Simple Project Builds with CMake

Evan Bollig

bollig@scs.fsu.edu

http://www.scs.fsu.edu/~bollig

Outline

- What is CMake?
- Getting Started
- Frequently Used Commands
- Various Examples
- Expectations for Homeworks
- Q&A Session

What is CMake?

- CMake is a cross platform Makefile generator
  - Windows, Linux, OS X, etc.
- We tell CMake our end goal and it takes care of the rest.
  - “the rest” includes searching for dependencies (libs, headers, etc), required commands (i.e. tar, doxygen, gcc)
  - CMake has lists of presets but we have the option to override if necessary.

(Continued)

- CMake provides support for compiling various languages and locating toolkits
  - C++, Fortan, Java, Perl, Python, Tcl, etc.
  - SWIG, Qt, MPI, OpenGL, PNG, etc.
  - Users can add custom support (i.e. Gordon’s macro for locating and using CUDA)
- Users can autoconfigure projects with “cmake .” or use the CMake UI with “ccmake .”
  - UI allows us to specify values for all known/available keys but prevents adding custom tags or macros

What CMake Is NOT

- CMake does not build your project; it only creates a Makefile to do this for you
- CMake is not well documented for all problems; experienced users and Google are the best resources
- CMake is not complete! Changes are made often unlike Autoconfig, Configure and QMake
  - Current release: 2.4.7
  - We will discuss a few known bugs/incomplete support in our examples: Fortran Modules; Java Subdirectories; Doxygen

Getting Started

- Write your code
- Write a CMakeLists.txt file in the same directory as your code
- “cmake .” (generates a Makefile)
- “make” (compiles your lib or binary)
- Now lets look at a helloworld example…

HelloWorld.cxx

```
#include <iostream>
using namespace std;

int main(int argc, char**argv) {
   cout << “hello world!” << endl;
}
```

CMakeLists.txt

```
PROJECT( hello CXX )

ADD_EXECUTABLE(hello.exe HelloWorld.cxx)
```

Directory Structure (Before)

- $HOME/helloworld/
  - HelloWorld.cxx
  - CMakeLists.txt

Run “CMake .”

- Output Should Resemble:

```
class03:~/helloworld> cmake .
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Configuring done
-- Generating done
-- Build files have been written to: /home/guests/users/bollig/helloworld
```

Directory Structure (After)

- $HOME/helloworld/
  - HelloWorld.cxx
  - CMakeLists.txt
  - Makefile → What we want! (rest is junk)
  - CMakeFiles/
  - CMakeCache.txt
  - cmake_install.cmake

Run “make”

- Output Should Resemble:

```
class03:~/helloworld> make .
Scanning dependencies of target hello.exe
[100%] Building CXX object CMakeFiles/hello.exe.dir/HelloWorld.o
Linking CXX executable hello.exe
[100%] Built target hello.exe
```

Directory Structure (Finished)

- $HOME/helloworld/
  - HelloWorld.cxx
  - CMakeLists.txt
  - Makefile
  - CMakeFiles/
  - CMakeCache.txt
  - cmake_install.cmake
  - hello.exe

Frequently Used CMake Commands

- PROJECT([name] [TYPE])
  - Start a new project named [name] with type [TYPE] (i.e. CXX, Fortran, Java, etc). NOTE: [TYPE] is case-sensitive!
- ADD_EXECUTABLE ([bin] [src1 src2 ..])
  - Create a binary named [bin] using source files [src1 src2 …]
- ADD_LIBRARY ([lib] [“”|“SHARED”|“STATIC”] [src1 src2 ..])
  - Create a library named [bin] that is either “SHARED” or “STATIC” using sources [src1 src2 ...].
- SUBDIRS ([subdir1 subdir2 …])
  - Do a multi-level build, including subdirectories [subdir1 subdir2 …] (each should have their own CMakeLists.txt)

(Continued)

- SET ([key] [value])
  - Insert a key=value pair into the build environment.
- FIND_PACKAGE([name])
  - Search for the package named [name] (i.e. Java, OpenGL, Doxygen, etc). If found CMake will setup vars specific to each package (i.e. ${JAVA_RUNTIME}, ${JAVA_COMPILER}, etc.) so you can use them in your CMake files.
- TARGET_LINK_LIBRARIES ([bin] [lib1 lib2 …])
  - Link libraries [lib1 lib2 …] with executable [bin]. NOTE: when specifying libraries do not put “lib[name].[a|so]”. Put “[name]” and cmake will determine correct type (shared/static) and location.
- INCLUDE_DIRECTORIES ([incdir1 incdir2 …])
  - Include the directories [incdir1 incdir2 …] (absolute or relative paths) in the search for dependencies, headers and libraries.

(Continued)

- ENABLE_TESTING()
  - Turn testing on so that CMake can call CTest on whatever tests we specify
- ADD_TEST([name] [bin] [arg1 arg2 …])
  - Specify a new test named [name] which consists of executing [bin] with arguments [arg1 arg2 …]. If the execution returns 0 the test passes otherwise it fails
- INCLUDE([file])
  - Merge the content of [file] directly into current file where this tag exists. This allows us to “inherit” properties, tests, commands, etc from external (possibly common CMake files).

Examples

- Get the tarball from http://www.scs.fsu.edu/~bollig/cmake_exa
- “tar xvfz cmake_examples.tgz”
- “cd cmake_examples”
- View README for an idea of what each example does (I will highlight key concepts in each example).

Expectations for SciProg Homeworks

- All students must submit homeworks to web submission tool.
  - This requires you to make a tarball
- TA or Professor must be able to build your source without problems
  - Unless you provide a one-liner in your README, you are required to have Makefiles or CMake files
- All code should be documented
  - Use Doxygen (Full lecture should happen soon)
- Many homeworks will have multiple directories or refer to code written for previous assignments
  - Consider SUBDIRS and compiling assignments as libraries with only the main method in external file to drive program.

Questions?

- Ask me now…or,
- Search Google, Yahoo, etc.
- Read the CMake Webpage and Wiki
  - http://www.cmake.org/HTML/Index.html
  - http://www.cmake.org/Wiki/CMake
- Read “Mastering CMake” by Martin and Hoffman
  - I have a single copy which I will keep in the SCS VisLab (428 Dirac). My copy does not leave that room, sorry! If it’s a problem, consider buying your own.

Thank You