-Cl))
-Cl(-C(-C-O-Cl))
-H(-C(-H-H-H-C))
-H(-C(-H-H-H-C))
-H(-C(-H-H-H-C))
```

Note: This is a write-only format.

Write Options

-L <num> *Levels (default = 2)*

18.2.8 Open Babel molecule report (molreport)

Generates a summary of the atoms and bonds in a molecule

Example output:

```
TITLE: Ethanol.mopout
FORMULA: C2H6O
MASS: 46.0684
ATOM:         1   C TYPE: C3     HYB:  3 CHARGE:  -0.2151
ATOM:         2   C TYPE: C3     HYB:  3 CHARGE:  -0.0192
ATOM:         3   O TYPE: O3     HYB:  3 CHARGE:  -0.3295
ATOM:         4   H TYPE: HC     HYB:  0 CHARGE:   0.0771
ATOM:         5   H TYPE: HC     HYB:  0 CHARGE:   0.0873
ATOM:         6   H TYPE: HC     HYB:  0 CHARGE:   0.0874
ATOM:         7   H TYPE: HC     HYB:  0 CHARGE:   0.0577
ATOM:         8   H TYPE: HC     HYB:  0 CHARGE:   0.0577
ATOM:         9   H TYPE: HC     HYB:  0 CHARGE:   0.1966
BOND:         0 START:         8 END:         2 ORDER:   1
BOND:         1 START:         6 END:         1 ORDER:   1
BOND:         2 START:         1 END:         2 ORDER:   1
BOND:         3 START:         1 END:         4 ORDER:   1
BOND:         4 START:         1 END:         5 ORDER:   1
```

(continues on next page)

(continued from previous page)

```
BOND:         5 START:         2 END:         3 ORDER:    1
BOND:         6 START:         2 END:         7 ORDER:    1
BOND:         7 START:         3 END:         9 ORDER:    1
```

See also:

Open Babel report format (report)

Note: This is a write-only format.

18.2.9 Open Babel report format (report)

A detailed report on the geometry of a molecule

The report format presents a report of various molecular information, including:

- Filename / molecule title
- Molecular formula
- Mass
- Exact mass (i.e., for high-resolution mass spectrometry, the mass of the most abundant elements)
- Total charge (if not electrically neutral)
- Total spin (if not singlet)
- Interatomic distances
- Atomic charges
- Bond angles
- Dihedral angles
- Chirality information (including which atoms are chiral)
- Additional comments in the input file

Example for benzene:

```
FILENAME: benzene.report
FORMULA: C6H6
MASS: 78.1118
EXACT MASS: 78.0469502
INTERATOMIC DISTANCES

              C   1      C   2      C   3      C   4      C   5      C   6
              ------------------------------------------------------------------
   C   1    0.0000
   C   2    1.3958     0.0000
   C   3    2.4176     1.3958     0.0000
   C   4    2.7916     2.4176     1.3958     0.0000
   C   5    2.4176     2.7916     2.4176     1.3958     0.0000
   C   6    1.3958     2.4176     2.7916     2.4176     1.3958     0.0000
   H   7    1.0846     2.1537     3.4003     3.8761     3.4003     2.1537
   H   8    2.1537     1.0846     2.1537     3.4003     3.8761     3.4003
   H   9    3.4003     2.1537     1.0846     2.1537     3.4003     3.8761
```

(continues on next page)

(continued from previous page)

```
   H  10     3.8761     3.4003     2.1537     1.0846     2.1537     3.4003
   H  11     3.4003     3.8761     3.4003     2.1537     1.0846     2.1537
   H  12     2.1537     3.4003     3.8761     3.4003     2.1537     1.0846

              H   7      H   8      H   9      H  10      H  11      H  12
              ------------------------------------------------------------------
   H   7     0.0000
   H   8     2.4803     0.0000
   H   9     4.2961     2.4804     0.0000
   H  10     4.9607     4.2961     2.4803     0.0000
   H  11     4.2961     4.9607     4.2961     2.4803     0.0000
   H  12     2.4803     4.2961     4.9607     4.2961     2.4804     0.0000

ATOMIC CHARGES
   C   1   -0.1000000000
   C   2   -0.1000000000
   C   3   -0.1000000000
   C   4   -0.1000000000
   C   5   -0.1000000000
   C   6   -0.1000000000
   H   7    0.1000000000
   H   8    0.1000000000
   H   9    0.1000000000
   H  10    0.1000000000
   H  11    0.1000000000
   H  12    0.1000000000

BOND ANGLES
   7    1    2   HC  Car  Car    120.000
   1    2    3  Car  Car  Car    120.000
   1    2    8  Car  Car   HC    120.000
   8    2    3   HC  Car  Car    120.000
   2    3    4  Car  Car  Car    120.000
   2    3    9  Car  Car   HC    120.000
   9    3    4   HC  Car  Car    120.000
   3    4    5  Car  Car  Car    120.000
   3    4   10  Car  Car   HC    120.000
  10    4    5   HC  Car  Car    120.000
   4    5    6  Car  Car  Car    120.000
   4    5   11  Car  Car   HC    120.000
  11    5    6   HC  Car  Car    120.000
   5    6    1  Car  Car  Car    120.000
   5    6   12  Car  Car   HC    120.000
  12    6    1   HC  Car  Car    120.000
   6    1    2  Car  Car  Car    120.000
   6    1    7  Car  Car   HC    120.000
   2    1    7  Car  Car   HC    120.000
   3    2    8  Car  Car   HC    120.000
   4    3    9  Car  Car   HC    120.000
   5    4   10  Car  Car   HC    120.000
   6    5   11  Car  Car   HC    120.000
   1    6   12  Car  Car   HC    120.000

TORSION ANGLES
   6    1    2    3      0.026
   6    1    2    8   -179.974
   7    1    2    3    179.974
```

(continues on next page)

(continued from previous page)

```
  7    1    2    8      -0.026
  1    2    3    4      -0.026
  1    2    3    9    -179.974
  8    2    3    4     179.974
  8    2    3    9       0.026
  2    3    4    5       0.026
  2    3    4   10     179.974
  9    3    4    5     179.974
  9    3    4   10      -0.026
  3    4    5    6      -0.026
  3    4    5   11     179.974
 10    4    5    6    -179.974
 10    4    5   11       0.026
  4    5    6    1       0.026
  4    5    6   12     179.974
 11    5    6    1    -179.974
 11    5    6   12      -0.026
  5    6    1    2      -0.026
  5    6    1    7    -179.974
 12    6    1    2     179.974
 12    6    1    7       0.026
```

See also:

Open Babel molecule report (molreport)

Note: This is a write-only format.

18.2.10 Outputs nothing (nul)

Note: This is a write-only format.

18.2.11 Read and write raw text (text)

Facilitates the input of boilerplate text with babel commandline

18.2.12 Title format (txt)

Displays and reads molecule titles

18.2.13 XYZ cartesian coordinates format (xyz)

A generic coordinate format

The "XYZ" chemical file format is widely supported by many programs, although no formal specification has been published. Consequently, Open Babel attempts to be extremely flexible in parsing XYZ format files. Similar formats include Tinker XYZ and UniChem XYZ which differ slightly in the format of the files. (Notably, UniChem XYZ uses the atomic number rather than element symbol for the first column.)

- Line one of the file contains the number of atoms in the file.
- Line two of the file contains a title, comment, or filename.

Any remaining lines are parsed for atom information. Lines start with the element symbol, followed by X, Y, and Z coordinates in angstroms separated by whitespace.

Multiple molecules / frames can be contained within one file.

On **output**, the first line written is the number of atoms in the molecule (warning - the number of digits is limited to three for some programs, e.g. Maestro). Line two is the title of the molecule or the filename if no title is defined. Remaining lines define the atoms in the file. The first column is the atomic symbol (right-aligned on the third character), followed by the XYZ coordinates in "10.5" format, in angstroms. This means that all coordinates are printed with five decimal places.

Example:

```
12
benzene example
  C        0.00000        1.40272        0.00000
  H        0.00000        2.49029        0.00000
  C       -1.21479        0.70136        0.00000
  H       -2.15666        1.24515        0.00000
  C       -1.21479       -0.70136        0.00000
  H       -2.15666       -1.24515        0.00000
  C        0.00000       -1.40272        0.00000
  H        0.00000       -2.49029        0.00000
  C        1.21479       -0.70136        0.00000
  H        2.15666       -1.24515        0.00000
  C        1.21479        0.70136        0.00000
  H        2.15666        1.24515        0.00000
```

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.3 Other cheminformatics formats

18.3.1 Accelrys/MSI Biosym/Insight II CAR format (arc, car)

Note: This is a read-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.3.2 Accelrys/MSI Cerius II MSI format (msi)

Note: This is a read-only format.

18.3.3 Accelrys/MSI Quanta CSR format (csr)

Note: This is a write-only format.

18.3.4 MCDL format (mcdl)

Modular Chemical Descriptor Language

As described in [gb2001].

Here's an example conversion from SMILES to MCDL:

```
obabel -:"CC(=O)Cl" -omcdl
CHHH;COCl[2]
```

18.3.5 MSI BGF format (bgf)

18.3.6 PubChem format (pc)

An XML format containing information on PubChem entries.

PubChem is a freely-available database of chemical compounds and their properties.

OpenBabel only extracts the chemical structure information, and the potentially large amount of other information is currently ignored. The format seems to handle multiple conformers, but only one is read (this needs testing).

Note: This is a read-only format.

18.3.7 Wiswesser Line Notation (wln)

A chemical line notation developed by Wiswesser

WLN was invented in 1949, by William J. Wiswesser, as one of the first attempts to codify chemical structure as a line notation, enabling collation on punched cards using automatic tabulating machines and early electronic computers. WLN was a forerunner to the SMILES notation used in modern cheminformatics systems, which attempted to simplify the complex rules used in WLN encoding (at the expense of brevity) to come up with an algorithmic system more suitable for implementation on computers, where historically WLN was typically encoded by hand by trained registrars.

WLN encoding makes use of uppercase letters, digits, spaces and punctuation:

- E Bromine atom
- F Fluorine atom

- G Chlorine atom
- H Hydrogen atom
- I Iodine atom
- Q Hydroxyl group, -OH
- R Benzene ring
- S Sulfur atom
- U Double bond
- UU Triple bond
- V Carbonyl, -C(=O)-
- C Unbranched carbon multiply bonded to non-carbon atom
- K Nitrogen atom bonded to more than three other atoms
- L First symbol of a carbocyclic ring notation
- M Imino or imido -NH-group
- N Nitrogen atom, hydrogen free, bonded to fewer than 4 atoms
- O Oxygen atom, hydrogen-free
- T First symbol of a heterocyclic ring notation
- W Non-linear dioxo group, as in -NO2 or -SO2-
- X Carbon attached to four atoms other than hydrogen
- Y Carbon attached to three atoms other then hydrogen
- Z Amino and amido NH2 group
- <digit> Digits ‘1’ to ‘9’ denote unbranched alkyl chains
- & Sidechain terminator or, after a space, a component separator

For a more complete description of the grammar see Smith’s book [1], which more accurately reflects the WLN commonly encountered than Wiswesser’s book [2]. Additional WLN dialects include inorganic salts, and methyl contractions.

Here are some examples of WLN strings along with a corresponding SMILES string:

- WN3 [O-][N+](=O)CCC
- G1UU1G ClC#CCl
- VH3 O=CCCC
- NCCN N#CC#N
- ZYZUM NC(=N)N
- QY CC(C)O
- OV1 &-NA- CC(=O)[O-].[Na+]
- RM1R c1ccccc1NCc2ccccc2
- T56 BMJ B D - DT6N CNJ BMR BO1 DN1 & 2N1 & 1 EMV1U1 (osimertinib) Cn1cc(c2c1cccc2)c3ccnc(n3)Nc4cc(c(cc4OC)N(C)CCN(C)C)NC(=O)C=C

This reader was contributed by Roger Sayle (NextMove Software). The text of this description was taken from his Bio-IT World poster [3]. Note that not all of WLN is currently supported; however, about 76% of the WLN strings found in PubChem can be interpreted.

1. Elbert G. Smith, "The Wiswesser Line-Formula Chemical Notation", McGraw-Hill Book Company publishers, 1968.
2. William J. Wiswesser, "A Line-Formula Chemical Notation", Thomas Crowell Company publishers, 1954.
3. Roger Sayle, Noel O'Boyle, Greg Landrum, Roman Affentranger. "Open sourcing a Wiswesser Line Notation (WLN) parser to facilitate electronic lab notebook (ELN) record transfer using the Pistoia Alliance's UDM (Unified Data Model) standard." BioIT World. Apr 2019. https://www.nextmovesoftware.com/posters/Sayle_WisswesserLineNotation_BioIT_201904.pdf

Note: This is a read-only format.

18.4 Computational chemistry formats

18.4.1 ABINIT Output Format (abinit)

Note: This is a read-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.4.2 ACES input format (acesin)

ACES is a set of programs that performs ab initio quantum chemistry calculations.

Note: This is a write-only format.

18.4.3 ACES output format (acesout)

ACES is a set of programs that performs ab initio quantum chemistry calculations.

Note: This is a read-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.4.4 ADF Band output format (adfband)

Note: This is a read-only format.

18.4.5 ADF DFTB output format (adfdftb)

Note: This is a read-only format.

18.4.6 ADF cartesian input format (adf)

Note: This is a write-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.4.7 ADF output format (adfout)

Note: This is a read-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.4.8 CAChe MolStruct format (cac, cache)

Note: This is a write-only format.

18.4.9 CASTEP format (castep)

The format used by CASTEP.

Note: This is a read-only format.

18.4.10 Cacao Cartesian format (caccrt)

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.4.11 Cacao Internal format (cacint)

Note: This is a write-only format.

18.4.12 Crystal 09 output format (c09out)

Note: This is a read-only format.

Read Options

-s	*Consider single bonds only*
-b	*Disable bonding entirely*

18.4.13 Culgi object file format (cof)

Culgi format

No options currently

18.4.14 DALTON input format (dalmol)

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

Write Options

-a	*write input in atomic units instead of Angstrom*
-b	*write input using the ATOMBASIS format*
-k <basis>	*specify basis set to use* e.g. `-xk STO-3G`

18.4.15 DALTON output format (dallog)

Note: This is a read-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.4.16 DMol3 coordinates format (dmol, outmol)

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.4.17 Extended XYZ cartesian coordinates format (exyz)

A format used by ORCA-AICCM

The "EXYZ" chemical file format is an extended version of the standard "XYZ" chemical file format with additional keywords and informations about the unit cell and virtual atoms.

- Line one of the file contains the number of atoms in the file.
- Line two of the file contains a title, comment, filename and/or the following keywords: `%PBC` or `%VIRTUAL`

Any remaining lines are parsed for atom information until a blank line. These lines start with the element symbol, followed by X, Y, and Z coordinates in angstroms separated by whitespace and - if `%VIRTUAL` is specified - the optional word `VIRTUAL` to mark virtual atoms. If `%PBC` is specified a second block will be present containing the 3 vectors for the unit cell in angstrom and the offset as shown in the example below:

```
4
%PBC
   C        0.00000        1.40272        0.00000
   H        0.00000        2.49029        0.00000
   C       -1.21479        0.70136        0.00000
   H       -2.15666        1.24515        0.00000

Vector1    2.445200    0.000000    0.000000
Vector2    0.000000    1.000000    0.000000
Vector3    0.000000    0.000000    1.000000
Offset     0.000000    0.000000    0.000000
```

On **output**, the first line written is the number of atoms in the molecule. Line two is the title of the molecule or the filename if no title is defined. Remaining lines define the atoms in the file. The first column is the atomic symbol (right-aligned on the third character), followed by the XYZ coordinates in "15.5" format separated by an addition whitespace, in angstroms. This means that all coordinates are printed with five decimal places.

The next block starts with a blank line to separate the coordinates from the unit cell vectors followed by the vectors of the unit cell marked with the keywords Vector1/2/3. The vectors themselves are written in the same format as the atom coordinates. The last line contains the keyword Offset and the offset of the unit cell. The unit is always angstrom.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.4.18 FHIaims XYZ format (fhiaims)

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.4.19 Fenske-Hall Z-Matrix format (fh)

Note: This is a write-only format.

18.4.20 GAMESS Input (gamin, inp)

Write Options

-k <keywords>	*Use the specified keywords for input*
-f <file>	*Read the file specified for input keywords*

18.4.21 GAMESS Output (gam, gamess, gamout)

Note: This is a read-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*
-c	*Read multiple conformers*

18.4.22 GAMESS-UK Input (gukin)

18.4.23 GAMESS-UK Output (gukout)

18.4.24 GULP format (got)

The format used by GULP (General Utility Lattice Program).

Note: This is a read-only format.

18.4.25 Gaussian Input (com, gau, gjc, gjf)

Note: This is a write-only format.

Write Options

-b	*Output includes bonds*
-k <keywords>	*Use the specified keywords for input*
-f <file>	*Read the file specified for input keywords*
-u	*Write the crystallographic unit cell, if present.*

18.4.26 Gaussian Output (g03, g09, g16, g92, g94, g98, gal)

Note: This is a read-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.4.27 Gaussian Z-Matrix Input (gzmat)

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

Write Options

-k <keywords>	*Use the specified keywords for input*
-f <file>	*Read the file specified for input keywords*

18.4.28 Gaussian formatted checkpoint file format (fch, fchk, fck)

A formatted text file containing the results of a Gaussian calculation

Currently supports reading molecular geometries from fchk files. More to come.

Note: This is a read-only format.

Read Options

-s	*Single bonds only*
-b	*No bond perception*

18.4.29 HyperChem HIN format (hin)

18.4.30 Jaguar input format (jin)

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.4.31 Jaguar output format (jout)

Note: This is a read-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.4.32 MOPAC Cartesian format (mop, mopcrt, mpc)

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

Write Options

-k <keywords>	*Use the specified keywords for input*
-f <file>	*Read the file specified for input keywords*
-u	*Write the crystallographic unit cell, if present.*

18.4.33 MOPAC Internal (mopin)

Write Options

-k <keywords>	*Use the specified keywords for input*
-f <file>	*Read the file specified for input keywords*

18.4.34 MOPAC Output format (moo, mopout)

Note: This is a read-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.4.35 MPQC output format (mpqc)

Note: This is a read-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.4.36 MPQC simplified input format (mpqcin)

Note: This is a write-only format.

18.4.37 Molpro input format (mp)

Note: This is a write-only format.

18.4.38 Molpro output format (mpo)

Note: This is a read-only format.

Read Options

-s *Output single bonds only*

-b *Disable bonding entirely*

18.4.39 NWChem input format (nw)

Note: This is a write-only format.

18.4.40 NWChem output format (nwo)

Note: This is a read-only format.

Read Options

-s *Output single bonds only*

-f *Overwrite molecule if more than one*

calculation with different molecules is present in the output file (last calculation will be prefered)

-b *Disable bonding entirely*

18.4.41 ORCA input format (orcainp)

Note: This is a write-only format.

Write Options

-k <keywords> *Use the specified keywords for input*

-f <file> *Read the file specified for input keywords*

18.4.42 ORCA output format (orca)

Note: This is a read-only format.

Read Options

-s *Output single bonds only*

-b *Disable bonding entirely*

18.4.43 PWscf format (pwscf)

The format used by PWscf, part of Quantum Espresso.

Note: This is a read-only format.

18.4.44 Parallel Quantum Solutions format (pqs)

18.4.45 Q-Chem input format (qcin)

Note: This is a write-only format.

Write Options

-k <keywords>	*Use the specified keywords for input*
-f <file>	*Read the file specified for input keywords*

18.4.46 Q-Chem output format (qcout)

Note: This is a read-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.4.47 TurboMole Coordinate format (tmol)

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*
-a	*Input in Angstroms*

Write Options

-a	*Output Angstroms*

18.4.48 Turbomole AOFORCE output format (aoforce)

Read vibrational frequencies and intensities

Note: This is a read-only format.

18.4.49 VASP format (CONTCAR, POSCAR, VASP)

Reads in data from POSCAR and CONTCAR to obtain information from VASP calculations.

Due to limitations in Open Babel's file handling, reading in VASP files can be a bit tricky; the client that is using Open Babel must use OBConversion::ReadFile() to begin the conversion. This change is usually trivial. Also, the complete path to the CONTCAR/POSCAR file must be provided, otherwise the other files needed will not be found.

Both VASP 4.x and 5.x POSCAR formats are supported.

By default, atoms are written out in the order they are present in the input molecule. To sort by atomic number specify `-xw`. To specify the sort order, use the `-xz` option.

Read Options

-s *Output single bonds only*

-b *Disable bonding entirely*

Write Options

-w *Sort atoms by atomic number*

-z <list of atoms> *Specify the order to write out atoms*

'atom1 atom2 …': atom1 first, atom2 second, etc. The remaining atoms are written in the default order or (if `-xw` is specified) in order of atomic number.

-4 *Write a POSCAR using the VASP 4.x specification.*

The default is to use the VASP 5.x specification.

18.4.50 ZINDO input format (zin)

The input format for the semiempirical quantum-mechanics program ZINDO.

Note: This is a write-only format.

Write Options

-c *Write an input file for the CNDO/INDO program.*

18.5 Molecular fingerprint formats

18.5.1 FPS text fingerprint format (Dalke) (fps)

The FPS file format for fingerprints was developed by Andrew Dalke to define and promote common file formats for storing and exchanging cheminformatics fingerprint data sets, and to develop tools which work with that format. For more information, see http://chem-fingerprints.googlecode.com

Any molecule without a title is given its index in the file as title.

A list of available fingerprint types can be obtained by:

```
obabel -L fingerprints
```

Note: This is a write-only format.

Write Options

-f <id>	*Fingerprint type*
-N <num>	*Fold to specified number of bits, 32, 64, 128, etc.*
-p	*Use full input path as source, not just filename*
-t <text>	*Use <text> as source in header*

18.5.2 Fastsearch format (fs)

Fingerprint-aided substructure and similarity searching

Writing to the fs format makes an index of a multi-molecule datafile:

```
obabel dataset.sdf -ofs
```

This prepares an index `dataset.fs` with default parameters, and is slow (~30 minutes for a 250,000 molecule file).

However, when reading from the fs format searches are much faster, a few seconds, and so can be done interactively.

The search target is the parameter of the `-s` option and can be slightly extended SMILES (with `[#n]` atoms and ~ bonds) or the name of a file containing a molecule.

Several types of searches are possible:

- Identical molecule:

  ```
  obabel index.fs -O outfile.yyy -s SMILES exact
  ```

- Substructure:

  ```
  obabel index.fs -O outfile.yyy  -s SMILES   or
  obabel index.fs -O outfile.yyy  -s filename.xxx
  ```

  where `xxx` is a format id known to OpenBabel, e.g. sdf

- Molecular similarity based on Tanimoto coefficient:

```
obabel index.fs -O outfile.yyy -at15  -sSMILES  # best 15 molecules
obabel index.fs -O outfile.yyy -at0.7 -sSMILES  # Tanimoto >0.7
obabel index.fs -O outfile.yyy -at0.7,0.9 -sSMILES
#     Tanimoto >0.7 && Tanimoto < 0.9
```

The datafile plus the `-ifs` option can be used instead of the index file.

NOTE on 32-bit systems the datafile MUST NOT be larger than 4GB.

Dative bonds like -[N+][O-](=O) are indexed as -N(=O)(=O), and when searching the target molecule should be in the second form.

See also:

Molecular fingerprints and similarity searching

Read Options

-t <num>	*Do similarity search:<num>mols or <num> as min Tanimoto*
-a	*Add Tanimoto coeff to title in similarity search*
-l <num>	*Maximum number of candidates. Default<4000>*
-e	*Exact match* Alternative to using exact in `-s` parameter, see above
-n	*No further SMARTS filtering after fingerprint phase*

Write Options

-f <num>	*Fingerprint type* If not specified, the default fingerprint (currently FP2) is used
-N <num>	*Fold fingerprint to <num> bits*
-u	*Update an existing index*

18.5.3 Fingerprint format (fpt)

Generate or display molecular fingerprints.

This format constructs and displays fingerprints and (for multiple input objects) the Tanimoto coefficient and whether a superstructure of the first object.

A list of available fingerprint types can be obtained by:

```
babel -L fingerprints
```

The current default type FP2 is is of the Daylight type, indexing a molecule based on the occurrence of linear fragment up to 7 atoms in length. To use a fingerprint type other than the default, use the `-xf` option, for example:

```
babel infile.xxx -ofpt -xfFP3
```

For a single molecule the fingerprint is output in hexadecimal form (intended mainly for debugging).

With multiple molecules the hexadecimal form is output only if the `-xh` option is specified. But in addition the Tanimoto coefficient between the first molecule and each of the subsequent ones is displayed. If the first molecule is a substructure of the target molecule a note saying this is also displayed.

The Tanimoto coefficient is defined as:

```
Number of bits set in (patternFP & targetFP) / Number of bits in (patternFP |␣
↪targetFP)
```

where the boolean operations between the fingerprints are bitwise.

The Tanimoto coefficient has no absolute meaning and depends on the design of the fingerprint.

Use the `-xs` option to describe the bits that are set in the fingerprint. The output depends on the fingerprint type. For Fingerprint FP4, each bit corresponds to a particular chemical feature, which are specified as SMARTS patterns in `SMARTS_InteLigand.txt`, and the output is a tab-separated list of the features of a molecule. For instance, a well-known molecule gives:

```
Primary_carbon: Carboxylic_acid: Carboxylic_ester: Carboxylic_acid_derivative:
Vinylogous_carbonyl_or_carboxyl_derivative: Vinylogous_ester: Aromatic:
Conjugated_double_bond: C_ONS_bond: 1,3-Tautomerizable: Rotatable_bond: CH-acidic:
```

For the path-based fingerprint FP2, the output from the `-xs` option is instead a list of the chemical fragments used to set bits, e.g.:

```
$ obabel -:"CCC(=O)Cl" -ofpt -xs -xf FP2
>
0 6 1 6 <670>
0 6 1 6 1 6 <260>
0 8 2 6 <623>
...etc
```

where the first digit is 0 for linear fragments but is a bond order for cyclic fragments. The remaining digits indicate the atomic number and bond order alternatively. Note that a bond order of 5 is used for aromatic bonds. For example, bit 623 above is the linear fragment O=C (8 for oxygen, 2 for double bond and 6 for carbon).

Note: This is a write-only format.

Write Options

-f <id>	*fingerprint type*
-N <num>	*fold to specified number of bits, 32, 64, 128, etc.*
-h	*hex output when multiple molecules*
-o	*hex output only*
-s	*describe each set bit*
-u	*describe each unset bit*

18.6 Crystallography formats

18.6.1 ACR format (acr)

CaRIne ASCII Crystal format (ACR)

Note: This is a read-only format.

Read Options

-s	*Consider single bonds only*

18.6.2 CSD CSSR format (cssr)

Note: This is a write-only format.

18.6.3 Crystallographic Information File (cif)

The CIF file format is the standard interchange format for small-molecule crystal structures

Fractional coordinates are converted to cartesian ones using the following convention:

- The x axis is parallel to a
- The y axis is in the (a,b) plane
- The z axis is along c*

Ref: Int. Tables for Crystallography (2006), vol. B, sec 3.3.1.1.1 (the matrix used is the 2nd form listed)

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*
-B	*Use bonds listed in CIF file from _geom_bond_etc records (overrides option b)*

Write Options

-g	*Write bonds using _geom_bond_etc fields*

18.6.4 Free Form Fractional format (fract)

General purpose crystallographic format

The "free-form" fractional format attempts to allow for input from a range of fractional / crystallography file formats. As such, it has only a few restrictions on input:

- Line one of the file contains a title or comment.
- Line two of the file contains the unit cell parameters separated by whitespace and/or commas (i.e. "a b c alpha beta gamma").
- Any remaining lines are parsed for atom information. Lines start with the element symbol, followed by fractional X, Y, and Z coordinates (in angstroms) separated by whitespace.

Any numeric input (i.e., unit cell parameters, XYZ coordinates) can include designations of errors, although this is currently ignored. For example:

```
C 1.00067(3) 2.75(2) 3.0678(12)
```

will be parsed as:

```
C 1.00067 2.75 3.0678
```

When used as an **output** format, The first line written is the title of the molecule or the filename if no title is defined. If a molecule has a defined unit cell, then the second line will be formatted as:

```
a b c alpha beta gamma
```

where a, b, c are the unit cell vector lengths, and alpha, beta, and gamma are the angles between them. These numbers are formatted as "10.5", which means that 5 decimal places will be output for all numbers. In the case where no unit cell is defined for the molecule, the vector lengths will be defined as 1.0, and the angles to 90.0 degrees.

Remaining lines define the atoms in the file. The first column is the atomic symbol, followed by the XYZ coordinates in 10.5 format (in angstroms).

Here is an example file:

```
ZnO test file
3.14 3.24 5.18 90.0 90.0 120.0
O 0.66667  0.33333  0.3750
O 0.33333  0.66667  0.8750
Zn 0.66667  0.33333  0.0000
Zn 0.33333  0.66667  0.5000
```

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.6.5 Macromolecular Crystallographic Info (mcif, mmcif)

Read Options

-s	*Output single bonds only*
-p	*Apply periodic boundary conditions for bonds*
-b	*Disable bonding entirely*
-w	*Wrap atomic coordinates into unit cell box*

18.6.6 POS cartesian coordinates format (pos)

A generic coordinate format

The “POS” file format is a modified version of the “XYZ” general format

- Line one of the file contains the number of atoms in the file.
- Line two of the file contains a title, comment, or filename.

Example:

```
.. note:: This is a read-only format.
```

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.6.7 ShelX format (ins, res)

Note: This is a read-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.7 Reaction formats

18.7.1 CML Reaction format (cmlr)

A minimal implementation of the CML Reaction format

This implementation uses libxml2.

Write Options

-1	*output CML1 (rather than CML2)*
-a	*output array format for atoms and bonds*
-l	*molecules NOT in MoleculeList*
-h	*use hydrogenCount for all hydrogens*
-x	*omit XML declaration*
-r	*omit rate constant data*
-N <prefix>	*add namespace prefix to elements*

-M *add obr prefix on non-CMLReact elements*

-p *add properties to molecules*

Comments

The implementation of this format which reads and writes to and from OBReaction objects is fairly minimal at present. (Currently the only other reaction format in OpenBabel is RXN.) During reading, only the elements <reaction>, <reactant>, <product> and <molecule> are acted upon (the last through CML). The molecules can be collected together in a list at the start of the file and referenced in the reactant and product via e.g. <molecule ref=”mol1”>.

On writing, the list format can be specified with the `-xl` option. The list containers are <moleculeList> and <reactionList> and the overall wrapper is <mechanism>. These are non-standard CMLReact element names and would have to be changed (in the code) to <list>,<list> and <cml> if this was unacceptable.

18.7.2 MDL RXN format (rxn)

The MDL reaction format is used to store information on chemical reactions.

Output Options, e.g. -xA A output in Alias form, e.g. Ph, if present G <option> how to handle any agents present

One of the following options should be specifed:

- **agent - Treat as an agent (default). Note that some programs** may not read agents in RXN files.
- reactant - Treat any agent as a reactant
- product - Treat any agent as a product
- ignore - Ignore any agent
- both - Treat as both a reactant and a product

18.7.3 RInChI (rinchi)

The Reaction InChI

The Reaction InChI (or RInChI) is intended to be a unique string that describes a reaction. This may be useful for indexing and searching reaction databases. As with the InChI it is recommended that you always keep the original reaction information and use the RInChI in addition.

The RInChI format is a hierarchical, layered description of a reaction with different levels based on the Standard InChI representation of each structural component participating in the reaction.

Note: This is a write-only format.

Write Options

-e *Treat this reaction as an equilibrium reaction*

Layer 5 of the generated RInChI will have /d=

18.7.4 Reaction SMILES format (rsmi)

Write Options

-r *radicals lower case eg ethyl is Cc*

18.8 Image formats

18.8.1 ASCII format (ascii)

2D depiction of a single molecule as ASCII text

This format generates a 2D depiction of a molecule using only ASCII text suitable for a command-line console, or a text file. For example:

```
obabel -:c1ccccc1C(=O)Cl -oascii -xh 20

       __
    __/__\_
  _/__/    \__
_/_/          \__
|               |
|             | |
|             | |
|             | |
|             | |
|___            _               Cl
  \_\__       _/ \_          __
     \_\_  __/     \__    __/
       \__/           \__/
                       | |
                       | |
                       | |
                       | |

                        O
```

If the image appears elongated or squat, the aspect ratio should be changed from its default value of 1.5 using the `-xa <ratio>` option. To help determine the correct value, use the `-xs` option to display a square.

Note: This is a write-only format.

Write Options

-w <characters> *Image width in characters, default 79*

-h <characters> *Image height in characters, default is width/aspect*

-a <ratio> *Aspect ratio of character height:width, default is 1.5*

-s *Display a square - this is useful for correcting the aspect ratio*

-t *Write the output molecule index and the title*

-m *Include a margin around the depiction*

18.8.2 PNG 2D depiction (png)

or add/extract chemical structures from a .png file

The PNG format has several uses. The most common is to generate a `.png` file for one or more molecules. 2D coordinates are generated if not present:

```
obabel mymol.smi -O image.png
```

Chemical structure data can be embedded in the `.png` file (in a `tEXt` chunk):

```
obabel mymol.mol -O image.png -xO molfile
```

The parameter of the `-xO` option specifies the format (“file”can be added). Note that if you intend to embed a 2D or 3D format, you may have to call `--gen2d` or `--gen3d` to generate the required coordinates if they are not present in the input.

Molecules can also be embedded in an existing PNG file:

```
obabel existing.png mymol1.smi mymol2.mol -O augmented.png -xO mol
```

Reading from a PNG file will extract any embedded chemical structure data:

```
obabel augmented.png -O contents.sdf
```

Read Options

-y <additional chunk ID> *Look also in chunks with specified ID*

Write Options

-p <pixels> *image size, default 300*

-w <pixels> *image width (or from image size)*

-h <pixels> *image height (or from image size)*

-c <num> *number of columns in table*

-r <num> *number of rows in table*

-N <num> *max number objects to be output*

-u *no element-specific atom coloring*

Use this option to produce a black and white diagram

-U *do not use internally-specified color*

e.g. atom color read from cml or generated by internal code

-b <color> *background color, default white*

e.g `-xb yellow` or `-xb #88ff00` `-xb none` is transparent. Just `-xb` is black with white bonds. The atom symbol colors work with black and white backgrounds, but may not with other colors.

-B <color> *bond color, default black*

e.g `-xB` yellow or `-xB #88ff00`

-C *do not draw terminal C (and attached H) explicitly*

The default is to draw all hetero atoms and terminal C explicitly, together with their attched hydrogens.

-a *draw all carbon atoms*

So propane would display as H3C-CH2-CH3

-d *do not display molecule name*

-m *do not add margins to the image*

This only applies if there is a single molecule to depict. Implies -xd.

-s *use asymmetric double bonds*

-t *use thicker lines*

-A *display aliases, if present*

This applies to structures which have an alternative, usually shorter, representation already present. This might have been input from an A or S superatom entry in an sd or mol file, or can be generated using the –genalias option. For example:

```
obabel -:"c1cc(C=O)ccc1C(=O)O" -O out.png
       --genalias -xA
```

would add a aliases COOH and CHO to represent the carboxyl and aldehyde groups and would display them as such in the svg diagram. The aliases which are recognized are in data/superatom.txt, which can be edited.

-O <format ID> *Format of embedded text*

For example, `molfile` or `smi`. If there is no parameter, input format is used.

-y <additional chunk ID> *Write to a chunk with specified ID*

Comments

If Cairo was not found when Open Babel was compiled, then the 2D depiction will be unavailable. However, it will still be possible to extract and embed chemical data in `.png` files.

See also:

PNG 2D depiction (png)

18.8.3 POV-Ray input format (pov)

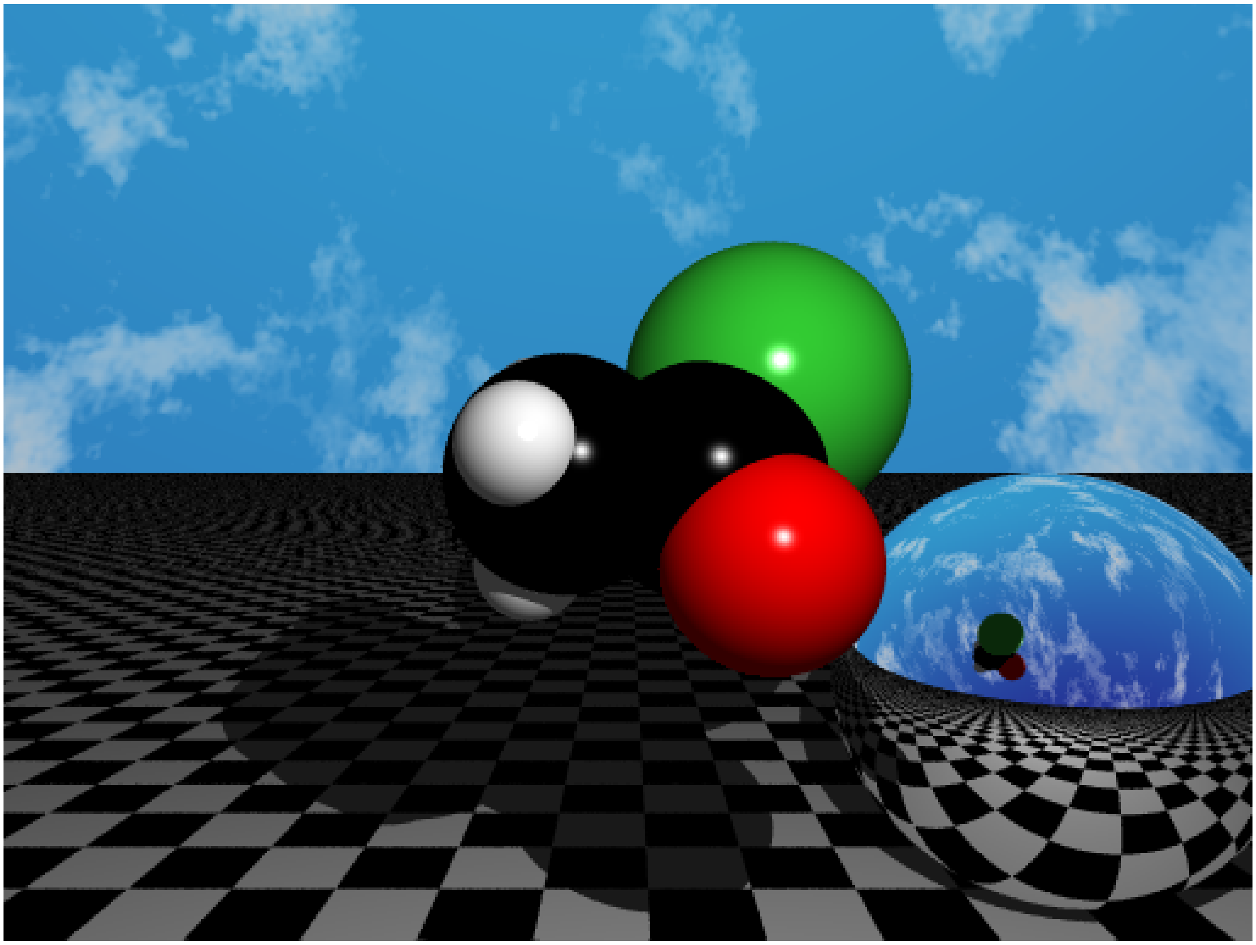
Generate an input file for the open source POV-Ray ray tracer.

The POV-Ray file generated by Open Babel should be considered a starting point for the user to create a rendered image of a molecule. Although care is taken to center the camera on the molecule, the user will probably want to adjust the viewpoint, change the lighting, textures, etc.

The file `babel_povray3.inc` is required to render the povray file generated by Open Babel. This file is included in the Open Babel distribution, and it should be copied into the same directory as the `.pov` file before rendering. By editing the settings in `babel_povray3.inc` it is possible to tune the appearance of the molecule.

For example, the image below was generated by rendering the output from the following command after setting the reflection of non-metal atoms to 0 (line 121 in `babel_povray3.inc`):

```
obabel -:"CC(=O)Cl acid chloride" --gen3d -O chloride.pov -xc -xf -xs -m SPF
```



Note: This is a write-only format.

Write Options

-c *Add a black and white checkerboard*

-f *Add a mirror sphere*

-m <model-type> *BAS (ball-and-stick), SPF (space-fill) or CST (capped sticks)*

The default option is ball-and-stick. To choose space-fill, you would use the following command line:

```
obabel aspirin.mol -O aspirin.pov -xm SPF
```

-s *Add a sky (with clouds)*

-t *Use transparent textures*

18.8.4 Painter format (paint)

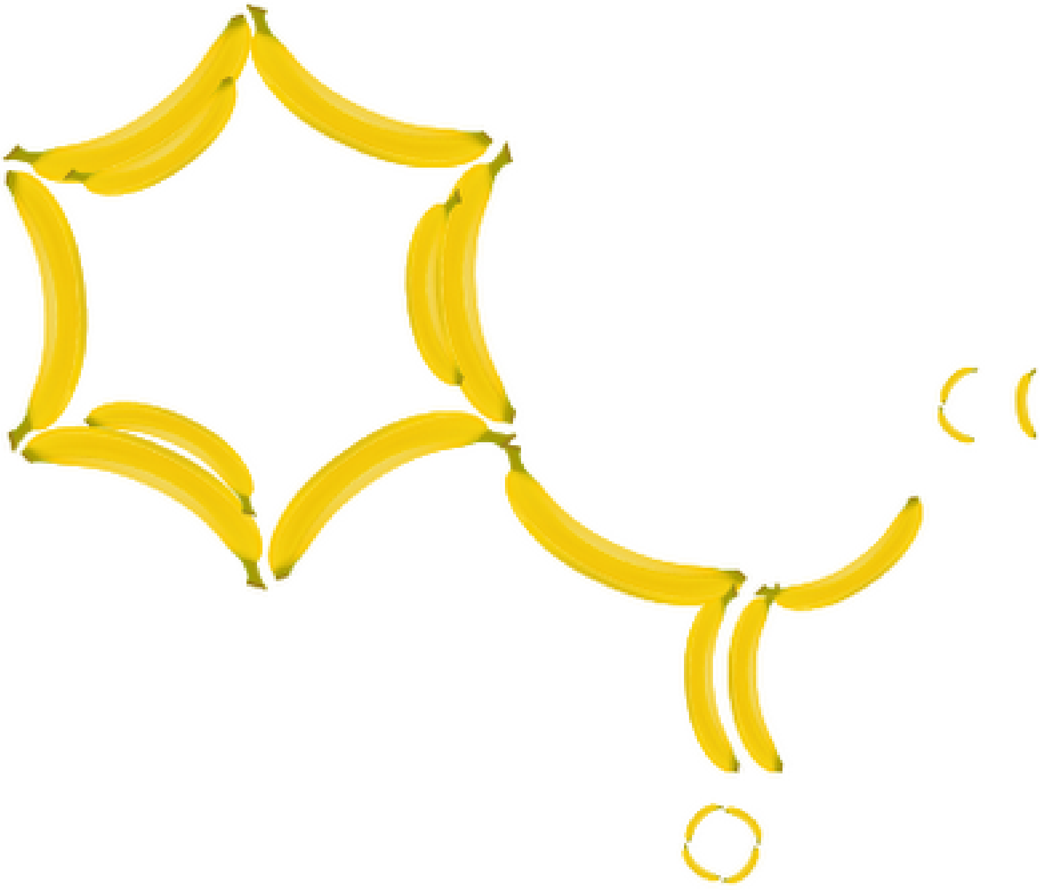
Commands used to generate a 2D depiction of a molecule

This is a utility format that is useful if you want to generate a depiction of a molecule yourself, for example by drawing on a Graphics2D canvas in Java. The format writes out a list of drawing commands as shown in the following example:

```
obabel -:CC(=O)Cl -opaint

NewCanvas 149.3 140.0
SetPenColor 0.0 0.0 0.0 1.0 (rgba)
DrawLine 109.3 100.0 to 74.6 80.0
SetPenColor 0.0 0.0 0.0 1.0 (rgba)
DrawLine 71.6 80.0 to 71.6 53.0
DrawLine 77.6 80.0 to 77.6 53.0
SetPenColor 0.0 0.0 0.0 1.0 (rgba)
DrawLine 74.6 80.0 to 51.3 93.5
SetPenColor 0.4 0.4 0.4 1.0 (rgba)
SetPenColor 0.4 0.4 0.4 1.0 (rgba)
SetPenColor 1.0 0.1 0.1 1.0 (rgba)
SetFontSize 16
SetFontSize 16
SetFontSize 16
DrawText 74.6 40.0 "O"
SetPenColor 0.1 0.9 0.1 1.0 (rgba)
SetFontSize 16
SetFontSize 16
SetFontSize 16
SetFontSize 16
DrawText 40.0 100.0 "Cl"
```

Note that the origin is considered to be in the top left corner.

The following image was drawn using the information in this format as described at http://baoilleach.blogspot.co.uk/2012/04/painting-molecules-your-way-introducing.html:



Note: This is a write-only format.

Write Options

-M *Do not include a margin around the depiction*

18.8.5 SVG 2D depiction (svg)

Scalable Vector Graphics 2D rendering of molecular structure.

When called from commandline or GUI or otherwise via Convert(), single molecules are displayed at a fixed scale, as in normal diagrams, but multiple molecules are displayed in a table which expands to fill the containing element, such as a browser window. When WriteMolecule() is called directly, without going through WriteChemObject, e.g. via OBConversion::Write(), a fixed size image by default 200 x 200px containing a single molecule is written. The size can be specified by the P output option.

Multiple molecules are displayed in a grid of dimensions specified by the `-xr` and `-xc` options (number of rows and columns respectively and `--rows`, `--cols` with babel). When displayed in most modern browsers, like Firefox, there is javascript support for zooming (with the mouse wheel) and panning (by dragging with the left mouse button).

If both `-xr` and `-xc` are specified, they define the maximum number of molecules that are displayed. If only one of them is displayed, then the other is calculated so that ALL the molecules are displayed. If neither are specified, all the molecules are output in an approximately square table.

By default, 2D atom coordinates are generated (using gen2D) unless they are already present. This can be slow with a large number of molecules. (3D coordinates are ignored.) Include `--gen2D` explicitly if you wish any existing 2D coordinates to be recalculated.

Note: This is a write-only format.

Write Options

-u *no element-specific atom coloring*

Use this option to produce a black and white diagram

-U *do not use internally-specified color*

e.g. atom color read from cml or generated by internal code

-b <color> *background color, default white*

e.g `-xb yellow` or `-xb #88ff00` `-xb none` is transparent. Just `-xb` is black with white bonds. The atom symbol colors work with black and white backgrounds, but may not with other colors.

-B <color> *bond color, default black*

e.g `-xB` yellow or `-xB #88ff00`

-C *do not draw terminal C (and attached H) explicitly*

The default is to draw all hetero atoms and terminal C explicitly, together with their attched hydrogens.

-a *draw all carbon atoms*

So propane would display as H3C-CH2-CH3

-d *do not display molecule name*

-s *use asymmetric double bonds*

-t *use thicker lines*

-e *embed molecule as CML*

OpenBabel can read the resulting svg file as a cml file.

-p <num> *px Scale to bond length(single mol only)*

-P <num> *px Single mol in defined size image*

The General option –px # is an alternative to the above.

-c <num> *number of columns in table*

-c *ols<num> number of columns in table(not displayed in GUI)*

-r <num> *number of rows in table*

-r *ows<num> number of rows in table(not displayed in GUI)*

-N <num> *max number objects to be output*

-l *draw grid lines*

-h <condition> *highlight mol if condition is met*

The condition can use descriptors and properties, See documentation on `--filter` option for details. To highlight in a particular color, follow the condition by a color.

-i *add index to each atom*

These indices are those in sd or mol files and correspond to the order of atoms in a SMILES string.

-j *do not embed javascript*

Javascript is not usually embedded if there is only one molecule, but it is if the rows and columns have been specified as 1: `-xr1 -xc1`

-x *omit XML declaration (not displayed in GUI)*

Useful if the output is to be embedded in another xml file.

-X *All atoms are explicitly declared*

Useful if we don't want any extra hydrogens drawn to fill the valence.

-A *display aliases, if present*

This applies to structures which have an alternative, usually shorter, representation already present. This might have been input from an A or S superatom entry in an sd or mol file, or can be generated using the –genalias option. For example:

```
obabel -:"c1cc(C=O)ccc1C(=O)O" -O out.svg
       --genalias -xA
```

would add a aliases COOH and CHO to represent the carboxyl and aldehyde groups and would display them as such in the svg diagram. The aliases which are recognized are in data/superatom.txt, which can be edited.

-S *Ball and stick depiction of molecules*

Depicts the molecules as balls and sticks instead of the normal line style.

Comments

If the input molecule(s) contain explicit hydrogen, you could consider improving the appearance of the diagram by adding an option `-d` to make it implicit. Hydrogen on hetero atoms and on explicitly drawn C is always shown. For example, if input.smi had 10 molecules:

```
obabel input.smi -O out.svg -xb -xC -xe
```

would produce a svg file with a black background, with no explicit terminal carbon, and with an embedded cml representation of each molecule. The structures would be in two rows of four and one row of two.

18.9 2D drawing formats

18.9.1 ChemDraw CDXML format (cdxml)

Minimal support of chemical structure information only.

18.9.2 ChemDraw Connection Table format (ct)

18.9.3 ChemDraw binary format (cdx)

Read only

The whole file is read in one call. Note that a file may contain a mixture of reactions and molecules. With the -ad option, a human-readable representation of the CDX tree structure is output as an OBText object. Use textformat to view it:

```
obabel input.cdx -otext -ad
```

Many reactions in CDX files are not fully specified with reaction data structures, and may not be completely interpreted by this parser.

Note: This is a read-only format.

Read Options

-m	*read molecules only; no reactions*
-d	*output CDX tree to OBText object*
-o	*display only objects in tree output*

18.9.4 Chemical Resource Kit diagram(2D) (crk2d)

18.9.5 Chemtool format (cht)

Note: This is a write-only format.

18.10 3D viewer formats

18.10.1 Ball and Stick format (bs)

18.10.2 Chem3D Cartesian 1 format (c3d1)

18.10.3 Chem3D Cartesian 2 format (c3d2)

18.10.4 Chemical Resource Kit 3D format (crk3d)

18.10.5 Ghemical format (gpr)

Open source molecular modelling

18.10.6 Maestro format (mae, maegz)

File format of Schrödinger Software

18.10.7 Molden format (mold, molden, molf)

Read Options

-b	*no bonds*
-s	*no multiple bonds*

18.10.8 PCModel Format (pcm)

18.10.9 UniChem XYZ format (unixyz)

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.10.10 ViewMol format (vmol)

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.10.11 XCrySDen Structure Format (axsf, xsf)

Note: This is a read-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.10.12 YASARA.org YOB format (yob)

The native YASARA format.

18.11 Kinetics and Thermodynamics formats

18.11.1 ChemKin format (ck)

Read Options

-f <file>	*File with standard thermo data: default therm.dat*
-z	*Use standard thermo only*
-L	*Reactions have labels (Usually optional)*

Write Options

-s	*Simple output: reactions only*
-t	*Do not include species thermo data*
-0	*Omit reactions with zero rates*

18.11.2 Thermo format (tdd, therm)

Reads and writes old-style NASA polynomials in original fixed format

Read Options

-e	*Terminate on "END"*

18.12 Molecular dynamics and docking formats

18.12.1 Amber Prep format (prep)

Note: This is a read-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.12.2 AutoDock PDBQT format (pdbqt)

Reads and writes AutoDock PDBQT (Protein Data Bank, Partial Charge (Q), & Atom Type (T)) format

Note that the torsion tree is by default. Use the `r` write option to prevent this.

Read Options

-b *Disable automatic bonding*

-d *Input file is in dlg (AutoDock docking log) format*

Write Options

-b *Enable automatic bonding*

-r *Output as a rigid molecule (i.e. no branches or torsion tree)*

-c *Combine separate molecular pieces of input into a single rigid molecule (requires "r" option or will have no effect)*

-s *Output as a flexible residue*

-p *Preserve atom indices from input file (default is to renumber atoms sequentially)*

-h *Preserve hydrogens*

-n *Preserve atom names*

18.12.3 DL-POLY CONFIG (CONFIG)

18.12.4 DL-POLY HISTORY (HISTORY)

Note: This is a read-only format.

18.12.5 Dock 3.5 Box format (box)

18.12.6 GRO format (gro)

This is GRO file format as used in Gromacs.

Right now there is only limited support for element perception. It works for elements with one letter symbols if the atomtype starts with the same letter.

Read Options

-s *Consider single bonds only*

-b *Disable bonding entierly*

18.12.7 GROMOS96 format (gr96)

Note: This is a write-only format.

Write Options

-n	*output nm (not Angstroms)*

18.12.8 LPMD format (lpmd)

Read and write LPMD's atomic configuration file

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

Write Options

-f <num>	*Indicate the level of the output file: 0 (default), 1 or 2.*
-m <num>	*Indicate the mode for level 2 output files* 0 (default) is for accelerations and 1 for forces
-c <vectorcells>	*Set the cell vectors if not present* Example: `-xc 10.0,0,0,0.0,10.0,0.0,0.0,0.0,20.0`
-e	*Add the charge to the output file*

18.12.9 MDFF format (CONTFF, MDFF, POSFF)

The format used in the POSFF and CONTFF files used by MDFF

POSFF and CONTFF are read to obtain information from MDFF calculations. The program will try to read the IONS.POT file if the name of the input file is POSFF or CONTFF.

Write Options

-w	*Sort atoms by atomic number*
-u <elementlist>	*Sort atoms by list of element symbols provided in comma-separated string w/o spaces*
-i	*Write IONS.POT file*

18.12.10 MacroModel format (mmd, mmod)

18.12.11 SIESTA format (siesta)

The format used by SIESTA (Spanish Initiative for Electronic Simulations with Thousands of Atoms).

Note: This is a read-only format.

18.12.12 The LAMMPS data format (lmpdat)

LAMMPS is a classical molecular dynamics code, and an acronym for Large-scale Atomic/Molecular Massively Parallel Simulator.

Note: This is a write-only format.

Write Options

-q <water-model> *Set atomic charges for water.*

There are two options: SPC (default) or SPCE

-d <length> *Set the length of the boundary box around the molecule.*

The default is to make a cube around the molecule adding 50% to the most positive and negative cartesian coordinate.

18.12.13 Tinker XYZ format (txyz)

The cartesian XYZ file format used by the molecular mechanics package TINKER.

By default, the MM2 atom types are used for writing files but MM3 atom types are provided as an option. Another option provides the ability to take the atom type from the atom class (e.g. as used in SMILES, or set via the API).

Read Options

-s *Generate single bonds only*

Write Options

-m *Write an input file for the CNDO/INDO program.*

-c *Write atom types using custom atom classes, if available*

-3 *Write atom types for the MM3 forcefield.*

18.12.14 XTC format (xtc)

A portable format for trajectories (gromacs)

Note: This is a read-only format.

18.13 Volume data formats

18.13.1 ADF TAPE41 format (t41)

Currently the ADF Tape41 support reads grids from TAPE41 text files. To generate an ASCII version from the default binary, use the dmpkf program.

Note: This is a read-only format.

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.13.2 Gaussian cube format (cub, cube)

A grid format for volume data used by Gaussian

Open Babel supports reading and writing Gaussian cubes, including multiple grids in one file.

Read Options

-b	*no bonds*
-s	*no multiple bonds*

18.13.3 OpenDX cube format for APBS (dx)

A volume data format for IBM's Open Source visualization software

The OpenDX support is currently designed to read the OpenDX cube files from APBS.

18.13.4 Point cloud on VDW surface (pointcloud)

Generates a point cloud on the VDW surface around the molecule

The surface location is calculated by adding the probe atom radius (if specified) to the Van der Waal radius of the particular atom multipled by the specified multiple (1.0 if unspecified).Output is a list of {x,y,z} tuples in Angstrom. Alternatively, if the `x` option is specified, the *XYZ cartesian coordinates format (xyz)* is used instead.

Note: This is a write-only format.

Write Options

-r <radii> *create a surface for each VDS radius (default 1.0)*

A comma-separated list of VDW radius multiples

-d <densities> *for each surface, specify the point density (default 1.0 Angstrom^2)*

A comma-separated list of densities

-p <radius> *radius of the probe atom in Angstrom (default 0.0)*

-x *output in xyz format*

18.13.5 STL 3D-printing format (stl)

The STereoLithography format developed by 3D Systems

Note: This is a write-only format.

Write Options

-p <radius> *radius for probe particle (default 0.0 A)*

-s <scale> *scale-factor for VDW radius (default 1.0 A)*

-c *add CPK colours*

18.14 JSON formats

18.14.1 ChemDoodle JSON (cdjson)

The native way to present data to the ChemDoodle Web Components

Read Options

-c <num> *coordinate multiplier (default: 20)*

Write Options

-c <num> *coordinate multiplier (default: 20)*

-m *minified output formatting, with no line breaks or indents*

-v *verbose output (include default values)*

-w *use wedge/hash bonds from input instead of perceived stereochemistry*

18.14.2 PubChem JSON (pcjson)

The JSON format returned by the PubChem PUG REST service

The data contained in this format closely resembles PubChem's internal data structure.

Read Options

-s	*disable stereo perception and just read stereo information from input*

Write Options

-m	*minified output, with no line breaks or indents*
-w	*use bond styles from input instead of perceived stereochemistry*

18.15 Miscellaneous formats

18.15.1 M.F. Sanner's MSMS input format (msms)

Generates input to the MSMS (Michael Sanner Molecular Surface) program to compute solvent surfaces.

Note: This is a write-only format.

Write Options

-a	*output atom names*

18.16 Biological data formats

18.16.1 FASTA format (fa, fasta, fsa)

A file format used to exchange information between genetic sequence databases

Read Options

-1	*Output single-stranded DNA*
-t <turns>	*Use the specified number of base pairs per turn (e.g., 10)*
-s	*Output single bonds only*
-b	*Disable bonding entirely*

Write Options

-n	*Omit title and comments*

18.16.2 PQR format (pqr)

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.17 Obscure formats

18.17.1 Alchemy format (alc)

18.17.2 CCC format (ccc)

Note: This is a read-only format.

18.17.3 Feature format (feat)

Read Options

-s	*Output single bonds only*
-b	*Disable bonding entirely*

18.17.4 SMILES FIX format (fix)

Note: This is a write-only format.

18.17.5 XED format (xed)

Note: This is a write-only format.

Chapter 19

Descriptors

19.1 Numerical descriptors

Number of atoms (atoms)

Add or remove hydrogens to count total or heavy atoms SMARTS: *

Number of bonds (bonds)

Add or remove hydrogens to count total or bonds between heavy atoms SMARTS: *~*

Number of Hydrogen Bond Donors (JoelLib) (HBD)

SMARTS: [!#6;!H0]

Number of Hydrogen Bond Acceptors 1 (JoelLib) (HBA1)

Identification of Biological Activity Profiles Using Substructural Analysis and Genetic Algorithms – Gillet, Willett and Bradshaw, U. of Sheffield and Glaxo Wellcome. Presented at Random & Rational: Drug Discovery via Rational Design and Combinitorial Chemistry, Strategic Research Institute, Princeton NJ, Sept. 1995 SMARTS: [$([!#6;+0]);!$([F,Cl,Br,I]);!$([o,s,nX3]);!$([Nv5,Pv5,Sv4,Sv6])]

Number of Hydrogen Bond Acceptors 2 (JoelLib) (HBA2)

SMARTS: [$([$([#8,#16]);!$(*=N~O);!$(*~N=O);X1,X2]),$([#7;v3;!$([nH]);!$(*(-a)-a)])]

Number of Fluorine Atoms (nF)

SMARTS: F

octanol/water partition coefficient (logP)

Datafile: logp.txt

Molecular Weight filter (MW)

Number of triple bonds (tbonds)

SMARTS: *#*

molar refractivity (MR)

Datafile: mr.txt

Number of aromatic bonds (abonds)

SMARTS: *:*

Number of single bonds (sbonds)

SMARTS: *-*

Number of double bonds (dbonds)

SMARTS: *=*

topological polar surface area (TPSA)

Datafile: psa.txt

Rotatable bonds filter (rotors)

Melting point (MP)

This is a melting point descriptor developed by Andy Lang. For details see: http://onschallenge.wikispaces.com/MeltingPointModel011 Datafile: mpC.txt

19.2 Textual descriptors

Canonical SMILES (cansmi)

Canonical SMILES without isotopes or stereo (cansmiNS)

IUPAC InChI identifier (InChI)

InChIKey (InChIKey)

Chemical formula (formula)

For comparing a molecule’s title (title)

19.3 Descriptors for filtering

Lipinski Rule of Five (L5)

HBD<5 HBA1<10 MW<500 logP<5

SMARTS filter (smarts)

SMARTS filter (s)

Chapter 20

Charge models

Insert text here.

20.1 Cheminformatics charge models

Assign Gasteiger-Marsili sigma partial charges (gasteiger)

Assign MMFF94 partial charges (mmff94)

20.2 Special charge models

Assign Electronegativity Equilization Method (EEM) atomic partial charges (eem)

Assign QEq (charge equilibration) partial charges (Rappe and Goddard, 1991) (qeq)

Assign QTPIE (charge transfer, polarization and equilibration) partial charges (Chen and Martinez, 2007) (qtpie)

Chapter 21

Release Notes

21.1 Open Babel 3.0.0

Released on 2019-10-10.

This is a major release. It fixes some long-standing issues affecting performance in terms of chemical accuracy and speed, and all users are recommended to upgrade. It also removes deprecated components and breaks the API in a few places. For information on migrating from the previous version, please see *Updating to Open Babel 3.0 from 2.x*.

21.1.1 Notable changes

- The babel program has been removed, and the replacement obabel should be used instead. The obabel program fixes some flaws with the original babel (not least that the user could accidentally overwrite an input file) and so has been preferred for many years.
- The Python bindings are now accessed via “from openbabel import pybel” or “from openbabel import openbabel”.
- Under the hood, the code for handling implicit hydrogens and kekulization has been entirely replaced in order to address problems with the original approach that had resulted in multiple bug reports over the years. As well as being accurate, the new approach is much faster.
- The speed of reading and writing SMILES has been improved by more than 50-fold.
- A faster and more accurate fragment-based 3D coordinate generation code has been added, part of Google Summer of Code 2018 and 2019, detailed in **J. Cheminf.* (2019) **11**, Art. 49.<https://doi.org/10.1186/s13321-019-0372-5>*
- New functionality in the API:
  - A new class for managing reactions stored as OBMols (OBReactionFacade)
  - A new method to copy part of an OBMol into a new OBMol (OBMol::CopySubstructure)

21.1.2 New file formats

- Add basic support for RInChI (Reaction InChI) (by baoilleach, PR#1667)
- Added basic ADF Band and ADF DFTB readers (by psavery, PR#1793)

- Add support for COF format (Culgi object file) plus tests (by pbecherer, PR#1944)
- Add maeparser support to openbabel (by lorton, PR#1993)

21.1.3 New file format capabilities and options

- Improve svg ball and stick (by ghutchis, PR#360)
- Add an option to the canonical SMILES format to specify the timeout. (by timvdm, PR#386)
- Allow to set space group origin in PDB CRYST1 section (by afonari, PR#1558)
- Parse _space_group_symop_operation_xyz in mmcif (by afonari, PR#1578)
- Improve performance of SMILES parser (by baoilleach, PR#1589)
- Handle undervalent atoms and radicals in Mol files and Smiles (by baoilleach, PR#1626)
- Add support for agents to RXN file format (by baoilleach, PR#1656)
- Allow RSMI format to read partial reactions (by baoilleach, PR#1660)
- Add support for %(NNN) notation for SMILES ring closures (by baoilleach, PR#1677)
- By default, don’t perceive stereo when reading SMILES, but have the option (by baoilleach, PR#1696)
- Speed up the SMILES writer (by baoilleach, PR#1699)
- Faster SMILES: Replace std::endl by “n” (by baoilleach, PR#1706)
- Speed up SMILES writer by replacement of SSSR in SMILES writer with a bounded BFS (by baoilleach, PR#1715)
- Speed up SMILES reading: don’t pre-scan the SMILES string for illegal characters (by baoilleach, PR#1716)
- Minor speedup in SMILES: avoid repeated calls to IsOption by caching some options (by baoilleach, PR#1718)
- Read reaction map from ChemDraw CDX files (by CamAnNguyen, PR#1720)
- Two minor SMILES speed improvements (by baoilleach, PR#1725)
- Speed up SMILES reading: Moved more inside the switch statement for SMILES parsing (by baoilleach, PR#1727)
- Speed up SMILES reading: In the SMILES reader, avoid allocating a BUFSIZE buffer, and the associated string copy (by baoilleach, PR#1728)
- Speed up SMILES writing: Make generation of the SMILES atom order vector optional (by baoilleach, PR#1712)
- Add support for using atom classes as Tinker atom types. (by ghutchis, PR#1734)
- Gaussformat reading electrostatic potentials (by mmghahremanpour, PR#1748)
- Reading Exact Polairzability from Gaussian log file (by mmghahremanpour, PR#1751)
- Gaussformat reading multiple charge models (by mmghahremanpour, PR#1752)
- Write atom occupancy (if present) to PDB (by afonari, PR#1799)
- Update reaction support in ChemDraw (by baoilleach, PR#1878)
- ADF DFTB: Add new detection string for ADF 2018 (by psavery, PR#1888)
- Update Gaussian format (by e-kwsm, PR#1969)
- Update URLs of specification of gromacs (by e-kwsm, PR#1974)

- Update URL of specification of MDL MOL (by e-kwsm, PR#1980)
- Add SMILES support for elements specified by 3-digit number, e.g. [#101] (by baoilleach, PR#1997)

21.1.4 Other new features and improvements

- Include original when there are zero rotatable bonds in confab (by cowsandmilk, PR#370)
- Improve thread safety for global objects (by baoilleach, PR#381)
- Change the OBAromTyper from using SMARTS patterns to a switch statement (rebased) (by baoilleach, PR#1545)
- Keep count of implicit hydrogens instead of inferring them (by baoilleach, PR#1576)
- Obthermo update patch (by mmghahremanpour, PR#1598)
- Improve performance of element handling (by baoilleach, PR#1601)
- Implement the Daylight aromaticity model as described by John Mayfield (by baoilleach, PR#1638)
- Allow multiple agents in OBReaction (by baoilleach, PR#1640)
- Clarify python examples (by theavey, PR#1657)
- Add support for wrapping GetRGB() call to return r, g, b params. (by ghutchis, PR#1670)
- Adding missing manpages (by merkys, PR#1678)
- Expose obfunctions api through python bindings (by cstein, PR#1697)
- Avoid logging messages that are taking time (by baoilleach, PR#1714)
- warning/error messages for fastindex when the structure file is compressed (by adalke, PR#1733)
- Refactor atom class to being data on an atom rather than on a molecule (by baoilleach, PR#1741)
- Add Molecule.make2D function (by eloyfelix, PR#1765)
- Change the behavior of OBMol.Separate so that it preserves atom order (by baoilleach, PR#1773)
- When calling OBMol.Separate, preserve whether aromaticity has been perceived (by baoilleach, PR#1800)
- Add OBMol::CopySubstructure (by baoilleach, PR#1811)
- Add OBMol::SetChainsPerceived(false) (by baoilleach, PR#1813)
- Add stereo + obfunctions + kekulize to ruby binding (by CamAnNguyen, PR#1824)
- Generate useful error messages if plugins can't be found. (by dkoes, PR#1826)
- Allow public access to retrieve gradients (by ghutchis, PR#1833)
- Re-enable vector.clear() to allow wrapped std::vectors to be reused (by baoilleach, PR#1834)
- Implement reaction handling as part of OBMol (by baoilleach, PR#1836)
- Added rotors as a descriptor/filter. (by ghutchis, PR#1846)
- Keep aromaticity in EndModify() (by baoilleach, PR#1847)
- Fragment-based coordinate generation (by n-yoshikawa, PR#1850)
- Rebuild OBMM tool for interactive MM optimization (by ghutchis, PR#1873)
- Update fragment based builder (by n-yoshikawa, PR#1931)
- Refactor python bindings so that openbabel.py and pybel.py are within an openbabel folder (by baoilleach, PR#1946)

- Tidy setting/unsetting of molecule perception flags (by baoilleach, PR#1951)
- Remove outdated stereo code (by baoilleach, PR#1967)
- Remove OBBond::GetBO() and SetBO() (by baoilleach, PR#1953)
- Remove OBRandom from the public API (by baoilleach, PR#1954)
- Remove miscellanous headers from mol.h, atom.h and bond.h (by baoilleach, PR#1958)
- enhancements to obrms to support optimization of pose alignment (by dkoes, PR#1961)
- Remove GetGenericValueDef from OBGenericData (by baoilleach, PR#1964)
- Remove low-hanging deprecated methods (by baoilleach, PR#1968)
- Improve python script (by e-kwsm, PR#1970)
- Make *pybel.Outputfile* compatible with *with* statment (by yishutu, PR#1971)
- Obrms enhancement (by dkoes, PR#1978)
- Move to a single function for setting/unsetting bond and atom flags (by baoilleach, PR#1965)
- Rename/add valence and degree methods (by baoilleach, PR#1975)
- Do not stoke around the (svg) text (by Artoria2e5, PR#2012)
- Add a warning message when both -p and -h options are set (by yishutu, PR#2031)
- "Bye bye babel" - remove the babel binary (by baoilleach, PR#1976)
- Add force field support for dielectric constants in charge terms. (by ghutchis, PR#2022)

21.1.5 Development/Build/Install Improvements

- Change default build type to RELEASE and add -O3 switch (by baoilleach, PR#352)
- Add a default issue template for Open Babel - Suggestions welcome (by ghutchis, PR#383)
- Compile position independent code for shared libraries. (by susilehtola, PR#1575)
- Introduce std:isnan for older versions of MSVC (by mwojcikowski, PR#1586)
- Prepend to LD_LIBRARY_PATH instead of overwrite (by barrymoo, PR#1588)
- Changes needed to compile with C++17 (by arkose, PR#1619)
- Compiler version parsing and comparison from CMake 2.8 (by cowsandmilk, PR#1630)
- Create CODE_OF_CONDUCT.md (by ghutchis, PR#1671)
- Clarify option needed to generate SWIG bindings. (by jeffjanes, PR#1686)
- Correct spelling of file name for Perl bindings (by jeffjanes, PR#1687)
- In the Python bindings, avoid adding methods from the iterated object to the iterator itself (by baoilleach, PR#1729)
- Ensure portability to ARM platforms (by baoilleach, PR#1744)
- Switch to rapidjson library for JSON parsing/writing (by mcs07, PR#1776)
- Fix linking of python bindings on Mac (by mcs07, PR#1807)
- Using pillow instead of PIL (by hille721, PR#1822)
- Ignore compile warnings on inchi directory. (by ghutchis, PR#1864)

- Compile project in Cygwin without xtcformat (by bbucior, PR#1894)
- Hyperlink DOIs to preferred resolver (by katrinleinweber, PR#1909)
- For Travis builds, include output for build failures (by baoilleach, PR#1959)
- Add __init__.py to gitignore (by yishutu, PR#1972)
- Ignore in-source installation (by RMeli, PR#2027)
- Add a GitHub funding link to the open collective page. (by ghutchis, PR#2042)

21.1.6 Bug Fixes

- Fix for missing ZLIB on win32 (by philthiel, PR#357)
- Depict headers were missing in the installation (by tgaudin, PR#359)
- Avoid IndexError for plugins with empty names (by langner, PR#361)
- Fixed a few errors in space-groups.txt (by psavery, PR#367)
- SF #909 - Fix segfault when ReadMolecule() called with PubChem document but file extension was generic .xml (by derekharmon, PR#369)
- Preserve triple bond when reading SMILES with a triple bond in an aromatic ring (by baoilleach, PR#371)
- Fix bug #368: Python3.6 openbabel: No module named 'DLFCN' (by hseara, PR#372)
- Fastsearch 64 fix (by dkoes, PR#1546)
- Don't try to install aromatic.txt as it is no longer present (by baoilleach, PR#1547)
- Make sure to add conformers *after* performing bond perception. (by ghutchis, PR#1549)
- Set default coordinates before doing bond perception. (by ghutchis, PR#1550)
- Ignore some non-functioning python SWIG bindings. (by djhogan, PR#1554)
- Remove delete statement. (by djhogan, PR#1556)
- Link libinchi with math library (by nsoranzo, PR#1564)
- Fix segfault in OBMol::GetSpacedFormula (by bbucior, PR#1565)
- Fix regression + minor cppcheck report (by serval2412, PR#1567)
- Convert tabs to spaces in testpdbformat.py (by adamjstewart, PR#1568)
- cppcheck: Condition '1==0' is always false (by serval2412, PR#1572)
- UFF: Fix conversion constant (by aandi, PR#1579)
- Remove the change in resonance structure from the vinylogous carboxylic acid pH model (by kyle-roberts-arzeda, PR#1580)
- Fix wedge/hash in cyclopropyl (by fredrikw, PR#1582)
- Fix multifragment depiction (by fredrikw, PR#1585)
- Fix wrong spin multiplicity assignment (by nakatamaho, PR#1592)
- Change silicon to correct MM3 atom type (by keipertk, PR#1593)
- Fix pubchem JSON handling of enum types as ints (by mcs07, PR#1596)
- Correct MM3 carboxyl oxygen atom type definition (by keipertk, PR#1599)
- Fix for calculating implicit H count when reading SMILES (by baoilleach, PR#1606)

- Fix some small misspellings in the csharp bindings (by cmanion, PR#1608)
- Tweak the handling of implicit Hs when reading SMILES (by baoilleach, PR#1609)
- Fix underflow causing a noticeable delay when e.g. writing a molfile (by baoilleach, PR#1610)
- Fix install regression with element data (by bbucior, PR#1617)
- Added some missing formats to the static build (by psavery, PR#1622)
- In SiestaFormat, print warnings to cerr (by psavery, PR#1623)
- For SIESTA format, use obErrorLog instead of cerr (by psavery, PR#1627)
- Correct the spelling of the Frerejacque number in a comment (by baoilleach, PR#1629)
- Lowercase second element letter in PDB and test (by cowsandmilk, PR#1631)
- Remove erroneous -1 in switch statement (by baoilleach, PR#1632)
- Make sure to handle molecular total charge by default for keywords (by ghutchis, PR#1634)
- Added fix for OBMolAtomBFSIter in Python3 (by oititov, PR#1637)
- space-groups.txt: correct Hall symbol for C -4 2 b (by wojdyr, PR#1645)
- Reset path to empty in kekulization code (potential segfault) (by baoilleach, PR#1650)
- Correct handling of stereo when writing InChIs (by baoilleach, PR#1652)
- ECFP Fixup (by johnmay, PR#1653)
- Fix "folding" for fingerprints to larger bit sizes - #1654. (by ghutchis, PR#1658)
- Fix reading atom symbols from XSF file (by sencer, PR#1663)
- Minor fixes in the nwchem format reader (by xomachine, PR#1666)
- use isinstance to test if filename is bytes (by cowsandmilk, PR#1673)
- Fix bug found due to MSVC warning (by baoilleach, PR#1674)
- Fix MSVC warning about unused variable (by baoilleach, PR#1675)
- Correct handling of atom maps (by baoilleach, PR#1698)
- Fix #1701 - a GCC compiler error (by baoilleach, PR#1704)
- Remove some audit messages (by baoilleach, PR#1707)
- Fix bug when copying stereo during obmol += obmolB (by baoilleach, PR#1719)
- Fix uninitialized read in kekulize.cpp found by Dr Memory. (by baoilleach, PR#1721)
- Fixes for ring closure parsing (by baoilleach, PR#1723)
- Make sure that OBAtom::IsInRing always triggers ring perception if not set as perceived (by baoilleach, PR#1724)
- Fix code error found from @baoilleach compiler warnings (by ghutchis, PR#1736)
- Fix Python3 compatibility (by ghutchis, PR#1737)
- Fix ChemDraw CDX incremental value (by CamAnNguyen, PR#1743)
- Fix error in VASPformat found by static code analysis (by baoilleach, PR#1745)
- Fix for 1731. Store atom classes in CML atomids by appending _ATOMCLASS. (by baoilleach, PR#1746)
- Fix GCC warnings (by baoilleach, PR#1747)

- Fix warning in fastsearch substructure fingerprint screen (by baoilleach, PR#1749)
- Fix #1684 - string comparison does not work with numeric sd titles (by cowsandmilk, PR#1750)
- Fixing minor things for reading ESP from log files (by mmghahremanpour, PR#1753)
- Fix #1569 - OB 2.4.1 loses the second molecule in a HIN file (by yishutu, PR#1755)
- Fix TESTDIR definition to allow space in path (by mcs07, PR#1757)
- Fix regression. Ensure that asterisk is unbracketed when writing a SMILES string (by baoilleach, PR#1759)
- Fix MSVC warning about type conversion (by baoilleach, PR#1762)
- Fix SMILES parsing fuzz test failures from AFL (by baoilleach, PR#1770)
- Fix warning about size_t versus int cast (by baoilleach, PR#1771)
- A small improvement of a bugfix solving segfault when reading GAMESS output with vibrations (by boryszef, PR#1772)
- In the Python bindings, reset the DL open flags after importing _openbabel (by baoilleach, PR#1775)
- fix cdxml stereo bonds (by JasonYCHuang, PR#1777)
- Install obabel target if using static build (by torcolvin, PR#1779)
- Fix #1769 by correctly handling the mass difference field in MDL mol files (by baoilleach, PR#1784)
- Kekulize hypervalent aromatic N and S (by baoilleach, PR#1787)
- Pdbqt fix (by dkoes, PR#1790)
- Raise a warning when coordinate is NaN (by n-yoshikawa, PR#1792)
- Use the InChI values for the average atomic mass when reading/writing isotopes (by baoilleach, PR#1795)
- Fix compile failure after recent Molden commit (by baoilleach, PR#1796)
- Fix segfault due to running off the start of an iterator in PDBQT format (by baoilleach, PR#1797)
- Fix#1768: Segfault upon reading GAMESS outputs of DFTB3 calculations (by serval2412, PR#1798)
- Always ensure hybridization (by ghutchis, PR#1801)
- Fix #1786 by changing the return value of OBResidue::GetNum() (by baoilleach, PR#1804)
- Apply fixes from Benoit Leblanc to address int/double type warnings. (by baoilleach, PR#1806)
- Fix#1607: check dynamic cast return (by serval2412, PR#1815)
- Fixes #1282: check format input is provided (by serval2412, PR#1818)
- Fix#1331: avoid crash with Q-Chem fragment (by serval2412, PR#1820)
- Set default to read CIFs with specified coordinates, no wrapping. (by ghutchis, PR#1823)
- Fix#1056: remove a debug output (by serval2412, PR#1825)
- Get ECFP working (by baoilleach, PR#1829)
- Fix cdxml upside down format (by JasonYCHuang, PR#1831)
- Fix to CopySubstructure found when running over ChEMBL (by baoilleach, PR#1832)
- Fix#192: parse and use '-a' flag for obrotate (by serval2412, PR#1835)
- Ensure carbonyl groups are checked at both 0 and 180. (by ghutchis, PR#1845)
- Ensure that the check for OBBond::IsInRing obeys the OBMol perception flags (by baoilleach, PR#1848)

- Simplify/fix behavior of OBAtom::GetResidue so that it behaves like other lazy properties (by baoilleach, PR#1849)
- Fixes #1851: check some limits when converting smi to sdf using –gen2D (by serval2412, PR#1852)
- Modify cleaning blank line behaviors (by yishutu, PR#1855)
- Ring membership of atoms and bonds was not being reset during perception (by baoilleach, PR#1856)
- Update qeq.txt (by mkrykunov, PR#1882)
- Support lone pair stereo on nitrogen as well as sulfur (by baoilleach, PR#1885)
- Changed indexing of fragments, should fix #1889 (by fredrikw, PR#1890)
- Avoid out-of-range access in OBMolBondBFSIter (by baoilleach, PR#1892)
- Fix OBChemTsfm wrapping of implicit H counts (by baoilleach, PR#1896)
- Updated the coordinate generation from templates. (by fredrikw, PR#1902)
- Fix incorrect use of *memcpy*. (by sunoru, PR#1908)
- Add SetChainsPerceived() after EndModify() in formats that add residues (by baoilleach, PR#1914)
- define isfinite removed. (by orex, PR#1928)
- Teach the isomorphism mapper to respect atom identity (by johnmay, PR#1939)
- Fix memory leak in OBSmartsPattern::Init() (by n-yoshikawa, PR#1945)
- Address CMake build warning about policy CMP0005 being set to OLD (by baoilleach, PR#1948)
- Fix clang warning about in-class init of a non-static data member (by baoilleach, PR#1949)
- Update bindings for changes to headers (by baoilleach, PR#1963)
- Fix randomly failing Python gradient test (by baoilleach, PR#1966)
- Exit with non-zero if an error occurs (by e-kwsm, PR#1973)
- Avoid non-finite bond vectors (by dkoes, PR#1981)
- Include babelconfig in vector3.h (by dkoes, PR#1985)
- Fix #1987: CMake failing at FindRapidJSON (by RMeli, PR#1988)
- fpsformat.cpp: compile bugfix header added. (by orex, PR#1991)
- Address Ubuntu bug in defining python install dir (by dkoes, PR#1992)
- PDB and PDBQT Insertion Code Fixes (by RMeli, PR#1998)
- Make pybel compatible with #1975 (by yishutu, PR#2005)
- H vector fix (by dkoes, PR#2010)
- Change forcefield.cpp so that steepest descent and conjugate gradient update maxgrad (by PeaWagon, PR#2017)
- Update coordinates in the fast option of obabel (by n-yoshikawa, PR#2026)
- Update the CSharp bindings (by baoilleach, PR#2032)
- Don’t make kekule SMILES the default in the GUI (by baoilleach, PR#2039)
- Bumping the major version requires more changes throughout the library. (by baoilleach, PR#2036)
- Fix reading of uninitialized data. (by dkoes, PR#2038)
- Remove minor version from some names (by baoilleach, PR#2040)

- Fixed alias expansion for files with multiple aliases (by fredrikw, PR#2035)
- Update doc (by e-kwsm, PR#1979)
- Fix compilation with GCC 4.8 (standard compiler on CentOS 7.5) (by baoilleach, PR#2047)
- Some tests (by dkoes, PR#2008)

21.1.7 Cast of contributors

aandi, adalke (Andrew Dalke), adamjstewart (Adam J. Stewart), afonari (Alexandr Fonari), artoria2e5 (Mingye Wang), baoilleach (Noel O'Boyle), barrymoo (Barry Moore), bbucior (Ben Bucior), boryszef (Borys Szefczyk), camannguyen (An Nguyen), cmanion (Charles A. Manion), cowsandmilk (David Hall), cstein (Casper Steinmann), derekharmon (Derek Harmon), djhogan (Daniel Hogan), dkoes (David Koes), e-kwsm (Eisuke Kawashima), eloyfelix (Eloy Felix), fredrikw (Fredrik Wallner), ghutchis (Geoff Hutchison), hille721 (Christoph Hille), hseara (Hector Martinez-Seara), jasonychuang (Jason Huang), jeffjanes (Jeff Janes), johnmay (John Mayfield), katrinleinweber (Katrin Leinweber), keipertk (Kristopher Keipert), kyle-roberts-arzeda, langner (Karol M. Langner), lorton (Pat Lorton), mcs07 (Matt Swain), merkys (Andrius Merkys), mkrykunov, mmghahremanpour (Mohammad Ghahremanpour), mwojcikowski (Maciej Wójcikowski), n-yoshikawa (Naruki Yoshikawa), nakatamaho (Nakata Maho), nsoranzo (Nicola Soranzo), oititov (Titov Oleg), orex (Kirill Okhotnikov), pbecherer (Paul Becherer), peawagon (Jen), philthiel (Philipp Thiel), psavery (Patrick Avery), rmeli (Rocco Meli), serval2412 (Julien Nabet), sunoru, susilehtola (Susi Lehtola), tgaudin (Théophile Gaudin), theavey (Thomas Heavey), timvdm (Tim Vandermeersch), torcolvin (Tor Colvin), wojdyr (Marcin Wojdyr), xomachine (Dmitriy Fomichev), yishutu (Yi-Shu Tu)

21.2 Open Babel 2.4.0

Released on 2016-09-21.

Note that this release deprecates the babel executable in favor of obabel. A future release will remove babel entirely. For information on the differences, please see http://openbabel.org/docs/current/Command-line_tools/babel.html.

21.2.1 New file formats

- DALTON output files (read only) and DALTON input files (read/write) (Casper Steinmann)
- JSON format used by ChemDoodle (read/write) (Matt Swain)
- JSON format used by PubChem (read/write) (Matt Swain)
- LPMD's atomic configuration file (read/write) (Joaquin Peralta)
- The format used by the CONTFF and POSFF files in MDFF (read/write) (Kirill Okhotnikov)
- ORCA output files (read only) and ORCA input files (write only) (Dagmar Lenk)
- ORCA-AICCM's extended XYZ format (read/write) (Dagmar Lenk)
- Painter format for custom 2D depictions (write only) (Noel O'Boyle)
- Siesta output files (read only) (Patrick Avery)
- Smiley parser for parsing SMILES according to the OpenSMILES specification (read only) (Tim Vandermeersch)
- STL 3D-printing format (write only) (Matt Harvey)
- Turbomole AOFORCE output (read only) (Mathias Laurin)
- A representation of the VDW surface as a point cloud (write only) (Matt Harvey)

21.2.2 New file format capabilities and options

- AutoDock PDBQT: Options to preserve hydrogens and/or atom names (Matt Harvey)
- CAR: Improved space group support in .car files (kartlee)
- CDXML: Read/write isotopes (Roger Sayle)
- CIF: Extract charges (Kirill Okhotnikov)
- CIF: Improved support for space-groups and symmetries (Alexandr Fonari)
- DL_Poly: Cell information is now read (Kirill Okhotnikov)
- Gaussian FCHK: Parse alpha and beta orbitals (Geoff Hutchison)
- Gaussian out: Extract true enthalpy of formation, quadrupole, polarizability tensor, electrostatic potential fitting points and potential values, and more (David van der Spoel)
- MDL Mol: Read in atom class information by default and optionally write it out (Roger Sayle)
- MDL Mol: Support added for ZBO, ZCH and HYD extensions (Matt Swain)
- MDL Mol: Implement the MDL valence model on reading (Roger Sayle)
- MDL SDF: Option to write out an ASCII depiction as a property (Noel O'Boyle)
- mmCIF: Improved mmCIF reading (Patrick Fuller)
- mmCIF: Support for atom occupancy and atom_type (Kirill Okhotnikov)
- Mol2: Option to read UCSF Dock scores (Maciej Wójcikowski)
- MOPAC: Read z-matrix data and parse (and prefer) ESP charges (Geoff Hutchison)
- NWChem: Support sequential calculations by optionally overwriting earlier ones (Dmitriy Fomichev)
- NWChem: Extract info on MEP(IRC), NEB and quadrupole moments (Dmitriy Fomichev)
- PDB: Read/write PDB insertion codes (Steffen Möller)
- PNG: Options to crop the margin, and control the background and bond colors (Fredrik Wallner)
- PQR: Use a stored atom radius (if present) in preference to the generic element radius (Zhixiong Zhao)
- PWSCF: Extend parsing of lattice vectors (David Lonie)
- PWSCF: Support newer versions, and the 'alat' term (Patrick Avery)
- SVG: Option to avoid addition of hydrogens to fill valence (Lee-Ping)
- SVG: Option to draw as ball-and-stick (Jean-Noël Avila)
- VASP: Vibration intensities are calculated (Christian Neiss, Mathias Laurin)
- VASP: Custom atom element sorting on writing (Kirill Okhotnikov)

21.2.3 Other new features and improvements

- 2D layout: Improved the choice of which bonds to designate as hash/wedge bonds around a stereo center (Craig James)
- 3D builder: Use bond length corrections based on bond order from Pyykko and Atsumi (https://doi.org/10.1002/chem.200901472) (Geoff Hutchison)
- 3D generation: "–gen3d", allow user to specify the desired speed/quality (Geoff Hutchison)

- Aromaticity: Improved detection (Geoff Hutchison)
- Canonicalisation: Changed behaviour for multi-molecule SMILES. Now each molecule is canonicalized individually and then sorted. (Geoff Hutchison/Tim Vandermeersch)
- Charge models: "–print" writes the partial charges to standard output after calculation (Geoff Hutchison)
- Conformations: Confab, the systematic conformation generator, has been incorporated into Open Babel (David Hall/Noel O'Boyle)
- Conformations: Initial support for ring rotamer sampling (Geoff Hutchison)
- Conformer searching: Performance improvement by avoiding gradient calculation and optimising the default parameters (Geoff Hutchison)
- EEM charge model: Extend to use additional params from https://doi.org/10.1186/s13321-015-0107-1 (Tomáš Raček)
- FillUnitCell operation: Improved behavior (Patrick Fuller)
- Find duplicates: The "–duplicate" option can now return duplicates instead of just removing them (Chris Morley)
- GAFF forcefield: Atom types updated to match Wang et al. J. Comp. Chem. 2004, 25, 1157 (Mohammad Ghahremanpour)
- New charge model: EQeq crystal charge equilibration method (a speed-optimized crystal-focused charge estimator, http://pubs.acs.org/doi/abs/10.1021/jz3008485) (David Lonie)
- New charge model: "fromfile" reads partial charges from a named file (Matt Harvey)
- New conversion operation: "changecell", for changing cell dimensions (Kirill Okhotnikov)
- New command-line utility: "obthermo", for extracting thermochemistry data from QM calculations (David van der Spoel)
- New fingerprint: ECFP (Geoff Hutchison/Noel O'Boyle/Roger Sayle)
- OBConversion: Improvements and API changes to deal with a long-standing memory leak (David Koes)
- OBAtom::IsHBondAcceptor(): Definition updated to take into account the atom environment (Stefano Forli)
- Performance: Faster ring-finding algorithm (Roger Sayle)
- Performance: Faster fingerprint similarity calculations if compiled with -DOPTIMIZE_NATIVE=ON (Noel O'Boyle/Jeff Janes)
- SMARTS matching: The "-s" option now accepts an integer specifying the number of matches required (Chris Morley)
- UFF: Update to use traditional Rappe angle potential (Geoff Hutchison)

21.2.4 Language bindings

- Bindings: Support compiling only the bindings against system libopenbabel (Reinis Danne)
- Java bindings: Add example Scala program using the Java bindings (Reinis Danne)
- New bindings: PHP (Maciej Wójcikowski)
- PHP bindings: BaPHPel, a simplified interface (Maciej Wójcikowski)
- Python bindings: Add 3D depiction support for Jupyter notebook (Patrick Fuller)
- Python bindings, Pybel: calccharges() and convertdbonds() added (Patrick Fuller, Björn Grüning)

- Python bindings, Pybel: compress output if filename ends with .gz (Maciej Wójcikowski)
- Python bindings, Pybel: Residue support (Maciej Wójcikowski)

21.2.5 Development/Build/Install Improvements

- Version control: move to git and GitHub from subversion and SourceForge
- Continuous integration: Travis for Linux builds and Appveyor for Windows builds (David Lonie and Noel O'Boyle)
- Python installer: Improvements to the Python setup.py installer and "pip install openbabel" (David Hall, Matt Swain, Joshua Swamidass)
- Compilation speedup: Speed up compilation by combining the tests (Noel O'Boyle)
- MacOSX: Support compiling with libc++ on MacOSX (Matt Swain)

21.2.6 Cast of contributors

Alexandr Fonari, Anders Steen Christensen, Andreas Kempe, arkose, Benoit Leblanc, Björn Grüning, Casper Steinmann, Chris Morley, Christoph Willing, Craig James, Dagmar Lenk, David Hall, David Koes, David Lonie, David van der Spoel, Dmitriy Fomichev, Fulvio Ciriaco, Fredrik Wallner, Geoff Hutchison, Heiko Becker, Itay Zandbank, Jean-Noel Avila, Jeff Janes, Joaquin Peralta, Joshua Swamidass, Julien Nabet, Karol Langner, Karthik Rajagopalan, Katsuhiko Nishimra, Kevin Horan, Kirill Okhotnikov, Lee-Ping, Matt Harvey, Maciej Wójcikowski, Marcus Hanwell, Mathias Laurin, Matt Swain, Mohamad Mohebifar, Mohammad Ghahremanpour, Noel O'Boyle, Patrick Avery, Patrick Fuller, Paul van Maaren, Peng Bai, Philipp Thiel, Reinis Danne, Ronald Cohen, Scott McKechnie, Stefano Forli, Steve Roughley, Steffen Moeller, Tim Vandermeersch, Tomas Racek, Tomáš Trnka, Tor Colvin, Torsten Sachse, Yi-Shu Tu, Zhixiong Zhao

21.3 Open Babel 2.3.1

Released on 2011-10-14.

This release represents a major bug-fix release and is a stable upgrade, strongly recommended for all users of Open Babel. Many bugs and enhancements have been added since the 2.3.0 release.

After 10 years, we finally published a paper discussing Open Babel. Please consider citing this work if you publish work which used Open Babel: Noel M. O'Boyle , Michael Banck , Craig A. James , Chris Morley , Tim Vandermeersch and Geoffrey R. Hutchison. "Open Babel: An open chemical toolbox." Journal of Cheminformatics 2011, 3:33. http://www.jcheminf.com/content/3/1/33

21.3.1 What's new from 2.3.0

- Better support for unknown stereochemistry, including a "wobbly" bond in 2D depiction.
- Many fixes for rare bugs with stereochemical conversions, including unusual valences.
- Significantly improved 2D depiction code, improving performance and cis/trans stereochemical accuracy
- Added support for direct 2D depiction to PNG files using the Cairo library, if available.
- PNG files from Open Babel contain molecular information and can be read to give the MDL Molfile.
- SVG files with 2D depiction can now include a grid of molecules with embedded JavaScript to zoom and scroll.

- Molecular formulas now include the total charge (e.g., HCO2-)
- Added the EEM partial charge model from Bultinck, et. al.
- Fixed problems with FastSearch databases larger than 4GB, now checking for large files.
- Improved performance with force field minimization, particularly the UFF and GAFF methods.
- Several MMFF94 atom typing bugs fixed.
- Updated GAFF parameters from the AmberTools distribution.
- Improvements in 3D coordinate generation, particularly more accurate sp3 bond angles
- Fixed tests for auto-typing molecules with force fields when running through different isomers.
- Improvements in scripting bindings, particularly Python, Ruby, and Java
- Pybel now uses the built-in 2D depiction, and no longer needs OASA.
- Added initial support for MM3 atom typing with the Tinker package
- Significant bug fixes for the PDBQT format.
- Reading FASTA files can now generate 3D coordinates for single-stranded DNA in addition to the default double-strand.
- Support for reading/writing unit cell information from MOPAC files.
- Support for re-numbering SMILES by specifying the first and last atoms with -xf and -xl flags.
- Better support for InChI -> InChI key generation by direct conversion, rather than re-perception of the InChI.
- Fix for rare stack overflow crash in SMARTS perception.
- Improved UNIX man pages.
- Many bug fixes and small enhancements

21.3.2 New File Formats

- Import and Export:

** Gromacs GRO * Import: ** ABINIT ** XCrySDen XSF * Export: ** InChI Key

21.4 Open Babel 2.3.0

Released on 2010-10-23.

This release represents a major update and should be a stable upgrade, strongly recommended for all users of Open Babel. Highlights include a completely rewritten stereochemistry engine, Spectrophore fingerprint generation, 2D depiction, improved 3D coordinate generation, conformer searching, and more. Many formats are improved or added, including CIF, PDBQT, SVG, and more. Improved developer API and scripting support and many, many bug fixes are also included.

21.4.1 What's new from 2.2.3

- Completely rewritten stereochemistry perception, including support for tetrahedral, square planar, and higher-order stereochemistry.

- Dramatically improved canonicalization algorithm (Note that in general, canonical SMILES have changed since the 2.2.x release.)
- 2D depiction, including SVG vector graphics generation using code from MCDL.
- New Spectrophore generation, contributed by Silicos NV.
- New ChargeMethod API including support for partial charge assignment from Gasteiger, MMFF94, QEq, QTPIE methods and plugin interface for adding more.
- Improved 3D coordinate generation.
- New conformer generation framework, including support for diverse conformer generation and genetic algorithm lowest-energy searching.
- Improved user documentation.
- Improved aromaticity / Kekule bond assignment.
- Improved unit test suite using the CMake-based CTest program.
- Improved support for crystallographic unit cells (e.g., in CIF format).
- Improved UFF force field method, including hypervalent 5, 6, 7 and higher coordination numbers.
- Support for the GAFF (Generalized Amber Force Field) method.
- Support for reading geometry optimizations as multiple conformers from Gaussian, GAMESS-US, and other quantum chemistry packages.
- Support for reading molecular orbital energies from quantum chemistry formats.
- Several memory leaks fixed.
- Fixed many compiler warnings.
- Fixed support for MinGW and Cygwin environments.
- Fixed bugs with Gaussian 09 output files.
- Latest released version of the InChI library (1.0.3) generating standard InChI.
- Many more bug fixes and small feature improvements.

21.4.2 New Command-Line Operations

- –canonical: Output atoms in canonical order for any file format (i.e., not just SMILES)
- –conformer: Run a conformer search on the input molecules (has many options)
- –gen2D: Generate a 2D depiction of the molecule
- –partialcharge <model>: Use the partial charge model supplied to generate charges (i.e., instead of default Gasteiger sigma model)
- –sort <descriptor>: Sort molecules by a specified descriptor
- –unique: Only output unique molecules (as determined by InChI generation)

21.4.3 New File Formats

Import & Export: - DL-POLY CONFIG - FHIaims XYZ - PDBQT

Import only: - DL-POLY HISTORY - GULP output - PWscf output - Text

Export only: - MNA (Multilevel Neighborhoods of Atoms) - SVG vector graphics

21.5 Open Babel 2.2.3

Released on 2009-07-31.

21.5.1 What's new from 2.2.2

This release represents an important bug-fix upgrade, strongly recommended for all users of Open Babel.

- Fixed bug in fingerprints in 2.2.2, where the default fingerprint format and bit length was changed inadvertently.
- Fixed detection of shared_ptr in tr1/memory.
- Fixed additional aromaticity / Kekule assignment bugs.
- Fixed several bugs in the MMCIF format.
- Additional bug fixes.

21.6 Open Babel 2.2.2

Released on 2009-07-04.

21.6.1 What's new from 2.2.1

This release represents a major bug-fix release and is a stable upgrade, strongly recommended for all users of Open Babel. While there may not be many new features, many crashes and other bugs have been fixed since 2.2.1.

- Upgraded to the new InChI 1.02 release to produce standardized InChI and InChIKey output.
- Fixed many stereochemistry errors when reading/writing SMILES. This is part of a larger project which will be finished in the 2.3 release.
- Fixed compilation and installation on Cygwin and MinGW platforms.
- Significantly improved aromaticity and Kekule bond assignment.
- Improved 2D -> 3D coordinate generation
- Improved coordinate generation using the –gen3d command-line operation
- Improved performance for coordinate generation.
- New –fillUC command-line operation for babel.
- Fixes to pH-dependent hydrogen addition.
- Added support for reading vibrational data from Molden, Molpro, and NWChem output files.
- Updated atomic radii from recent theoretical calculations.
- Fixed bug when reading gzip-compressed Mol2 or XML files.
- Close files after an error. Fixes a bug with Pybel where files would remain open.
- Many more bug fixes and small feature improvements.

21.6.2 New File Formats

Import & Export: - Molpro input and output. - VASP coordinate files (CONTCAR and POSCAR).

21.7 Open Babel 2.2.1

Released on 2009-03-01.

21.7.1 What's new from 2.2.0

This release represents a major bug-fix release and is a stable upgrade, strongly recommended for all users of Open Babel. While there may not be many new features, many crashes and other bugs have been fixed since 2.2.0.

- Improved scripting interfaces, including Python 3 support and improved Java and C# support.
- Added support for MACCS fingerprints. Thanks to the RDKit project.
- Many fixes and enhancements to the force field code. In particular, the UFF force field implementation should handle many more molecules.
- Improved 3D coordinate generation, particularly with ring fragments. You can give this a try with the obgen utility.
- Fixed a variety of PDB import errors with atom types.
- Added support for reading charges and radii from PQR file formats.
- Added support for reading and writing unit cells in PDB formats.
- New "output" file format for taking generic ".out", ".log", and ".dat" files and reading with appropriate file type based on contents. Currently works extremely well for quantum chemistry packages.
- Added improved error handling and reporting when unable to load file formats.
- Improved CIF file format support.
- Many, many, many additional bug fixes and small enhancements.

21.8 Open Babel 2.2.0

Released on 2008-07-04.

21.8.1 What's new from 2.1.1

- **New support for 3D coordinate generation using the OBBuilder class.** Note that this code directly supports non-chiral compounds Stereochemistry may or may not be supported in this release
- **Significantly faster force fields (up to 200x faster) and support** for constrained optimization.
- New force fields, including complete UFF, MMFF94, and MMFF94s implementations.
- Monte Carlo conformer search support, including a new obconformer tool.
- Unified framework for plugin classes, including easy-to program file formats, descriptors, filters, force fields, fingerprints, etc.

- A new "descriptor" plugin framework for QSAR descriptors, etc. Initial descriptors include hydrogen-bond donors, acceptors, octanol/water partition, topological polar surface area, molar refractivity, molecular weight, InChI, SMARTS, titles, Lipinski Rule of Five, etc.
- A new "filter" plugin framework for selecting molecules by title, molecular weight, etc.
- Facility to add new "ops", commandline options or operations on the conversion process as plugin code. Initial operations include 3D coordinate generation, tautomer standarization, and addition of polar hydrogens.
- Code for integrating Open Babel and the BOOST graph library.
- Improved scripting support, including new bindings for C# and improved Java, Ruby, Python, and Perl bindings.
- Space group support and thoroughly revised and improved CIF format.
- Initial support for 3D point group symmetry perception.
- Improved support for "grids" or "cubes" of molecular data, such as from quantum mechanics programs. (See below for supported file formats.)
- Initial support for reading trajectories and animations.
- Improved support for reaction formats, including CML, RXN, and Reaction SMILES.
- Improved residue handling in PDB and Mol2 formats.
- Improved pH-dependent hydrogen addition.
- **Latest released version of the InChI library, including use of the** latest "preferred" options for InChI generation.
- Support for the cross-platform CMake build system.
- **File format modules are now installed in a version-specific** directory on unix, preventing problems between 2.2.x and 2.1.x (or older) plugin libraries.
- **Framework to support "aliases" for group abbreviations, partially** implemented for MDL formats.
- Many more bug fixes and small feature improvements.

21.8.2 New File Formats

Import & Export: Chemkin Gaussian Cube Gaussian Z-matrix GROMACS xtc trajectories MCDL mmCIF OpenDX cube (e.g., from APBS) Reaction SMILES

Import only: Accelrys/MSI Cerius II MSI text format ADF output ADF Tape41 ASCII data GAMESS-UK input and output Molden structure PNG (for embedded chemical data) PQR

Export only: MSMS input ADF input InChI Keys

21.9 Open Babel 2.1.1

Released on 2007-07-07.

21.9.1 What's new from 2.1.0

- **Improved scripting support, including dictionary-support for** OBGenericData in pybel, casting from OBUnitCell, etc. Improved access to OBRings from OBMol.GetSSSR().
- **Added support for descriptors (e.g., PSA, logP) from scripting** interfaces.

- **Added support for reading all PDB records (beyond current atom and** bond connections). Records not handled directly by Open Babel are added as key/value pairs through OBPairData.
- **Added a new configure flag –with-pkglibdir to allow Linux package** distributors to define version-specific directories for file format plugins.
- **Fixed a bug which would not output chirality information for** canonical SMILES with 3D files.
- **Fixed problems with new line-ending code. Now correctly reads DOS** and old Mac OS files with non-UNIX line endings.
- **Correctly rejects SMILES with incorrect ring closures. Thanks to** Craig James for the report.
- Fixed a crash when output to canonical SMILES.
- Fixed a crash when converting from SMILES to InChI.
- Fixed a crash when reading some PDB files on Windows.
- Fixed a crash when reading invalid MDL/SDF files.
- Fixed a bug which made it impossible to read some GAMESS files.
- Fixed a problem when reading ChemDraw CDX files on Mac OS X.
- A large number of additional fixes, including some rare crashes.

21.10 Open Babel 2.1.0

Released on 2007-04-07.

21.10.1 What's new from 2.0.2

- **Now handles molecules with >65536 atoms or bonds. Some PDB entries,** in particular have such large molecular systems.
- **New features for molecular mechanics force fields, including energy** evaluation and geometry optimization. Ultimately, this will enable coordinate generation and refinement for SMILES and other formats. (A flexible force field framework is available for developers.)
- Implementation of the open source Ghemical all atom force field.
- **Framework for canonical atom numbering, including a new canonical** SMILES format.
- New support for Ruby and Java interfaces to the Open Babel library.
- Improved scripting interfaces through Perl and Python, including the new "pybel" module with a more Python-like syntax.
- **Automatically handles reading from text files with DOS or Mac OS 9** line endings.
- **Many enhancements to the Open Babel API: See the Developers API Notes for** more information.
- **New obenergy tool - evaluate the energy of a molecule using** molecular mechanics.
- **New obminimize tool - optimize the geometry of structures using** molecular mechanics.
- **Improved obprop tool - outputs a variety of molecular properties including** Topological Polar Surface Area (TPSA), Molar Refractivity (MR), and logP.

- **The babel tool can now setting program keywords for some quantum mechanics** formats from the command-line, including: GAMESS, Gaussian, Q-Chem, and MOPAC. (This feature can also be accessed by developers and expanded to other formats.)
- **New options for babel tool, including:** -e for continuing after errors -k for translating computational keywords (e.g., GAMESS, Gaussian, etc.) –join to join all input molecules into a single output –separate to separate disconnected fragments into separate molecular records -C (combine mols in first file with others having the same name) –property to add or replace a property (e.g., in an MDL SD file) –title to add or replace the molecule title –addtotitle to append text to the current molecule title –addformula to append the molecular formula to the current title
- Many more bug fixes and small feature improvements.

21.10.2 New File Formats

Import & Export: Carine's ASCII Crystal (ACR) ChemDraw CDX & CDXML Crystallographic Interchange Format (CIF) Fasta Sequence Thermo Format

Import: Gaussian fchk InChI

Export: Open Babel MolReport Titles

21.11 Open Babel 2.0.2

Released on 2006-07-24.

21.11.1 What's new from 2.0.1

- Substantial fixes to the SMILES and SMARTS parsing support, thanks to a variety of bug reports.
- A variety of fixes to aromaticity perception and Kekule form assignment.
- Fixed gzip support, broken in version 2.0.1 inadvertantly.
- Output a warning when a multi-molecule files is converted to a single-molecule format.
- Better support for command-line tools such as obgrep on Cygwin.
- Fixed a variety of crashes.
- Countless other bug fixes.

21.12 Open Babel 2.0.1

Released on 2006-04-17.

21.12.1 What's new from 2.0.0

- Support for dynamic building on the Cygwin environment. This fixes a long-standing problem that made Open Babel useless to Cygwin users.
- Fixed a variety of memory leaks and improved overall memory use. More work to reduce memory consumption is underway for the 2.1 release.

- Improved Perl and Python scripting wrappers, including many bug-fixes.
- Fixes to the “make check” test suite, which should prevent problems running before babel is installed.
- Fixes compilation problems with AIX, Fedora Core 4, and the newly-released GCC-4.1.
- Fixed several reported compilation problems with Windows builds using VisualC++.
- Fixed several reported crashes.
- Fixed problems with the Turbomole format, thanks to Mikael Johansson.
- Fixed a bug with PDB files with coordinates < -1000 Ang.
- Improved support for the Sybyl mol2 format, thanks to Kevin Parkes.
- Fixed a variety of typos in the API documentation.
- Countless bug fixes.

21.13 Open Babel 2.0

Released on 2005-11-26.

21.13.1 What’s new from 1.100.2

This release represents Open Babel’s fourth “birthday” and a milestone for a stable, flexible interface for developers and users alike.

- **New conversion framework. The new framework allows dynamic** loading/unloading of file translator modules (i.e., shared libraries, DLLs, DSO, etc.). More importantly, it facilitates adding new formats, since each format is self-contained and no editing of other files is required.
- Improved support for XML chemistry formats, including CML, PubChem XML,
- **Support for fingerprinting and calculation of Tanimoto coefficients for** similarity consideration. (A flexible fingerprint framework is available for developers.)
- New support for Perl and Python “wrappers” of the Open Babel library.
- **Many enhancements to the Open Babel API: See the Developers API Notes for** more information. Some code will require updating, see the Developer’s Migration Guide for details.
- **Support for automatically reading .gz compressed files.** (e.g., 1abc.pdb.gz is uncompressed and treated as a PDB file) Use of the -z flag creates gzip-compressed output files.
- Support for the new IUPAC InChI identifiers.
- **Improved bond order typing, including flexible SMARTS matching in** bondtyp.txt.
- **New Kekulization routine – improves aromaticity detection in aromatic amines** like pyrroles, porphyrins, etc.
- **Improved support for radicals and spin multiplicity, including** assignment of hydrogens to radicals.
- Improved support for 2D vs. 3D file formats.
- **New error logging framework keeps an “audit log” of changes to files** (hydrogen addition, bond order assignment) and different levels of error reporting / debugging. Use the “—errorlevel 4” flag to access this information.
- Improved atom typing and hydrogen addition rules.

- **Improved obfit utility will output RMSD and find matches with the** best RMSD.
- Updated isotope data from 2003 IUPAC standard.
- **Updated elemental data from the Blue Obelisk Data Repository.** (project started, in part, to validate the old Open Babel data)
- Improved z-matrix code (CartesianToInternal / InternalToCartesian).
- Countless bug fixes.

21.13.2 New File Formats

- **Import & Export:** ChemDraw CT (Connection Table) CML Reaction files MDL Molfile V3000 MDL Rxn files Open Babel free-form fractional (crystallographic coordinates) Open Babel fastsearch database format Open Babel fingerprint formats PCModel format YASARA.org YOB format Turbomole Improved CML support Improved Gaussian 98/03 support Improved SMILES import / export
- **Import-Only:** PubChem XML
- **Export-Only:** MPQC input Open Babel "copy" format (i.e., copy the raw input file) Sybyl MPD descriptor format IUPAC InChI descriptor
- **Changed formats:**
  - MMADS - eliminated
  - bin - OpenEye binary v 1, eliminated
  - GROMOS96 - changed from separate g96a & g96nm types to a unified g96 type. Defaults to output Angstroms, Use -xn to output nm.
  - Titles - eliminated – can be produced with SMILES -xt

21.14 Open Babel 1.100.2

Released on 2004-02-22.

21.14.1 What's new from 1.100.1

- Shared library (version 0:0:0) built by default on POSIX systems (e.g. Linux, BSD, Mac OS X…)
- Fixed installation of header files. The headers in the math/ subdirectory were not installed alongside the other headers.
- Added tools/ directory with small examples of using libopenbabel: * obgrep: Use SMARTS patterns to grep through multi-molecule files. * obfit: Use SMARTS patterns to align molecules on substructures. * obrotate: Rotate a torsional bond matching a SMARTS pattern.
- Improved PDB support: uses HETATM records more appropriately, attempts to determine chain/residue information if not available.
- Fixed a variety of bugs in ShelX support.
- Added support for handling atom and molecule spin multiplicity.
- Updated documentation – not yet complete, but significantly improved.

- Fixed major omissions in CML readers and writers. All versions of CML are now supported (CML1/2 and array/nonArray). Also added *.bat file for roundtripping between these formats for both 2- and 3-D data. Fixed bugs in test/cmltest/cs2a.mol.cml.
- Building and running the test-suite in a build-directory other than the source-directory is now fully supported.
- Support for the Intel C++ Compiler on GNU/Linux.
- Miscellaneous fixes to make it easier to compile on non-POSIX machines.

21.14.2 New File Formats

-Export: Chemtool Chemical Resource Kit (CRK) 2D and 3D Parallel Quantum Solutions (PQS)

-Import: CRK 2D and 3D PQS

21.15 Open Babel 1.100.1

Released on 2003-6-24.

21.15.1 What's new from 1.100.0

- Much better bond typing overall for files converted from formats without bond information (e.g. XYZ, QM codes). Fixed some bugs in 1.100.1 and added additional improvments.
- Support for the command-line "babel" program to convert some or all structures in a file with multiple molecules. By default this version will convert all molecules in a file. To change this, use the -f and -l command-line options as documented in the man page.
- Isotope support, including exact masses in the "report" file format and SMILES data.
- Updated API documentation.
- Support for the Borland C++ compiler.
- Fixed a variety of bugs in the PDB file format support, including better bond typing.
- Support for output of residue information in the Sybyl Mol2 file format.
- Some support for conversion of unit cell information, both in the library and in some file formats (i.e. DMol3, Cacao).
- Coordinates now use double-precision floating point libraries for greater accuracy in conversions.
- Fixed a variety of bugs uncovered in roundtrip testing.
- Fixed a bug when attempting to perceive bond information on 2D structures.
- Fixed several rare bugs that could cause segmentation faults.

21.15.2 New File Formats

-Import: ShelX -Export: ZINDO input

21.16 Open Babel 1.100.0

Released on 2002-12-12.

21.16.1 What's new from 1.99

- Bond order typing is performed when importing from formats with no notion of bonds (quantum chemistry programs, XYZ, etc.). -Now better conforms to the ISO C++ standard, should compile on most modern C++ compilers.
- Improved test suite, including "roundtrip" testing, ensuring more accurate translations.
- Support for the Chemical Markup Language (CML) and other file formats. (see below)
- Improved PDB support – should read PDB files more accurately and hew closer to the current PDB standard for export.
- Improved Gaussian input generation.
- Added support for the Chemical MIME standards, including command-line switches.
- Added support for using the babel program as a pipe for a "translation filter" for other programs.
- Can add hydrogen atoms based on pH.
- Fixed a variety of memory leaks, sometimes causing other bugs.
- Fixed a wide variety of bugs in various file formats.
- Faster SMARTS matching and some overall speedups across the program.
- API documentation using the Doxygen system.
- Of course there are *many* other bug-fixes and improvements.

21.16.2 New File Formats

> -Import: NWChem Output -Export: POV-Ray, NWChem Input -Both: CML, ViewMol, Chem3D

21.17 Open Babel 1.99

Released on 2002-1-29.

The Open Babel team is pleased to announce the release of Open Babel 1.99, a first beta release for the 2.0 version of the free, open-source replacement for the Babel chemistry file translation program.

At the moment, the beta release is not a drop-in replacement for babel as some file formats are not implemented and bond orders are not calculated for QM file formats.

Open Babel includes two components, a command-line utility and a C++ library. The command-line utility is intended to be used as a replacement for the original babel program, to translate between various chemical file formats. The C++ library includes all of the file-translation code as well as a wide variety of utilities to foster development of other open source chemistry software.

Bibliography

[obj2011] Noel M. O'Boyle, Michael Banck, Craig A. James, Chris Morley, Tim Vandermeersch, Geoffrey R. Hutchison **Open Babel: An open chemical toolbox.** *J. Cheminf.* **2011**, *3*, 33. [Link]

[bll2002] P. Bultinck, W. Langenaeker, P. Lahorte, F. De Proft, P. Geerlings, C. Van Alsenoy, and J. P. Tollenaere. **The Electronegativity Equalization Method II: Applicability of Different Atomic Charge Schemes.** *J. Phys. Chem. A* **2002**, *106*, 7895-7901. [Link]

[blc2003] Patrick Bultinck, Wilfried Langenaeker, Ramon Carbó-Dorca, and Jan P. Tollenaere. **Fast Calculation of Quantum Chemical Molecular Descriptors from the Electronegativity Equalization Method.** *J. Chem. Inf. Comput. Sci.* **2003**, *43*, 422-428. [Link]

[omh2008] N.M. O'Boyle, C. Morley and G.R. Hutchison. **Pybel: a Python wrapper for the OpenBabel cheminformatics toolkit.** *Chem. Cent. J.* **2008**, *2*, 5. [Link]

[bmg2004] Andreas Bender, Hamse Y. Mussa, and Robert C. Glen. **Molecular Similarity Searching Using Atom Environments, Information-Based Feature Selection, and a Naive Bayesian Classifier.** *J. Chem. Inf. Comput. Sci.* **2004**, *44*, 170-178. [Link]

[fpbg99] Dmitrii Filimonov, Vladimir Poroikov, Yulia Borodina, and Tatyana Gloriozova. **Chemical Similarity Assessment through Multilevel Neighborhoods of Atoms: Definition and Comparison with the Other Descriptors.** *J. Chem. Inf. Comput. Sci.* **1999**, *39*, 666-670. [Link]

[gb2001] A.A. Gakh and M.N. Burnett. **Modular Chemical Descriptor Language (MCDL): Composition, Connectivity and Supplementary Modules.** *J. Chem. Inf. Comput. Sci.*, **2004**, *41*, 1491-1499. [Link]