C++ Build Environment Setup for Building Bio-Formats and OmeroCpp

Roger Leigh

Wednesday 3rd September 2014
University of Dundee

Overview

Default compilers

- FreeBSD: LLVM/`clang++` or GCC/`g++`
- Linux: GCC/`g++` and GNU Binutils/`ld`
- MacOS X: XCode (custom LLVM/`clang++`)
- Windows: Visual Studio or Visual Studio Express (MSVC/`cl`)

Earlier versions of MacOS X used GCC 4.2.

Package managers

- FreeBSD: Ports (e.g. `pkg`, `portmaster`)
- Linux: Distribution package manager (e.g. `apt-get` or `yum`)
- MacOS X: homebrew (`brew`)
- Windows: Yeah, right. You need to manually download all the tools and then compile all the libraries by hand for your specific version of Visual Studio. (Microsoft love to make development for their platform easy and painless. Not!)

On the next few pages, the needed packages for each platform will be detailed. This includes all packages needed for Bio-Formats and OMERO including unit testing and API documentation generation; you might not need them all but it doesn't hurt to have them all.

FreeBSD packages

Run `pkg install` to install:

```
devel/apache-ant
devel/boost-all
devel/binutils
devel/cmake
devel/doxygen
devel/git
devel/googletest
devel/ice
graphics/graphviz
graphics/tiff
java/openjdk7
java/junit
lang/clang33
lang/python
lang/python27
print/texlive-full
science/hdf5
textproc/py-genshi
textproc/py-sphinx
textproc/xerces-c3
```

Add `/usr/local/bin` before `/usr/bin` in the `PATH` so that the newer GNU `ld` is used.

Debian and Ubuntu packages

Run `apt-get install` to install:

- `ant ant-contrib ant-optional`
- `build-essential`
- `cmake`
- `doxygen`
- `git`
- `graphviz`
- `junit4`
- `libboost-all-dev`
- `libgtest-dev`
- `libtiff5-dev`
- `libxerces-c-dev`
- `libzeroc-ice35-dev ice35-services ice35-slice ice35-translators icebox python-zeroc-ice`
- `openjdk-7-jdk openjdk-7-jre`
- `python python2.7`
- `texlive-full`
- `libhdf5-dev`
- `python-genshi` (or use `pip`)
- `python-sphinx` (or use `pip`)

CentOS and RHEL packages

Run `yum groupinstall "Development Tools"`

Run `yum install` to install:

`boost-devel`	`java-1.7.0-openjdk`
`cmake`	`junit4`
`doxygen`	`libtiff-devel`
`git`	`python-genshi` (or use `pip`)
`graphviz`	`python`
`gtest-devel`	`texlive-full`
`hdf5-devel`	`xerces-c-devel`

Install the following by hand:

- Ant
- JUnit
- Ice (RPMs available)
- TeXLive (via `install-tl`)
- `sphinx` (via `pip`)

MacOS X homebrew packages

Install XCode and its command line tools

Run `brew install` to install:

- `ant`
- `boost`
- `cmake`
- `doxygen`
- `git`
- `graphviz`
- `hdf5`
- `ice`
- `libtiff`
- `python`
- `xerces-c`

Install the following by hand:

- Google Test (`gtest`) from zip or subversion
- Java (JDK 1.7 from Oracle)
- TeXLive (via `install-tl` or MacTeX)
- `sphinx` (via `pip`)

Windows installation (packages)

Install the following by hand:

- Ant
- CMake
- Doxygen and Graphviz
- Git (msysgit)
- Ice (latest ZeroC installer or our 3.5.1 build)
- Java (latest JDK 1.7 from Oracle)
- LaTeX (MikTeX)
- Python (latest 2.7 from python.org; 64-bit recommended)
- `genshi`
- `sphinx`
- Visual Studio (2010, 2012 or 2013; Full or Express edition)

Windows installation (libraries)

For python, either download separate installers for each packages, or install `setuptools` and `pip` for Python, then `pip install` needed packages; ensure any downloaded packages are 64-bit if using 64-bit python)

Download and build `gtest` using `cmake` (no installation required)

Build and install the following by hand (for Bio-Formats):

- `boost`
- `hdf5`
- `icu`
- `tiff`
- `xerces`
- `zlib`

…and possibly more—we haven't yet done a Bio-Formats C++ build on Windows.

Obtaining packages by hand

- Google Test .... (website) (download zip) (svn tag)
- CMake .... (website) (download)
- Java .... (JDK7 download)
- Visual Studio .... (Dundee staff) (Express download)
- Ant .... (website) (download)
- Git .... (website) (download)
- Ice .... (website) (download)
- Python .... (website) (download) (extra packages)
- LaTeX .... (TeXLive) (TeXLive install) (MikTeX website) (MikTeX download)
- Doxygen .... (website) (download)
- Graphviz .... (website) (download)

System configuration

- In general, none of the tools should require any configuration
- LaTeX may require local font configuration to make the TeX Gyre fonts available.
  - Linux and FreeBSD: Use the provided `fontconfig` template or create your own
  - MacOS X: Add to system using FontBook
  - Windows: May need adding to the system fonts if not found automatically

Environment configuration

- Primarily needed on Windows
- Rather than setting globally, make a batch file which can set up the environment.
- Activate a python virtualenv if needed
- Ensure that all tools are on the user `PATH`
  - `ant`, `cmake`, `doxygen`, `dot`, `git`, `python`, `java`, `sphinx`, `xelatex`
- Set `CMAKE_PREFIX_PATH` if some libraries and tools are not on the default search path.
- Not all tools need to be on the default path; some will be discovered automatically by `cmake`
- No need to use a special Visual Studio shell when using `cmake`

Available build systems

There are many available build systems, which include:

- Make and GNU Make
- GNU Autotools
- CMake
- Qt `qmake`
- SCons
- Jam / BJam
- Ant / Maven / Gradle

`cmake` overview

`CMakeLists.txt`

`cmake`

- Compiler detection
- Program detection
- Library detection
- Feature tests
- Package options
- Rules for creating libraries programs, unit tests, installation

`Makefile` (**`make`**) or `project.sln and .vcxproj` (Visual Studio **`msbuild`**)

Eclipse project | Sublime text project

…and many more build systems and IDES are supported

`cmake` features

- `cmake` is a generic cross-platform build system
- `cmake` generates build files for a large number of common build systems
- On FreeBSD, Linux and MacOS X, `make` `Makefile`s will be used
- On Windows with Visual Studio, `msbuild` `.sln` solution files will be used
- Eclipse, Sublime Text, Kate, Code::Blocks or several other IDEs or build systems may be used instead, if desired

Using `cmake` (live demo)

Basic cmake usage

- Basic options
- Available generators

Building `gtest` on MacOS X

- Running cmake
- Building

Using `cmake` (live demo)

Building Bio-Formats on MacOS X

- Running cmake
- Cache variables
- Building
- Testing
- Installing

Building OmeroCpp on Windows

- Running cmake
- Building
- Installing

Building `gtest` on UNIX

Build from downloaded zip:

```
% cd /tmp
% unzip ~/Downloads/gtest-1.7.0.zip
% cd gtest-1.7.0
% cmake .
% make
```

This is used with other builds by setting the `GTEST_ROOT` environment variable or the *GTEST_ROOT* `cmake` cache variable.

Building `gtest` on Debian or Ubuntu

Build using installed sources and headers from the `libgtest-dev` package:

```
% cd /tmp
% mkdir gtest
% cd gtest
% cmake /usr/src/gtest
% make
```

This is used with other builds by setting the `GTEST_ROOT` environment variable or the *GTEST_ROOT* `cmake` cache variable.

Building Bio-Formats on Unix (1)

Building from git or release zip:
Configure the build:

```
% mkdir /tmp/bfbuild
% cd /tmp/bfbuild
% cmake -DGTEST_ROOT=/tmp/gtest /path/to/bioformats
```

Show cache variables and advanced cache variables which may be used to customise the build:

```
% cmake -LH
% cmake -LAH
```

Run the build with either of:

```
% make [VERBOSE=1]
% cmake --build .
```

Build the API reference documentation with either of:

```
% make doc
% cmake --build . --target doc
```

Building Bio-Formats on Unix (2)

Run the unit tests with any of:

```
% make test
% cmake --build . --target test
% ctest [-V]
```

Individual tests may be run by hand:

```
% cpp/test/ome-bioformats/pixelbuffer
% cpp/test/ome-bioformats/pixelbuffer --gtest_help
```

Use *--gtest_help* to list test options. Useful when debugging to run specific tests or subsets of the tests.

Building Bio-Formats on Unix (3)

Install the build with either of:

```
% make install [VERBOSE=1] [DESTDIR=/staging/path]
% cmake --build . --target install
```

By default, this will install into *CMAKE_INSTALL_PREFIX* which will default to `/usr/local`. Use *DESTDIR* to install into an alternative prefixed location, which is useful for testing and packaging for release.

Building OmeroCpp on Unix (1)

Building from git or release zip:
Configure the build. optionally showing Ice autodetection diagnostics:

```
% mkdir /tmp/ocppbuild
% cd /tmp/ocppbuild
% cmake -DGTEST_ROOT=/tmp/gtest [-DIce_HOME=/path/to/ice] \
  [-DIce_DEBUG=ON] /path/to/openmicroscopy
```

Show cache variables and advanced cache variables which may be used to customise the build:

```
% cmake -LH
% cmake -LAH
```

Run the build with either of:

```
% make [VERBOSE=1]
% cmake --build .
```

Building OmeroCpp on Unix (2)

Alternatively, it is possible to build in the openmicroscopy tree directly:

```
% ./build.py
% ./build.py build-cpp -Dcmake.opts="cmake options"
```

However, passing in cmake options and using different generators is much more difficult and more fragile with this method.

Building OmeroCpp on Unix (3)

Run the unit tests with any of:

```
% make test
% cmake --build . --target test
% ctest [-V]
```

Note that `ICE_CONFIG` needs setting with the details of a running OMERO server which the unit and integration tests can connect to for testing against.

Individual tests may be run by hand:

```
% components/tools/OmeroCpp/test/unit/unit
% components/tools/OmeroCpp/test/unit/unit --gtest_help
```

Use *--gtest_help* to list test options. Useful when debugging to run specific tests or subsets of the tests.

Building OmeroCpp on Unix (4)

Install the build with either of:

```
% make install [VERBOSE=1] [DESTDIR=/staging/path]
% cmake --build . --target install
```

By default, this will install into *CMAKE_INSTALL_PREFIX* which will default to `/usr/local`. Use *DESTDIR* to install into an alternative prefixed location, which is useful for testing and packaging for release.

Windows environment

I set up the environment with a custom batch file:

```
set "ICE_HOME=C:\Program Files (x86)\ZeroC\Ice-3.5.1"
set "PATH=%ICE_HOME%\bin\vc110\x64;C:\Program Files (x86)\CMake\bin;%PATH%"
c:\venv\27\scripts\activate
```

I also have Ant, Git, Java (JDK), and LaTeX on the default `PATH`. However, these could also be included in the custom batch file.
I use ConsoleZ with custom tabs which source different batch files to create different environments. For the above, I use the following command to set up a custom OMERO tab:

```
C:\Windows\System32\cmd /k C:\Users\rleigh\bin\omeroenv.bat
```

Note that the Ice setup is only required for running `build.py`; it is optional for direct use of `cmake`.

Building `gtest` on Windows

Download and unpack `gtest`, then run:

```
> set CL=/D_VARIADIC_MAX=10
> cd c:\Users\rleigh\gtest-1.7.0
> cmake -G "Visual Studio 11 2012 Win64" .
> cmake --build .
```

The `_VARIADIC_MAX=10` define works around a lack of variadic templates in this version of Visual Studio; may affect other Visual Studio versions. Leave set for the remaining steps.

Building OmeroCpp on Windows

Note: OmeroCpp building on Windows is a work in progress and not get completely finished.

Note: starting from a *clean* and up-to-date *develop* branch of `openmicroscopy.git` located in `c:\Users\rleigh\openmicroscopy`.

```
> mkdir c:\Users\rleigh\ocppbuild
> cd c:\Users\rleigh\ocppbuild
> cmake -G "Visual Studio 11 2012 Win64" \
  -DGTEST_ROOT=C:\Users\rleigh\gtest-1.7.0 \
  -DGTEST_LIBRARY=C:\Users\rleigh\gtest-1.7.0\Debug\gtest.lib \
 -DGTEST_MAIN_LIBRARY=C:\Users\rleigh\gtest-1.7.0\Debug\gtest_main.lib \
  ..\openmicroscopy
> cmake --build .
```

After running `cmake`, it's also possible to open the solution file in Visual Studio and build from inside the application.

Acknowledgements

- OME Team, Dundee
  - Jason Swedlow
  - Jean-Marie Burel
  - Mark Carroll
  - Andrew Patterson
  - …and the rest of the team
- Micron, Oxford
  - Douglas Russell
- Glencoe Software
  - Melissa Linkert
  - Josh Moore

OME

University of Dundee

wellcome trust

Strategic Award