JohnLamp.net

Search …

CMake Tutorial – Chapter 1: Getting Started →

CMake Tutorial

Posted on 2013-03-28 by John Lamp

Contents

Introduction

What is CMake?

According to CMake's creators, Kitware, CMake is an open-source cross platform build system. This is not completely accurate as CMake is not actually a build system. What CMake provides is an easy way to build C/C++ projects across platforms. The reason I say that CMake isn't a build system is because it doesn't actually build software. "A build system that *doesn't* build software?" you ask. Yes; what CMake does is generate a configuration for your existing build system, e.g. Make. This allows CMake to focus on things that most build systems don't; such as cross platform configuration, dependency calculation, testing, packaging, and installation.

Why CMake?

By not being a true build system, per se, CMake allows for a more flexible development environment as it can generate Makefiles or projects for a variety of IDEs. This allows developers to easily work on different platforms using different tools since one can build using Microsoft's Visual Studio on Windows or with GNU Make on Linux just as easily.

CMake also includes tools for finding libraries, e.g. boost, and the ability to easily include external projects in your build. These two features, in particular, make it much simpler to build projects that have external dependencies and by using the find tools rather than hard coding paths it is much easier for new developers to get started on an existing project.

Included with CMake is CTest, a test driver program. Both work together to make it easy to run a project's test programs and/or scripts. When you configure your project you specify how to run your tests and CMake generates a configuration for CTest. CTest will run all of your tests and provide a summary of which ones passed and which ones failed. In addition it logs the output of all the tests it ran. Optionally CTest can be directed to run only specific tests or skip specific tests, perhaps the slow ones. While it may not be a continuous build system you have most of the components provided.

In addition to setting up a build CMake can also create an install target that will install the outputs of your project in the appropriate locations. Once you have configured your project to be installed you can also package your project using the included CPack utility. A variety of packages can be created including tar files, zip files, or an installer.

Acknowledgements

I would like to thank the following people for their help and contributions to this tutorial. Without them it would not exist.

Devin Ronge
: This tutorial would not exist without Devin, he suggested I write it and motivated me to start. Despite being primarily a C# and JavaScript developer Devin has read every word of this tutorial at least once. Thanks to him you get a better written tutorial than you would had he not proof-read it first.

Steve Rieman
: As a C++ developer who actively uses CMake Steve has provided a technical review of the sample code in addition to a review of the prose. He has also provided numerous ideas for the contents of this tutorial.

This entry was tagged CMake, long, tutorial. Bookmark the permalink.

CMake Tutorial – Chapter 1: Getting Started →

One thought on “CMake Tutorial”

2014-01-12 at 16:48:17

Hamid said:

Thanks for the great tutorial, IMO this is the missing manual for those who really want to use CMake in a non-trivial project and to take advantages of its cross-platforms capabilities.