# Windows Installation Guide

Matthew Caldwell

August 6, 2015

BCMD requires a Unix-esque runtime environment including a command shell, compilers for C and Fortran, a Make system, Python interpreter and some other tools. Getting all of this up and running on Windows can be a bit fiddly and tedious. Here we describe an approach that has worked for us. However, many of the tools mentioned continue to evolve, as indeed does the Windows operating system itself, so some of this advice might be obsolete by the time you come to attempt it. If you discover any issues, or find other better ways to do things, please let us know.

## 1 Essentials

### 1.1 Text Editor

To edit configuration files, model definitions, input files, batch jobs and miscellaneous other text formats encountered along the way, you will need a decent editor. Feel free to use any particular favourite, but not the Windows default Notepad, which is definitely not up to the job. If you don't have another strong preference, Notepad++ (`https://notepad-plus-plus.org`) is popular and well-regarded. Notepad++ syntax highlighting definitions for BCMD's file formats can be found in the `util` directory of the BCMD distribution (in the files `modeldef.xml`, `input.xml` and `batch.xml`). For information on installing these, see the Notepad++ documentation.

### 1.2 Base Runtime Environment

To provide the basis of the environment, we use the MinGW port of the GNU compiler collection, along with the closely-related MSYS2 command shell. As of this writing, the easiest way to install these is to use the installer provided at `http://msys2.github.io`. Instructions are also given on that page, but in brief:

- Download and run the appropriate installer. (You probably want the 64-bit version. The instructions below assume this, but you should be able to get things working with the 32-bit version too with some minor changes.) Accept the defaults for installation options.
- After the installer finishes, run MSYS2 and update the system packages with the command:

  ```
  pacman --needed -Sy bash pacman pacman-mirrors msys2-runtime
  ```

- Exit and reopen MSYS2 and update the rest of it:

  ```
  pacman -Su
  ```

- Exit and reopen MSYS2 again, and install the required languages and build tools:

```
pacman -S gcc-fortran
pacman -S make
pacman -S git
```

At this point most of the basic requirements should be met. Try typing

```
gfortran --version
```

You should get a message along the lines of

```
GNU Fortran (GCC) 4.9.2
Copyright (C) 2014 Free Software Foundation, Inc.
```

and some licensing boilerplate, in which case all is well.

MSYS creates a fake Unix directory structure inside the Windows filing system, complete with standard directories like `/usr/local/bin`, plus directories corresponding to Windows drive letters such that `C:` becomes `/c` and so on. The 'root' of this directory structure is the directory you installed MSYS into — for the 64-bit version the default is `C:\msys64`. Within this is your user directory, which is where the MSYS shell starts up — something like `C:\msys64\home\USERNAME`. Find this and make a note of its location.

## 1.3 Python

The next thing you need is an installation of Python with a NumPy/SciPy stack. There are several options, none of which is ideal.

If you are happy to use only the command-line environment, then you can use the version of Python built for MinGW/MSYS2 itself. This is relatively simple to install, but unfortunately it is built without support for the Tkinter GUI package. Since building and running models is significantly easier using the BCMD GUI, we somewhat reluctantly recommend using the Enthought Canopy Python distribution instead. Installation for both options is described below.

### 1.3.1 Using the MinGW Python

Assuming you have a 64-bit installation of MinGW, install the required packages as follows:

```
pacman -S mingw64/mingw-w64-x86_64-python2-scipy
pacman -S mingw64/mingw-w64-x86_64-python2-matplotlib
```

Note that this installation is quite slow, and in particular gives the misleading impression of having stalled while setting up `fontconfig` — just give it plenty of time.

The Python executable that gets installed here is named `python2.exe`. Some of the BCMD files assume that Python will be called `python`, so create a soft link like this:

```
ln -s /usr/local/bin/python2.exe /usr/local/bin/python
```

Now type:

```
python --version
```

If you see something along the lines of

```
Python 2.7.3 --  64-bit
```

then you're ready to install BCMD (§1.4 below).

### 1.3.2 Using Enthought Canopy

Canopy is a pre-built Python/SciPy distribution available for various operating systems, including Windows. It comes in several versions, and there is an Academic license available, but the free Express edition is sufficient for running BCMD.

Download and run the installer from https://store.enthought.com/downloads/. By default the Canopy installation will be placed in your Windows user home directory — the installer should report the location, somewhere like `C:\Users\USERNAME\AppData\Local\Enthought\Canopy`. If you prefer to change this, ensure that you modify any corresponding paths below.

You then need to ensure that the Python executable is found on the PATH in MSYS. While you're at it, it's best to also remove any paths to Windows programs that you won't need from the command line. If any paths have spaces in them (things like `Local Data` or `Program Files`), these should be escaped with a backslash (\). Edit the file `.bash_profile` (note the leading full stop) in your MSYS home directory to set and export the necessary paths, like this:

```
PATH=/c/users/USERNAME/appdata/local/enthought/canopy/user/scripts
PATH=$PATH:/c/users/USERNAME/appdata/local/enthought/canopy/user
PATH=$PATH:/usr/local/bin:/usr/bin:/bin:/opt/bin
PATH=$PATH:/usr/bin/site_perl:/usr/bin/vendor_perl:/usr/bin/core_perl
PATH=$PATH:/c/windows/system32:/c/windows
export PATH
```

Note that the exact paths needed may vary, so check the actual locations on your system.

Now open a new MSYS window and type:

```
python --version
```

If you see something along the lines of

```
Python 2.7.3 --  64-bit
```

then you're good to go.

**Note:** Canopy's Python interpreter does not recognise the MSYS shell as an interactive terminal, so it will not run an interactive session by default. If you just type

```
python
```

then it will look like the shell has hung up (you can break it out of this by pressing ctrl-C). Instead, you must explicitly specify the `-i` option:

```
python -i
```

(This shouldn't affect the BCMD scripts, it's only important if you wish to run interactive Python sessions of your own.)

## 1.4 BCMD

The best way to install the BCMD system itself is from GitHub:

```
git clone https://github.com/bcmd/BCMD.git bcmd
```

(This puts BCMD in a directory called `bcmd` inside your MSYS home directory. Feel free to put it somewhere else if you prefer, just remember to adjust any relevant paths used below accordingly.)

Once the clone has completed, go into the `bcmd` directory and build it:

```
cd bcmd
./configure
make
```

This should generate quite a few lines of output. The following warnings and errors should appear near the end:

```
rm: parsetab.*: No such file or directory
WARNING: Token 'COMMENT' defined, but not used
WARNING: Token 'NEWINDENT' defined, but not used
WARNING: There are 2 unused tokens
Generating LALR tables
Makefile:115: recipe for target 'bparser/parsetab.py' failed
make: [bparser/parsetab.py] Error 2 (ignored)
```

These are expected and can be safely ignored. If there are other errors reported, check the output carefully. It probably means that some required software has not been installed correctly. Re-check all the steps above. Failing that, please email m.caldwell@ucl.ac.uk with details of the problem.

If the system appears to have built correctly, try running a simple model:

```
make build/huxley.detail
tail -n 1 build/huxley.detail
```

This should print a line that looks something like this:

```
88 20 0.47454650748579158 0.05586874751218391 0.31533817617053456 0.60142841645812495
```

If you are using Enthought Canopy for your Python installation, you should also be able to run the GUI by typing:

```
python bgui.py
```

See the main BCMD manual for more information about using these interfaces.

### 1.4.1 Updating

The BCMD distribution on GitHub is updated periodically as new features are added and bugs get fixed. To update your installation to the latest version, open an MSYS window, change to your BCMD directory and type:

```
git pull
```

Note that this will only work seamlessly if you have not modified the BCMD files yourself (that doesn't include adding new files of your own—model definitions, inputs and so on—which is fine). If you have made changes, you will probably see a warning about conflicting changes. In that case, you can either commit and merge your own changes, or delete the changed files and allow Git to restore the master versions. The latter is usually easier, but is obviously not appropriate if you've made changes you want to keep. For more information about using Git, see the extensive documentation at https://git-scm.com/doc. You might also find Roger Dudler's quick-and-dirty overview helpful: http://rogerdudler.github.io/git-guide/.

# 2 Additional Capabilities

The above process should have got you to the point of being able to build and run models. Some further features of BCMD require additional external software. Installing these things is optional but recommended.

## 2.1 Python packages

Optimisation and sensitivity analysis require the installation of additional Python packages. Before doing so, you need to ensure your Enthought Canopy setup tools are up to date:

- Run the Canopy GUI.
- Log in (create a free user account if you have not done so previously).
- Go to the Package Manager and select the Updates item in the list on the left hand side.
- As a minimum, check for updates to **pip** and **setuptools** and install those if present. In general it is probably a good idea to choose the Install All Updates button, since there may also be other relevant packages that need updating.

Python provides several different ways of distributing packages. Unfortunately, they don't work consistently for our requirements. In particular, packages that have native code dependencies may demand building with Microsoft Visual C++ (possibly even a specific version of it). If you do have this installed, please let us know how it works out for the cases mentioned below that we haven't managed to get working without it.

### 2.1.1 Sensitivity Analysis

The sensitivity analysis features of `dsim.py` require the Python sensitivity library SALib. This can be installed with `pip`, without recourse to Visual C++:

```
pip install SALib
```

### 2.1.2 Optimisation

Running optimisation jobs with `optim.py` requires the OpenOpt libraries (http://openopt.org/). These do not install correctly with `pip`, but they will with `easy_install`. Four separate installations are needed:

```
easy_install openopt
easy_install FuncDesigner
easy_install DerApproximator
easy_install SpaceFuncs
```

The distribution provides several useful solvers including the Genetic Algorithm `galileo`, which is `optim.py`'s default.

Using the PSwarm method involves an additional external dependency on the PSwarm library by Vaz & Vicente. Compiled Windows binaries with Python bindings are available from the PSwarm home page at http://www.norg.uminho.pt/aivaz/pswarm/. However, we have not yet been able to persuade these to work correctly with Canopy.

### 2.1.3 libSBML

Exporting models to SBML requires Python bindings to the parsing library libSBML (Bornstein et al. 2008). In theory, the corresponding package should be installable via `pip`:

```
pip install python-libsbml
```

However, this not only requires Microsoft Visual C++ but also introduces further dependencies on several other libraries. At present we have not succeeded in getting this to work correctly with Canopy. (To be honest, we don't think the functionality is worth having anyway.)

## 2.2 GraphViz

BCMD creates dependency graphs of models using the GraphViz DOT language. In order to compile these to actual graphics you need to install GraphViz.

Download and run the installer from http://www.graphviz.org/Download_windows.php. Once installed, you will need to add the GraphViz `bin` directory to your PATH. Locate this directory—in our tests it was `C:\Program Files\GraphViz2.38\bin`—and add it to the `.bash_profile` file that you edited earlier. Insert a line like this just before the `export PATH` line:

```
PATH=$PATH:/c/Program\ Files/GraphViz2.38/bin
```

Note that the space in `Program Files` needs to be escaped with a backslash `\`.

## 2.3 LaTeX

BCMD can generate model documentation in LaTeX `.tex` format. To typeset this into a final document you will need a LaTeX distribution such as TeX Live, obtainable from the TeX Users Group site (http://www.tug.org/texlive/acquire-netinstall.html). Alternatively, you could use an online LaTeX service such as Overleaf.

Note that BCMD does not invoke LaTeX itself, and does not need it to be present to generate the file.

# References

Benjamin J Bornstein, Sarah M Keating, Akiya Jouraku, and Michael Hucka. LibSBML: an API library for SBML. *Bioinformatics (Oxford, England)*, 24(6):880–881, March 2008. doi: 10.1093/bioinformatics/btn051. URL http://bioinformatics.oxfordjournals.org/cgi/doi/10.1093/bioinformatics/btn051.