A Cross Platform Development Workflow for C/C++ Applications.

Martin Wojtczyk and Alois Knoll
Department of Informatics
Robotics and Embedded Systems
Technische Universität München
Boltzmannstr. 3, 85748 Garching b. München, Germany
wojtczyk@in.tum.de

Abstract

Even though the programming languages C and C++ have been standardized by the American National Standards Institute (ANSI) and the International Standards Organization (ISO) [2, 15] and - in addition to that - the availability of the C library and the Standard Template Library (STL) [26] enormously simplified development of platform independent applications for the most common operating systems, such a project often already fails at the beginning of the toolchain – the build system or the source code project management.

In our opinion this gap is filled by the open source project CMake in an excellent way. It allows developers to use their favourite development environment on each operating system, yet spares the time intensive synchronization of platform specific project files, by providing a simple, single source, textual description. With KDE4, CMake was introduced to a very popular project [28]. In this article we propose a workflow to ease the development of cross platform projects and we show, how we used CMake to create an OpenGL application as a demonstrator for a windowed application running on Windows, Linux and Mac OS X as well as a platform independent camera interface as an example for hardware dependent cross platform applications.

1. Introduction

Due to the standardization of C and C++, applications which can be compiled on multiple different platforms, can be easily implemented. On Windows platforms, given a source file like the very simple "Hello world!" application in Listing 1, the translation however requires the manual creation of a Visual Studio project file referencing the source file [22]. On Macintosh computers, people often are used to the Xcode IDE, where the developers need to create the necessary Xcode projects, which reference the source [3]. On Unix/Linux systems developers often use the GNU Autotools or even write Makefiles manually [29, 10].

```
#include <iostream>

using namespace std;

int main(int argc, char** argv)
{
   cout << "Hello world!" << endl;
   return 0;
};
```

Listing 1: hello.cpp

At a final stage of a cross platform application, the developers may just provide project files for the different platforms, but in most cases a software project is continuously being maintained and as soon as new classes or modules are added to a project or as soon as multiple engineers co-operate on a project, developers desire a tool, that supports synchronizing the different Visual Studio- or Xcode projects as well as the Unix Makefiles. The open source project CMake supports developers to manage this kind of projects with simple textual project descriptions, out of which generators provide platform specific project files. Table 1 summarizes the project generators of the current version 2.6.0 for the Windows, Mac OS X and Unix/Linux operating systems. As you can see, a wide variety of development environments is supported on every platform, for example Eclipse with the C/C++ Development Tooling (CDT) extension and even KDevelop 3 or CodeBlocks on each of them in addition to the previously mentioned ones [7, 17, 5].

2. Application

CMake can be downloaded as source code or as installable executable for Windows or as precompiled binaries for

978-0-7695-3372-8/08 $25.00 © 2008 IEEE
DOI 10.1109/ICSEA.2008.41

Windows	Unix/Linux	Mac OS X
Borland Makefiles	Unix Makefiles	Unix Makefiles
MSYS Makefiles	CodeBlocks - Unix Makefiles	Xcode
MinGW Makefiles	Eclipse CDT4 - Unix Makefiles	CodeBlocks - Unix Makefiles
NMake Makefiles	KDevelop3	Eclipse CDT4 - Unix Makefiles
Unix Makefiles	KDevelop3 - Unix Makefiles	KDevelop3
Visual Studio 6		KDevelop3 - Unix Makefiles
Visual Studio 7		
Visual Studio 7 .NET 2003		
Visual Studio 8 2005		
Visual Studio 8 2005 Win64		
Visual Studio 9 2008		
Visual Studio 9 2008 Win64		
Watcom WMake		
CodeBlocks - MinGW Makefiles		
CodeBlocks - Unix Makefiles		
Eclipse CDT4 - MinGW Makefiles		
Eclipse CDT4 - NMake Makefiles		
Eclipse CDT4 - Unix Makefiles		

Table 1. Available project generators for the supported operating systems.

Mac OS X, Linux and several Unix systems for free at [18]. Furthermore packages for many Linux distributions, the MacPorts- and the Fink-project for Macintosh users are provided in their repositories for convenient installation and automated updates [21, 9]. By default the precompiled Windows package comes with a Qt based GUI to ease the setup of initial settings for the project under development, while the Unix/Linux and Mac OS X versions of the precompiled package come with a ncurses based console user interface application `ccmake` (see Figure 1, 2nd row) [27, 11]. If built from source, a Qt based GUI and/or a ncurses based console user interface will be built on each platform, provided that the necessary libraries Qt and ncurses are installed and accessible by the compiler. Subsequent project updates can be generated by the utilization of the `cmake` command on each platform.

```
SET(SRCS main.cpp)
ADD_EXECUTABLE(hello ${SRCS})
```

Listing 2: CMakeLists.txt

Listing 2 shows a simple project description for the previously mentioned ”Hello world!” application. Invoking CMake with this description will produce a project for the desired development environment on each supported platform. A subsequent build process in the individual developer’s favourite environment will then build a native application for this platform.

To build an application out of the two files `hello.cpp` and `CMakeLists.txt`, the path to the project description in the CMakeLists.txt file is passed to `cmake`, an optional parameter `-G` can specify the desired generator. Calling `cmake --help` lists all possible parameters and the supported generators on the running platform.

On Unix/Linux/Mac OS X the build process is performed by the calls in Listing 3, since per default Unix Makefiles are generated.

```
hello$ cmake .
hello$ make
```

Listing 3: Building the ”Hello world!” application on Linux/Unix and Mac OS X.

On Windows the commands in Listing 4 perform the same, provided that Visual Studio is installed. If installed, also the GUI application or the ncurses based console user interface application can be used to create the project files. As an alternative to Listings 3 and 4 you can also generate a Visual Studio Solution, a Xcode project or a KDevelop 3 project as desired and invoke the build process within the IDEs as usual. The only important prerequisite is, that the necessary dependent tools (make, nmake, gcc, link, cl, ...) are available on the command line, thus the environment variables are set properly. A very useful feature is the out of source build, which means that the source code of a project stays separated from the created project files, compilations and possibly generated temporary files to avoid the accidental submission of platform specific files to the sourcecode

repository.

```
hello$ cmake . -G "NMake Makefiles"
hello$ nmake
```

Listing 4: Building the "Hello world!" application on Windows.

2.1. Project Description

The project description of the "Hello world!" example only defines a variable `${SRCS}`, which references the necessary source code files. The instruction `ADD_EXECUTABLE` notifies CMake to create a `hello` executable target out of the source files passed as an argument. In a similar way static or dynamic libraries can be created by the `ADD_LIBRARY(libname [SHARED | STATIC] sourcelist)` instruction. A complete list of the CMake instructions can be found in the online documentation at [19]. In contrast to the GNU Autotools - which are only available for Unix environments - complex projects can be described and maintained after a very short learning phase already.

Often software systems need to access external libraries or connected hardware, which is accessed differently on different operating systems. Sections 2.2 and 2.3 describe both scenarios.

2.2. Project Configuration

Since nowadays software projects often rely on external libraries, one of the most necessary features of a build system is the support of the configuration step, hence finding necessary header and library files of external packages. CMake simplifies this step by providing many Configuration scripts for the most common libraries. However reusable configuration scripts can also be easily implemented by the user himself as described in the documentation. If, for example, a cross platform application should not be restricted to the command line, you can easily utilize one of the supported GUI toolkits (Qt3, Qt4, GTK+, wxWidgets, FLTK etc.) [13, 16, 8]. This way impressive, windowed cross platform applications can already be implemented just by choosing the right external libraries.

To inspect 3D models on every operating system in the same environment and to establish a software basis for animated simulations for future use, qiew, a demonstrator and test-bench, was implemented. Using the Qt toolkit and Coin3D, a reimplementation of SGI's Open Inventor, a platform independent VRML viewer was created (see Figures 1, 4th row) [20, 25]. The source code of the application is available at its website [30]. In the source we also contributed configuration scripts to find the Coin3D libraries on Windows, Linux and Mac OS X.

2.3. Resolving Platform Characteristics

When it comes to hardware access, software engineers often have to deal with different Application Programming Interfaces (APIs) on each operating system.

As part of a unified, vision based tracking library, we needed to implement a platform independent camera interface, which was put into effect by applying common software engineering methods [24]. In this case we utilized the `AbstractFactory` Design Pattern as described in [12] to encapsulate platform specific code in specialized classes derived from an abstract interface class, which provides the function interfaces for the end user. Figure 2 shows the implemented UML class hierarchy with a subset of the implemented methods.

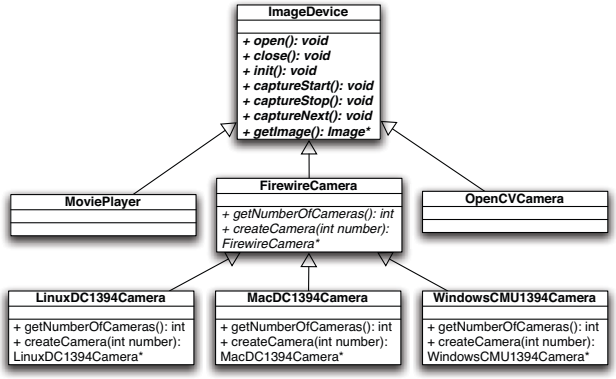

Figure 2. A Cross Platform class hierarchy for unified camera access.

Video sources which we defined as cameras can be recorded movie sequences, USB web-cams out of which many are supported by the open source vision library OpenCV [14] or Firewire cameras. Since most Firewire cameras support the Instrumentation & Industrial Digital Camera (IIDC) Standard [1], they are frequently used in academia. Therefore enhanced support for Firewire cameras was implemented to make use of the standardized access to relevant registers such as white balance, gain or shutter speed.

On Linux and Mac OS X access to Firewire cameras is provided by the commonly used library libdc1394 [6] while the CMU 1394 Digital Camera Driver [4] provides a similar functionality for Windows, yet with a completely different interface. We contributed the necessary configuration scripts to find the libdc1394 library on Linux and Mac

OS X and the CMU 1394 Digital Camera Driver on Windows. Furthermore we contributed a software package providing a uniform programming interface for the platform specific APIs.

2.4. Deployment

Once an application or library is built, it is usually packaged for distribution. While Windows packages mostly come as installable setup files, Unix/Linux packages are often Tarballs or self-extracting shell-scripts and Mac OS X packages usually come as DMG disk images, which directly contain the binaries or an installer package. By the simple use of an `INCLUDE(CPack)` directive in the CMakeLists.txt file, a package target will be generated in the project file and all files, which are tagged with an `INSTALL` command will automatically be added to the appropriate deployment package, when invoked. Table 2 summarizes all package generators. The generators STGZ, TBZ2, TGZ, TZ and ZIP are available on every platform, provided that the necessary packaging tool is available and create archives with the necessary binaries and/or sources if tagged for installation. The NSIS generator creates an installable windows package based on the Open Source Tool: Nullsoft Scriptable Install System (NSIS) [23]. The generated and installed package will also show up in the system wide list of installed Programs and provides an uninstaller for clean removal. The DEB and RPM generators are available on Linux machines to build commonly used Debian- (DEB) or Red Hat Package Manager (RPM) packages which can be installed and removed easily. The OSXX11 and PackageMaker generators are only available on Macintosh systems and provide native installers for this platform.

3. Workflow

Figure 3. The Cross Platform Workflow and its involved Tools.

To sum up, we propose the following workflow for cross platform applications or software components. In the first place, developers get the current source code and project descriptions from a source code and version management system such as Subversion or CVS, no matter which operating system they work on. Afterwards they generate native project files for the development environment, which they prefer to work with by the use of the CMake project generator. From within their favourite IDE they contribute to the project and, if necessary, to the project description files, which are committed back to the source code management system, once a goal is achieved. For testing, the native build processes are invoked from within the IDE. In addition to that, packaging for deployment can be performed automatically on each supported platform to reduce the effort of application bundling for every new version. Figure 3 shows the workflow depicting the project's states in boxes and the utilized transformation tools and their impact directions as arrows. Figure 1 depicts the workflow for the previously mentioned cross platform application in section 2.2 at its different stages which was developed for Windows, Linux and Mac OS X.

4. Future Work

One feature we were missing and hence are working on ourselves is the automatic generation of the textual project descriptions by scanning an application's directories for source files and inspecting it's `#include` directives. That will ease the migration of software projects towards cross platform support dramatically from our point of view.

5. Conclusions

It is true, that the proposed workflow requires software engineers developing C/C++ applications to learn an additional tool, on the other hand doing so could make their applications available for many more users using different operating systems. And actually the learning curve with CMake is not as steep as with the GNU Autotools by far. In some software projects the co-operation of users of different platforms is just inevitable. At the chair for Robotics and Embedded Systems at the Technische Universität München a majority of the vision group prefers to implement their algorithms on Windows, while the robotics group usually uses Realtime Linux on their robot platforms, where the vision algorithms are put into effect. This was the initial reason to create a workflow connecting both worlds.

The authors are not affiliated in any way with CMake or the company behind this open source tool. However CMake was the most promising out of several other tested tools during an evaluation by the author in 2004 and has been used in very many cross platform projects at the chair since then. Furthermore it has been introduced in two spin-off companies of the chair due to the many advantages: from project generation and synchronization, to configuration and dependency resolving and to deployment packaging, as mentioned above. Still, many developers don't know the power of this tool. This is the reason, why we want to share our

Generator	Description	Windows	Unix/Linux	Mac OS X
NSIS	Null Soft Installer	+	−	−
DEB	Debian packages	−	+	−
RPM	RPM packages	−	+	−
OSXX11	Mac OSX X11 bundle	−	−	+
PackageMaker	Mac OSX Package Maker installer	−	−	+
STGZ	Self extracting Tar GZip compression	+	+	+
TBZ2	Tar BZip2 compression	+	+	+
TGZ	Tar GZip compression	+	+	+
TZ	Tar Compress compression	+	+	+
ZIP	ZIP file format	+	+	+

Table 2. Available deployment package generators for the supported operating systems.

experience with the community. The utilization within well-known projects such as KDE4 or OpenSceneGraph may raise its popularity. We showed, it can be easily integrated in the development process on the most popular operating systems, still letting the developer choose his favourite environment, however more important than that, we showed, it can be used very well as the missing link in managing cross platform projects.

References

[1] 1394 Trade Association. *IIDC 1394-based Digital Camera Specification*, 2000.

[2] American National Standards Institute. *ANSI X3.159-1989 "Programming Language C"*.

[3] Apple Inc. *Xcode*. http://developer.apple.com/tools/xcode/.

[4] C. Baker. *CMU 1394 Digital Camera Driver*. http://www.cs.cmu.edu/~iwan/1394/.

[5] The Code::Blocks Team. *Code::Blocks – The open source, cross platform, free C++ IDE*. http://www.codeblocks.org/.

[6] D. Douxchamps. *libdc1394: The API for IEEE1394 / Firewire cameras*. http://damien.douxchamps.net/ieee1394/libdc1394/.

[7] The Eclipse Foundation. *Eclipse – an open development platform*. http://www.eclipse.org/.

[8] B. S. et al. *Fast Light Toolkit (FLTK)*. http://www.fltk.org/.

[9] The Fink Team. *Fink*. http://www.finkproject.org/.

[10] Free Software Foundation, Inc. *GNU Make*. http://www.gnu.org/software/make/.

[11] Free Software Foundation, Inc. *ncurses*. http://www.gnu.org/software/ncurses/.

[12] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, 2003.

[13] GTK+ Team. *GTK+*. http://www.gtk.org/.

[14] Intel Corporation. *OpenCV – Open Source Computer Vision*. http://opencvlibrary.sourceforge.net/.

[15] International Standards Organization. *ISO/IEC International Standard 14882, Programming Languages – C++*.

[16] Julian Smart et al. *wxWidgets*. http://www.wxwidgets.org/.

[17] KDevelop Team. *KDevelop – an Integrated Development Environment*. http://www.kdevelop.org/.

[18] Kitware Inc. *CMake - Cross-platform Make*. http://www.cmake.org.

[19] Kitware Inc. *CMake documentation*. http://www.cmake.org/HTML/Documentation.html.

[20] Kongsberg SIM AS. *Coin3D – 3D Graphics Developer Kit*. http://www.coin3d.org/.

[21] The MacPorts Project Team. *The MacPorts Project*. http://www.macports.org/.

[22] Microsoft Corporation. *Visual Studio Developer Center*. http://msdn.microsoft.com/en-us/vstudio/default.aspx.

[23] NSIS Team. *Nullsoft Scriptable Install System (NSIS)*. http://nsis.sourceforge.net/.

[24] G. Panin, C. Lenz, M. Wojtczyk, S. Nair, E. Roth, T. Friedlhuber, and A. Knoll. A unifying software architecture for model-based visual tracking. In *Image Processing: Machine Vision Applications. Edited by Niel, Kurt S.; Fofi, David. Proceedings of the SPIE, Volume 6813, pp. 681303-681303-14 (2008).*, volume 6813 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, Mar. 2008.

[25] Silicon Graphics, Inc. *Open Inventor*. http://oss.sgi.com/projects/inventor/.

[26] Silicon Graphics, Inc. *Standard Template Library Programmer's Guide*. http://www.sgi.com/tech/stl/.

[27] Trolltech. *Qt Cross-Platform Application Framework*. http://trolltech.com/products/qt/.

[28] T. Unrau. *The Road to KDE 4: CMake, a New Build System for KDE*, 2007. http://dot.kde.org/1172083974/.

[29] G. V. Vaughan, B. Elliston, T. Tromey, and I. L. Taylor. *GNU Autoconf, Automake, and Libtool*. Sams, 2000. http://sources.redhat.com/autobook/.

[30] M. Wojtczyk. *qiew – a minimalistic and portable VRML/Inventor Viewer*. http://www.qiew.org.

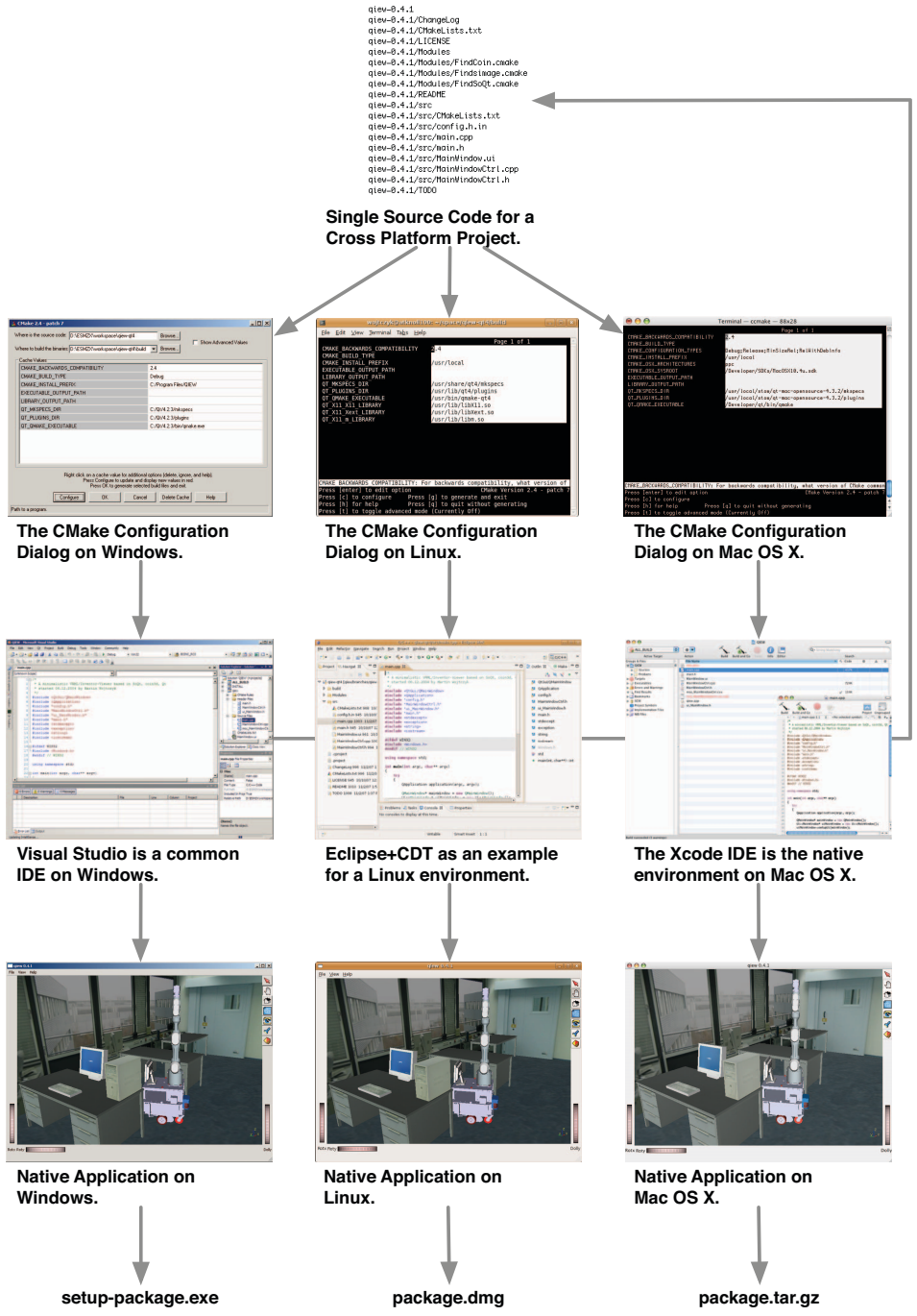

Figure 1. Exemplary workflows depicted for the development process on Windows, Linux and Mac OS X platforms. The first row symbolises the single source for the cross platform application. The configuration step in the second row shows the Qt-based user interface of CMake on Windows and the ncurses-based application ccmake on Linux and Mac OS X. In the depicted case, Visual Studio was used to build the native Windows application, eclipse was used for Linux and Xcode for Mac OS X. Changes can be commited directly to the source code repository from within the IDEs. The result of the build process is a native application on each system, which optionally can be packaged automatically for deployment.