C++ Across Multiple Platforms

A Guide to Cross-Platform Programming

[picture here]

C++ Across Multiple Platforms

A Guide to Cross-Platform Programming

Library of Congress Cataloging-in Publication Data

Peavy, Matthew D.
..... C++ Across Multiple Platforms : A Guide to Cross-Platform Programming
..... QA76.xxxx

ISBN 0-xxx-xxxxx-x

Dedication

Summary of Contents

Table of Contents

0

Preliminary Issues

This chapter discusses a few necessary issues to get you started with this book. Please take a moment and read through this chapter before skipping ahead.

0.1 Welcome

Welcome to *C++ Across Multiple Platforms*, a book designed to help you.

This is a non-traditional book in that it is a community supported project. The book is available as a PDF download, and it will be available in print form when finished.

This chapter contains preliminary information that is relevant to the book and its continued development – meta-information of sorts. It is strongly recommended that you read the following short sections before continuing to the first chapter.

0.2 Version

The version of the book you are reading is 0.1.0a.

The date of publication for this version is 2007 Nov 26.

This information is important when checking out the Updated Information and Links Section (see below).

An MD5 hash code for the PDF version is available on-line to verify authenticity and fidelity to the original.

0.3 Copyright and Distribution of this Book

This book is under US copyright. In addition, it is available as a free download from www.givemefish.com for personal use.

Personal use, as defined here, means use within a private, academic, or non-profit organizational setting. In other words, use of this book within a non-business setting is free. However, if this book is used at work, within a business, or in the scope of a for-profit project, it requires that the reader purchase a copy.

This book falls under the same restrictions as other copyrighted works with regards to redistribution and copying. Limited copying for academic reasons (e.g., handouts in class) is permitted of any copyright work under the fair use doctrine.

This book may not be copied or distributed (in electronic or print form). Although I allow free downloads from my website, this does grant you the right to distribute or mirror this book without permission.

I am making this freely available to the software development community, without cost, for personal use. Please respect my rights regarding copying.

0.4 Register this Book

0.4.1 Why Register

Why should you register this book? There are several reasons:

- Notification of updates
- Developer survey – by filling out the developer survey, you can help …

0.4.2 How to Register

Here is how to register:

0.4.3

0.5 How to Use this Book

Each chapter in this book is fairly self-contained, meaning that it is not essential that you read this book cover-to-cover. If you are only interested in information covered in a later chapter, skip ahead.

For your convenience, a comprehensive index and glossary are

present at the end of this book.

0.5.1 C++ Sample

All C++ code samples will be in the following font in order to distinguish code from the text in the book. Also, code is indented from normal text to accentuate clarity. C++ comments are formatted in italics to distinguish them from non-comment source code. For example:

```
#include <iostream>
#include <cstdlib>

int main()
{
      //Print hello and exit successfully

      std::cout << "Hello world." << std::endl;
      return EXIT_SUCCESS;
}
```

0.5.2 Command Line

All command line user-input will be in the following format. A light gray background is used to indicate command-line environment, and user input is denoted in bold. A down-left error indicates a carriage return. The user in this example types in two commands:

```
[matthew@localhost]$ ls↵
Development  Documents  Music  GiveMeFish

[matthew@localhost]$ pwd↵
/home/matthew
```

0.5.3 User-Interface Navigation

User navigation of GUI's will follow this format:

$UPDATE THIS

Menu -> SubMenu -> Sub-subMenu

Button OK

Tab1 -> RadioButton check

0.5.4 Code Style

Code will follow style rules laid out in Sutter's book C++ Coding Standards [Sutter05]. Classes, functions, and enumerations are named `ThisWay`, variables `thisWay`, and private member variables `m_thisWay`. Public member variables are simply relegated to the trash bin.

Within functions, I have chosen to place the initial curly bracket on the same line (directly following) the programming statement For example:

```
if(result == true) {
      cout << "Result is true. Updating data.";
      updateData();
      return true;
}
else {
      cout << "Result false. Exiting without update.";
      return false;
}
```

This choice is partly personal a preference and partly to reduce the size of code examples. Although this is often leads to contentious debate, to quote from Sutter regarding various bracket placement styles, "Any professional programmer can easily read and write any of these styles without hardship." [SuttCUJ Oct 2004]

0.5.5 HyperLinks

HyperLinks within the electronic book are actually hyper. Links within the printed version appear the following way:

$UPDATE THIS

Links from one section of the book to another are only visible within the electronic version of the book and look like this:

$UPDATE THIS

0.5.6 Naming Conventions

There is a small feud within the free and open source software world regarding names. There are two main points of debate:

1) How to refer to "free" and "open source" software.

2) Whether the operating system GNU/Linux should include GNU within the name or not.

Both of these issues are semantic as well as philosophical as anyone

who has heard Richard Stallman can attest.

While not wising to entry the fray, I will refer to “free” (as in freedom) as well as “open source” software as “free and open source software” or FOSS. The practical differences between the two are largely irrelevant for the material in this book. I will therefore lump them together. Regardless of the name, a programmer must have a solid understanding of the various licenses governing FOSS packages, tools, and libraries. I would also guide the interested reader to the GNU web-site $LINK for more information on important philosophy of software freedom.

Regarding the name GNU/Linux, it is my feeling that the GNU operating system is a significant piece of GNU/Linux and should therefore be a part of the name.

0.6 How to Reference this Book

While you may have downloaded this book for free, the information contained herein should be regarded as public domain. If you use this book in a report, article, or book, it is your responsibility to cite the information used.

Part of the decision of whether to release this book as a free download stems from the desire for the widest possible distribution and dissemination of the information. Referencing this book serves the purpose of establishing it as a reputable source. Just as linking within the web enhances the richness of ..., referencing ...

- Bibliographic reference:

  Peavy, Matthew., ... 2004

- Hyperlink reference:

  http://www.GiveMeFish.com/books

Thank you for your cooperation on this issue.

0.7 Updated Information and Links

As the software industry moves at an incredibly fast pace, some of the information contained in these pages will likely be out of date by the time you read it. Therefore I maintain a section on my website for current information and links.

This information will be added to each new edition of the formal book. All “update” information will be saved for each past edition.

Corrections are welcome! Please help me keep this information up-to-

date. Nothing is worse than a dead link. When you discover a mistake, a dead link, or have a disagreement about any issue in this book, inform me at your earliest convenience. The following two sections discuss contributions.

Updated information can be found here: $UPDATE THIS

0.8 What to Contribute

There are lots of potential contributions that readers are encouraged to make. The writing and wide-spread dissemination of this book will be in large part thanks to the readers – the wonderful community of software developers.

Please note that contributions may become part of this book in the future. They will always fall under the same copyright rules as the rest of the book. Your contribution will be available identically in the future on-line and print versions. See the section on How To Contribute (below) for information on recognition.

The following items are strongly encouraged:

1) Spread the word – when discussing issues with fellow developers (whether in person or on-line), suggest this book. If a fellow developer poses a question related to cross-platform development, give him or her the link.

2) Link or reference this book – when you use the information in this book, reference it so that others will know where to find the information.

3) Send me typographical and non-current information – any time you find errors, dead-links, or programming bugs, please submit these to me.

4) Point out omissions of important tools and ideas – if you feel that I have missed an important idea, tool, programming library, or anything else, please let me know. There is a huge body of programming tools and knowledge (and I certainly do not pretend to know every tool and topic indepth). I am relying on your help to give a complete and accurate portrayal of cross—platform development.

5) Raise fundamental issues – I have tried to follow state-of-the art coding and design practices for source code and application implementation. However, if you feel that I am in error with regards to a fundamental programming concept or style, or if you think that I have recommend the misuse or misconfiguration of a

software tool, let me know.

6) Translate some or all of this to another language – the more languages we can translate this book in to, the wider its adoption will be. Programmers will gain the most from this book if they read it in a language with which they are most comfortable. If you have the skills, please consider translating some or all of this book. I will publish sub-section translations of this book as I receive them. So don't feel that you have to translate the whole book (or even a whole chapter) to contribute and see your results. Revisions and corrections of translations are also encouraged.

0.9 How to Contribute

Contributions may be made in a number of ways: anonymously, under a pseudonym, name attribution (potentially with contact information).

In all cases, the contributor must agree to the following stipulations:

- The work is original and not taken from another source; or, if derived from another source, is properly and completely attributed;
- The contributed work may be included in whole or in part in all future versions of this book without restriction;
- Any contributed code will be published under a BSD-style license;
- Code may be revised to clarify or make conforming to the standards used throughout this book. The code will be available for review by the original contributor before inclusion in order to allow the contributor to forgo having his / her name associated with it (...in other words, if you think I mangled your code to the point that you are embarrassed to have your name associated with it, I will allow you the opportunity to review and then decline attribution).

If you prefer to contribute anonymously, please indicate this in any contribution contact. You may use your real name or email address during contact with me, and I will guarantee not to reference this information in the book. In order to guarantee the authenticity or work (especially code), and to discuss any potential copyright issues, I would strongly prefer to have some means of contact with an anonymous contributor.

Pseudonym or screen name – same rules as for anonymous contributing except that I will attribute the contribution to any name or pseudonym you would like.

If you would like to be mentioned by name and / or contact information (company name, address, URL, etc.), please forward me the exact information you would like to have listed. I am, however, restricting this to text only. So please do not send photos, logos, or anything else that can not be typed.

All contributors will be recognized in the Contributions section at the end of this book.

I am using the excellent OpenOffice.org Writer (www.OpenOffice.org) to compose this book. Text contributions will be accepted in any open format, e.g., OpenOffice.org (sxw or odt), Rich Text Format (rtf), or plain text (txt). Contributions made in closed / proprietary formats will be accepted. However, I cannot guarantee that I can open such documents correctly.

Any code should be submitted in standard .cpp and .h files. They should naturally be in standard C++ and (attempt to) follow the coding style used in this book.

1
Reasons

This chapter covers some of the reasons why I decided to write this book and the importance of multiple platform development.

1.1 Reason for this Book

What is the reason behind this book? I have had numerous conversations with programmers regarding the platform-independent possibilities of C++ programming. Many technologically aware people are unaware that C++ can be a “write once – compile anywhere” language. It is assumed that Java is the main cross platform (or platform independent) programming paradigm.

This book addresses these issues in detail. It provides the reader with ready information as to the tools, means and methods, costs vs. benefits, and programming particularities necessary for successful cross-platform development. It provides reasons for adopting platform-independent standards. And it provides information for the implementation of those standards.

Many Free and Open Source Software (FOSS) projects strive for wide user adoptance. In fact, many FOSS projects gauge their success by how widely the program is implemented. So it is natural to find FOSS projects ported to many platforms. In addition, the FOSS community has a tendency towards platform inclusion rather than exclusion, and thus the desire exists from the outset to write FOSS projects in a cross platform manner.

Far from being the sole domain of FOSS projects, however, cross platform programming methods are also advantageous for proprietary and commercial projects. Many of the same reasons FOSS project

leaders would development for multiple platforms are applicable to proprietary projects. Most notably is, of course, an increase in the target audience.

Being cross platform is also a means of "programming in the future tense." This view assumes that the long-term prospects of any one platform is never guaranteed. And planning this in from the beginning may be viewed as a wise long-term business choice. To quote Scott Meyers: "Good software adapts well to change. It accommodates new features, it ports to new platforms, it adjusts to new demands, it handles new inputs. Software this flexible, this robust, and this reliable does not come about by accident. It is designed and implemented by programmers who conform to the constraints of today while keeping in mind the probable needs of tomorrow. This kind of software — software that accepts change gracefully — is written by people who *program in the future tense*." [Meyers96]

Thus this book attempts to aggregate information, advocate for the adoptance of, and aid in the implementation of cross-platform development.

1.2 Not a Java Killer

...but close. Just kidding. This book is not intended to give the impression that Java can be replaced by platform-independent C++. The discussion amongst programmers who favor different languages often elevates to a frenzied level of debate. The truth is, there is no "one true language." Java exists for specific reasons and should continue to remain a programming platform in the future.

Everyone has, however, their preferred language. And for a lot of industrial-level application programmers, C++ is that language. The mistaken belief that C++ can only target a single platform has led to more than one project being written in Java. This detrimental, since an informed decision may lead to a better implementation within C++.

With the information in this book, a C++ programmer can make the case to an employer or client that C++ can indeed be used for that multi-platform project.

In addition, if C++ has already been chosen as the language for implementation, a programmer can make the case to plan in platform independence from the outset. This is true, regardless of whether or not the current (short-term) intention is for mono-platform

deployment. Given an accurate view of the costs and benefits, an employer or client may be persuaded to take the cross platform development route even though this was not part of the original development plan.

1.3 Multiple Platform vs. Platform Independent

Originally I decided to title this book *Platform Independent Programming in C++*. I changed the name because I felt that the use of the term "multiple platforms" better reflects the focus of the book. A discussion of the difference between the two terms is justified.

Multiple-platform programming refers to programming in way that an application is either expressly written for multiple or platforms, or that it can be easily adopted for use on another platform. Multiple-platform programming may mean that some changes are necessary in order to port the program, but those changes should be relatively painless.

Platform independent programming refers to the art of writing code that runs on various platforms with essentially no change to the code. This could mean that the source code is platform independent, or the binary is independent (for example, POSIX compatibility), or somewhere in between (such as byte code execution within a virtual environment – the method of independence achieved in the Java paradigm).

Both multiple platform and platform independent programming are discussed in this book. While the latter term is more "independent" (and therefore should be the preferred goal when programming), the former is usually the only practical means for achieving multiple platform execution. Some platform particularities are impossible to abstract away, and thus platform specific tweaking is required.

A programming library is much more likely to achieve true platform independence, unless the library is heavily file system or user-interface related. Libraries should aim for true platform independence.

1.4 Client-Side Applications Live On

Will every program eventually just be a web application, cobbled together from smaller, interoperable, XML-enabled, leverageable, scalable, grid-capable, enterprise-ready browser-accessible web services? Hyperbole aside, the hype for web-services (and the related business jargon that naturally follows suit) would have you believe

that.

It is important to differentiate between web-services and web-applications. The web-services idea is to allow the creation of modular programming entities (something along the lines of a function or class object) that may be utilized over the Internet by different programs. The supporting architecture (from security, to directory look-up, to payments for use of the web-service) are provided within the .NET and J2EE application architectures. In this way, it is possible for programmers to “publish” their web-services and allow them to be used (often times with a fee) by other programs and programmers. It's something akin to renting out a function or class library which resides on, and is called from, the Internet.

Web applications are something altogether different. A web-application runs mostly on a remote server (though partially as Javascript within the local machine) and is viewed within a local web-browser. The application may be written in one or many languages (with Perl, PHP, and Java being the most common). Web applications may make use of web-services, but do not necessarily have to do so. The emergence of AJAX has brought web-applications to a sufficiently high level that they can now be considered an alternative to client side applications. Witness the success of Google Apps.

Web applications have some very appealing attributes. They are generally truly platform independent (with the exception being some Internet Explorer specific applications, especially those relying on ActiveX). They require no installation, take no client disk space, can be upgraded (on the server) without the user's intervention or attention, and can be viewed from any computer that has a browser. In addition, a well written application can be compiled as a web application running on a server or as a local application utilizing a web-browser as the user interface. Web applications and browser based UIs are discussed further in chapters $UPDATE THIS

Client-side applications will continue to exist and proliferate. There are several reasons for this prediction and I offer them here:

- Not everyone is web-connected at every moment. Although this is changing thanks to broad-band and wi-fi (and will increase dramatically due to 3G and wi-max in the future), ubiquitous computing is simply not guaranteed. This is especially true for the business road-warriors and people who work in remote areas. If they can't access their application at any moment desired (especially their critical applications), then the delay in deployment of true web-applications will be apparent.

- Not everyone wants to trust everything to a remote process. Some people have serious privacy and security reservations about their data, whether it be personal or professional. Entrusting all of that sensitive data to an off-site, 3rd party web-service “somewhere” in the world is scary to businesses and citizens alike.
- The current model is familiar and comfortable. People are often adverse to change. The current model of local applications is familiar and appears to work well. Why change this process? What advantage, asks Joe user, is there to my word processor actually existing on a server cluster in India? My lowly PC, says Joe, doesn't ever break a sweat keeping up with my two finger hunt-and-peck method of composing the morning's meeting agenda.
- Some client-side applications simply work better. Depending on the needs of the application, client-side applications may simply fulfill the needs of the user better. This is especially true where transmission of a lot of data across slow networks is occurring, where a lot of quick user-interaction is needed (low latency), where the data is largely, entirely, and often only stored on the user's PC already, etc.
- Development inertia. As with development for the ubiquitous Windows platform, many applications will continue to be developed in the manner that is most comfortable for the developer. And with the skill set largely developed for local applications (and the learning curve for many of the security, communications, and authentication issues involved in web services steep), adoption by application developers will not match the hype. It should be noted that this same argument could be used to counter the reasons for cross-platform development. It is hoped that this book will go some way towards overcoming those obstacles.
- Web-service integration difficulties. While XML is touted as the “cure all” for disparate data formats and incongruous data sets, it simply does not solve the problems that many lay people have been led to believe. Passing data between between different programs is usually a daunting task. Slight differences in the way data is stored and used means that large amounts of human intervention is needed to make sure the right data is formatted in the correct manner. Web services relies on the notion that this can somehow be automated (largely through the use of XML and schemas). While this may be the case for simple and well defined tasks (for example, returning a postal “zip” code given a city and state), any larger and more complicated task becomes exceedingly difficult. XML as a means of automated data malleability is not the panacea that many

believe. The simple fact of the matter is that useful, generic web-services will have a limited role in the construction of large, complex, custom applications. Their reusablitiy will be far more limited than people believe. Their reliability and accuracy will be an issue (what testing steps will an “enterprise level” application take to guarantee that a web-service is reliable? If the web-service provider suddenly disappears, will the “back-up” web-service provided be equally tested? Will things be guaranteed to work in an exactly identical manner?)

Given the above points, I expect the development and deployment of web-services and web-applications to be a long and slow process. The demand for local versions of the same applications will likely remain high.

1.5 Platform Proliferation

After years of consolidation for the Windows desktop platform, there finally seems to be a loosening of Microsoft's grip. The venerable Macintosh has made its comeback with OS X. GNU/Linux is on the rise. And Unix still exists and will continue to exist for a long, long time to come.

And then there are the portable and embedded systems that are multiplying faster than one can count: mobile phones, PDAs, tablet computers, and who knows what else coming down the pike.

It is probable that a useful application that currently runs only on Windows will see demand on one of these other platforms in the near future. We currently see major efforts underway to port enterprise level applications to the Mac and GNU/Linux platforms. And stripped down (mobile) versions of many useful applications will make their way to mobile devices in the near future.

1.6 Because I Am Enthusiastic

The primary reason that I am writing this book is because I am enthusiastic. I like the C++ language for development. I am excited about GNU/Linux. I dislike closed systems and formats.

Because of these reasons, I am writing this book. I hope to share some of my knowledge and enthusiasm and to help do my part to turn this tide towards more open systems and more abundant, quality software.

2

Benefits of Multiple Platform Development

This chapter covers the numerous benefits of multiple platform development. It is likely that you already have a good idea of the benefits, and this chapter will only reinforce your already-held belief. The points made in this chapter can also be invoked to persuade bosses or clients that multiple platform development may make sense for a particular project.

It is important that the benefits be weighed against the costs, which are discussed in the next chapter. After reading these two chapters, the reader should have a good idea whether multiple platform development suits the project he or she is working on.

2.1 Benefits for the Developer

The benefits of multiple platform development are numerous. We'll start with the immediate benefits that the developers will see.

2.1.1 Favorite Platform

First of all, developers have their own favorite platform. Those who have developed on Windows for a decade may never want to leave the sheltered environment of Microsoft Visual Studio. However many developers dream about being able to boot into GNU/Linux at work and unleash the power of Emacs on their code. While others have taken to the Macintosh as a preferred platform.

If keeping the employee happy is one of the important jobs of a good manager, then letting a programmer choose a platform is probably

one of the fundamental keys to that happiness.

2.1.2 More Tools

There exist a lot of tools that are platform specific. While many open-source projects have been ported across several platforms, there exists a large number of proprietary tools which are platform specific. Windows has an extremely large number of development tools. Increasingly these are tools that work with the Visual Studio compiler.

2.1.3 Larger Test Audience

When you want to distribute your application for testing, whether in-house or in the wild, the more platforms you target, the greater the testing.

2.1.4 More Thorough Testing

Multi-platform testing will stress your application in ways that you may not intend. And this is a good thing.

For example...

2.2 Benefits for your Employer or Client

While it is fun to concentrate on the benefits that are directed to the employee, in for-hire projects, it is really the employer or client who must make major design decisions. Thus the employer or client must be convinced that it is in his or her best interests to go cross-platform.

2.2.1 Larger User Base

Obviously the more number of people that can buy your product, the better chances for success. Rise of GNU/Linux, Embedded, ???

There is a second important consideration when discussing platforms. Some users will prefer one application over another if they have the opportunity to use it on multiple platforms. This is true of people who dual-boot in GNU/Linux and Windows, as well as people who have a different platform at work than at home (think Unix at work and Windows at home, or Windows at work and Macintosh at home). And with the advent of virtualization, many people are running two operating systems simultaneously.

The decision to purchase one piece of software over another can easily be tipped by the fact that it is available for more than one platform.

In Appendix $LINK, I include a list of software that can be used for various purposes that is explicitly cross-platform. This isn't development software. This is regular productivity software. The decision to use an application from this list may be made based solely on basis that you can use this software on multiple platforms.

So playing the “installed OS” numbers game isn't the most accurate way to gauge potential application adoption.

2.2.2 Better Image

A product that lists multiple platforms takes on a level of quality and maturity in many peoples' eyes. Why is this so? Traditionally, software targeted a single platform and was then ported after it was determined that the market was established on another platform. Recently it generally moves out from Windows to the other platforms. Within the open-source world, it may move in the opposite direction, starting as GNU/Linux only and ported to Windows. For scientific programming, applications may have their roots in the various flavors of Unix before moving away. And graphic design and multi-media applications have always been a strong point for the Macintosh.

The porting generally happens only after a maturation period for the application. When it finally reaches the multi-platform state, it is usually a polished product.

In addition, a growing number of computer users feel strongly that their platform should be supported. This is particularly true of the GNU/Linux crowd, and to a lesser degree the Mac crowd. There is a halo effect achieved by software distributors who have made the effort to compile their software for a minority platform.

2.3 For the Sake of Standards

According to the old joke: The great thing about standards is that there are so many to pick from. In reality though, standards plan an essential role in defining a

Even if the application is never ported, it is

2.4 For Future Compatibility

Predicting what the next big platform will be is a fool's errand. At the same time, we know that well written code lasts a long time. Thus the most sensible decision we can make about future compatibility is to write code that is as platform-independent as possible.

After this has been exhausted, we should aim to write code that is as cross-platform as possible. Finally, any resulting platform dependencies should be isolated and marked as such. This is the best possible way to future proof your code.

3

Costs of Multiple Platform Development

Multiple platform development does not happen by itself. It takes planning, implementation, and testing. This additional effort is manifested in cost. Costs may be monetary, which is the case in the traditional software development world. Or the costs may come in the form of increased development time.

Additional costs may be incurred in maintaining software licenses for the various platforms and programming tools. Platform maintenance and administration must also be included.

The costs of multiple platform development are substantially reduced when the project is properly planned from the beginning. Many of the design decisions that enable cross-platform development do not cost anything additional themselves. Not choosing this path from the beginning, however, can lead to significant costs if the program must be retro-fitted to span multiple platforms.

An honest analysis of costs vs. benefits of multiple platform development is essential for any project. This chapter will aid the development team and/or management to evaluate these costs.

3.1 Costs of Hardware and Software

One of the easiest costs to quantify is the need for additional software licenses. The licenses required for development may be extensive and expensive or few and free.

The development environment and supporting development tools for another platform may need to be purchased. In additional, licenses

for programming libraries that you use may incur fee per platform.

In addition to the “one-time” costs of hardware and OS acquisition, the most modern version of an OS is often the target platform. This requires a future commitment to software upgrades that cannot be forgotten.

If new hardware and operating systems need to be purchased, this may be accounted for as a cost of the cross-platform development requirements. However, the hardware may have additional benefits and uses, and the cost may therefore be spread across different projects or subsumed into overhead or capital expenditures.

Virtualization can help minimize the cost of new hardware, often at the cost of increasing the software budget. A VMWare license may cost far less than the price of a new computer, but it still does cost.

3.2 Costs of Platform Administration

Simply maintaining two or three different platforms can become an expensive proposition for small software shops. The skills needed to learn administration skill for different OS's is often significant.

In addition to the learning curve, the act of installing two (or more) sets of development software, two sets of productivity software, and patching two OS's for security updates is significant.

Very small software companies may leave this task to their own employees. If each employee is responsible for administering two of everything, then cost will be a significant percentage indeed.

Fortunately, system administrators can be hired for these purposes. And knowledge of heterogeneous systems is becoming essential.

3.3 Costs of Additional Development Work

The largest overall cost of cross-platform development may be the additional development work required. However, the cost of additional development work is likely to be less than half of the overall cost. The other costs presented in this chapter likely add up to be greater than development work.

First off, two code lines must be maintained. This results in extra back-ups and additional disk space. The amount of overhead involved in tracking two code lines will affect all developers and especially the manager of the project.

3.4 Costs of Additional Testing

As a percentage of project budget, testing has been gaining since the dawn of programming. Early projects were simple enough not to justify large amounts of testing. But multiple programmers and large, complex projects have demonstrated the need for significant testing resources. Many new projects have settled on 1/3 of the total budget just for testing alone $Reference.

The cost of testing for additional platforms cannot be forgotten. While a large part of the computational aspects of a project should be independent of the platform, most all of the UI and OS calls will need to be tested.

3.4.1 Unit Testing

Unit tests are stand-alone tests that test one particular feature of the application. These generally test functionality outside of the UI. These should largely be unaffected by cross-platform development. Properly written cross-platform code should test equally well under multiple OS's.

3.4.2 UI Testing

This area of testing will be the most time intensive. UI testing is already a difficult topic that is heard to automate. Must of the work is done by hand, sometimes according to test scripts. UI testing is also an essential piece of cross-platform development since so much of the platform uniqueness exists within the UI layer.

Some test frameworks exist for UI testing. These should be used if possible. More information is given in the chapter on testing.

Finally, the choice of a solid UI toolkit will aid the consistency of UI development across multiple platforms, thereby minimizing the testing needed.

3.4.3 Automation is Key

One of the keys to being able to successfully develop cross-platform software is to automate as much of the testing procedure as possible. As stated above, Unit Testing should be automated as completely as possible. Any UI testing automation that is possible should be implemented.

3.4.4 Platform Specific Testing

There may exist areas of the code which are totally platform specific. When one OS handles some functionality completely differently than another, this portion of code should be tested explicitly. This will lead to two different test cases. This, in turn, leads to separate lines of code for the test suite. As discussed in the Costs of Additional Development Work above, maintaining two or more separate lines of codes increases the cost of the project.

3.5 Costs of Additional Deployment Mechanisms

Just as there are additional development and testing costs, deployment is often times an overlooked and under-valued issue. Installers for different systems may need to be written. This is compounded by the problem that installation is extremely platform specific. Thus there is oftentimes little re-use from one platform's installation to another.

3.6 Costs of Maintaining Projects

Once a project has been written and released, it rarely remains static for long. A release if usually followed by bug reports and then patches. A client may request additional functionality or small modification yielding different behavior. This is a normal part of the software development cycle.

Any change on one platform must be implemented on the others. In well designed cross-platform code, most such changes will only need to be made in one place. But occasionally multi code lines will have to be updated separately.

In addition, the changes must be tested on each platform. This will occur regardless of whether the original implementation was in one or more code lines.

Thus the additional costs due to on-going maintenance cannot be ignored.

3.7 Costs of Submitting Source Code Due to FOSS Licenses

When a developer modifies code within a FOSS library that exists under certain licenses such as the LGPL, the developer is required to submit those changes back to the original project. Other licenses (such as the GPL) have more stringent requirements for making the

source code available. The submission or distribution of source code has two costs.

3.7.1 Costs In Terms of Time Spent Submitting or Distributing

The developer may be required to expend time submitting code back to the original project. This may seem like a trivial task, though it does take time. One may have to register for an account (on SourceForge, for example) or obtain a CVS login and password. The developer may have to submit the code in a certain manner, which requires some reading.

Perhaps portions of the project must be made available on your website (as per the GPL). Someone is required to create the web pages, make the code available (perhaps via CVS or Subversion, or simply as a zip file).

And these tasks must be undertaken each time a release of the software is made where the FOSS library has been modified. Thus some bookkeeping overhead is required.

3.7.2 In Terms of Lost Exclusiveness of Code

Releasing code under a FOSS license could be viewed (especially be management or accounting) as a loss of the exclusiveness of the code. While you as a developer may view this act as a good thing, you may be surprised by reluctance of the higher-ups in your company.

3.8 Writing 90% Cross-Platform Code is Also Good

Is the process of cross platform code writing an all-or-nothing endeavor? The answer to this is no.

3.8.1 Easy to Plan Ahead

It is easy to follow most or all of the steps in this book while only developing for one platform. This means planning for eventual cross-platform distribution at some point in the future, but not making the up-front commitment to constantly develop on more than platform.

In doing so, you obtain several of the benefits mentioned in the previous chapter without a lot of the costs mentioned in this chapter.

3.8.2 Path of Least Resistance

In fact, this method is probably the path of least resistance for

introducing the concepts of cross-platform development into a formerly mono-platform shop. The idea would be to push for as nearly a platform-independent development as possible, without actually developing on two platforms.

The actual job of porting can be finished at a later time. This pushes the majority of the costs into the future. The costs are only incurred if it is sure that deployment on multiple platforms is desirable.

3.8.3 The Danger

The danger lies in allowing certain platform dependencies to surreptitiously invade the project. If you're not testing on multiple platforms, you may not have any idea that this happening. For example, the use (and then probable strong dependence on) a certain 3rd party library may bind needlessly to one platform. This is especially true in the proprietary world, where libraries are severely shackled by licenses and may only be released in binary form.

Once certain design decisions have become entrenched, they may be prohibitively expensive to change later. Had the developers realized (through multi-platform compilation) that a particular library was not platform neutral, an informed decision could be made at that point as to whether another library might be more appropriate. Other “evasive action” such as wrapping the interface of a dependent library might also be taken to compartmentalize the library. Pre-processor directives or “build isolation” files could later be used for compile-time platform dependencies.

UI libraries can be the biggest lock-in of all. The insulation of a program from the UI is rarely well implemented. Much of the underlying program is mingled with purely UI code such that they become inextricably linked. If the UI being used is platform dependent, then the game is largely lost. This is easily seen in the use of Microsoft's .NET Managed C++. Not only is the UI platform specific, the language extensions introduced with Managed C++ will lead to code that is impossible to easily port later.

3.9 Additional Platforms are Cheap(er)

The additional expense of writing cross platform code for a third platform is generally small compared to the expense of writing for a second. Once the program has been ported to a second platform, much of the work has been done. The second platform is proof that the code is cross-platform.

The costs associated with the third platform should involve little extra development. The additional costs will be incurred in platform management, testing, and deployment. Depending of the tools used for development, testing is likely to be the largest of these additional costs.

4

Platforms

This chapter discusses the platforms that are targeted in this book, as well as several other potential targets. Any platform that offers a standard C++ compiler is a potential target. However, the user-interface issues may limit platform choice to those which offer UI tools.

4.1 Choice of Platforms

Deciding which platform(s) to support must be made on a case-by-case basis. The costs and benefits of each must be carefully weighed. In addition, platforms should be viewed with an eye towards the future, as the industry is constantly changing. Disrupting technologies emerge quickly and, as the name implies, disrupt the status quo.

4.2 Windows

The obvious 800-pound-gorilla in the software industry, Microsoft Windows is often the “main” platform targeted in desktop software development. With a market share of upwards of 90%, neglecting Windows means losing a significant number of potential customers[1].

Windows is a somewhat nebulous term that engenders a large number of different operating systems. Up through Windows 3.1, the operating system was a 16 bit, non-multitasking operating system. Windows 95 moved to 32 bits and multitasking. Windows is also used as the name for Microsoft's server platform.

1 Customers in this sense means potential users, whether they purchase a license or obtain the software without cost.

Desktop Windows development is inherently GUI based. There are still a few console applications, which are discussed in the User Interface chapter $LINK. However, the vast majority of Windows programs are Win32 based.

The Win32 API may be accessed directly by the programmer. The direct use of the Windows API was more common with the predecessor Win16 API, largely before C++ was the dominant language. As C++ UI libraries became popular, programming to the Win32 platform largely fell out of fashion. This is due to the large over-head needed to “hand code” menus, dialogs, and controls. GUI toolkits abstracted a lot of the functionality and provided standardized methods for Windows programming.

Microsoft Foundation Classes (MFC) became the nearly de facto standard toolkit. Microsoft strongly promoted MFC and provided an extremely powerful IDE which facilitated the use of MFC. MFC included many non-standard C++ extensions and macros, thus eliminating the possibility of writing easily portable MFC code.

Borland was another major player in the Windows application development field. Their C++ Builder IDE was xxx

Development that targets the Win32 platform is not necessarily platform dependent however. Because of the abstraction layer provided by C++ GUI toolkits, a single source tree can be compiled for Win32 as well as other platforms. This requires the choice of an appropriate toolkit.

Win32 programs have an entry point called WinMain which takes the place of the normal main(). Once inside this function, no other requirements for interacting with the Windows OS are obligatory. ...

Windows is a multi-threaded operating system. Access to threading is provided in MFC as well as the other major UI toolkits. $UPDATE threading issues.

All versions of Windows since Windows 95 are 32 bit operating system. Windows XP now accommodates 64 bit architectures, and Vista (the most recent Windows release) also supports 64 bit architectures.

4.3 GNU/Linux

GNU/Linux has come on the computer scene from an unlikely source.

4.4 Mac OS X

The first computer available to the home user that boasted a graphical user interface, the Macintosh has seen its fortunes wax and wane and wax again. The Macintosh was instrumental in heralding the desktop publishing revolution and has since become the platform of choice for graphic artists.

The Macintosh has evolved greatly since its inception. Internally, the Macintosh moved from Motorola 68000 series to IBM's Power series of CPUs to the Intel x86 series. The operating system also evolved from MacOS to the current OS X, which is a Unix based operating system.

The Macintosh OS X operating system is based on the FreeBSD Unix operating system. The Cocoa UI environment is Apple's native object-oriented toolkit for the Macintosh. The theme Aqua is the current default theme within the UI.

4.5 Solaris and OpenSolaris

I differentiate Solaris from the “other Unix” variants below because of it's open source status. The platform is likely to attain a greater share of developers.

4.6 Other Unix Variants

IBM's AIX and Hewlet-Packard's HP-UX are the tw

4.7 Web or Browser Based

Browser-based applications can be viewed as a platform unto themselves. Whether the C++ portion of the program exists on the local machine and uses the browser as an interface, or whether the application is hosted on a remote server, the browser can serve as the user interface to the application.

4.8 Embedded and Portable

The embedded and portable market is the next source of large growth for computing. Mobile phones have long since surpassed PCs in numbers ($REFERENCE). Their growth is continuing to accelerate, especially in the heretofore sparsely targeted markets in the developing work such as India and China.

With so much development potential, it would be foolish not to keep

an eye on this market as a possible platform for the future of your application. Wireless connectivity is a given with these devices (the *raison d'etre* for most of them). The computational power of these devices is increasing rapidly, so as to make them viable platforms for mainstream applications already this year.

Embedded and portable platforms present their own sets of problems. Often times portable devices have small screens and limited input devices, such as a simple phone keypad. Embedded devices may have no real user interface at all. Most will face serious memory restrictions, which means that an application ported to such a platform must deal gracefully with any such memory issues. In addition, battery life may influence the emphasis placed on quick application shut down, saving intermediate or temporary data, and atomic database and OS commits.

Portable devices are also beginning to surge. A raft of small, lightly powered devices are making the presence know. These devices are showing up as tablets and palm-top computers. They lack some of the power and peripherals of laptop computers, but they support full graphical interfaces and much of the same functionality.

The mobile phone market is quickly moving towards the “smart phone” concept. Smart phones have very limited power compared to modern PCs. But they are also starting to make use of modern OSs.

The top target platforms for this segment are currently SymbianOS, Windows CE and Windows XP Embedded, Embedded Linux, PalmOS, and VXWorks ($REFERENCE).

4.9 Niche

Niche operating systems include OS/2, Windows 3.1 and earlier (including MS-DOS), mainframe OS's (such as IBM's z/OS and HP's OpenVMS), Novel NetWare, etc.

These operating systems are generally not the sole target platform of any new project. However, additions and modifications to existing programs on such platforms occur relatively often. New application development may also choose to support a niche platform at the request of a client. Because most new systems acquisitions would move away from these platforms, any development for a niche OS benefits immensely from

5

Source Code Licenses

Source code licenses come in all shapes and sizes. This topic is large enough to necessitate its own book. Many of the libraries and tools used for cross-platform development are under FOSS licenses. The probability of the use of multiple licenses within a single project is extremely high. Therefore it is essential that this topic be explored to a limited degree in this book. In addition, please see the section of Further Information at the end of this chapter.

5.1 FOSS Licenses Pervasive

As stated in the chapter 1, FOSS licenses tend to be pervasive in the area of cross-platform programming. Many tools and libraries that can be used

5.2 Review of Most Common Licenses

The following is a non-exhaustive list of the most common types of licenses. The proliferation of licenses, and the continually shifting winds of development means that licenses will continue to come in and go out of vogue.

5.2.1 Public Domain

This type of license is the simplest license. It states that the code is in the public domain and may be used for any purpose whatsoever with no restrictions. This code can be used in any manner, without attribution, remuneration, or so much as a thanks (although developer of public domain code also appreciate a thank you every once in a while.)

Many government projects are licensed in the public domain as a product of their funding source. Some hobbyists are also generous with their code and publish it under a public domain license.

Public domain code is nearly always published “as is” and without warranty.

5.2.2 Proprietary, Source Code Not Available

Proprietary licenses that do not provide source code are the most common within the commercial software world. They often come in the form of a pre-compiled library (.lib, .a, or .so file) and header files (.h or .hpp files). The header files are necessary to expose the programming interface, while the library implementation is hidden.

Proprietary licenses that restrict access to the source code are generally for pay licenses. Payment is usually required on the basis of per seat, per developer, site license, or per implementation sold.

While the access to the source code is restricted, the use of the library is often unrestricted. You are essentially paying for the right to use that library in whatever manner you want. This may be entirely appropriate (or desirable) for commercial projects. FOSS projects will likely not make use of these type of libraries, as the license may be anathema to their sense of fairness or simply incompatible with the license governing the rest of the project.

5.2.3 Proprietary, Source Code Available

Some proprietary license provide the source code to the underlying library, generally under certain restrictions. Often the library code cannot modified or used in any way outside of the library. The source code can usually never be published or passed on. Payment schemes and developers rights are generally the same under the two proprietary schemes.

The benefit of having the source code to examine (and possibly to modify) can be significant. The user may discover the cause of seemingly unexplainable behavior or may be able to modify or extend the library to accommodate some particular need.

5.2.4 Shared or Community Source Licenses

Shared source licenses have been spearheaded by Microsoft in the last several years. Sun also publishes the Sun Community Source License (SCSL), which has many similar features. Both licenses

5.2.5 GPL

The original and mostly widely used of the “free” software license, the General Public License ushered in the FOSS movement. Verison 1 of the unified GPL was published by Richard Stallman in 1989. Version 2 of the license was published shortly thereafter.

5.2.6 LPGL

The GNU Lesser General Public License

5.2.7 BSD

5.2.8 Apache

5.2.9 MIT

5.2.10 Mozilla

5.3 Restrictions vs. Freedoms

There is a grand debate within the FOSS community of restrictions and freedoms when discussing such licenses. The FSF folks feel strongly that there are certain freedoms that you have as a software user and developer, and their licenses are there to protect your freedom. The feel that by using other licenses, you are placing undue restrictions on others.

The other side of the debate (let's call it the BSD side) feels that their license provides the use with the ultimate freedom and flexibility to do whatever the developer wants with the software without the onerous requirements of publishing any source code. They view such requirements as unnecessary restrictions to the use of their code.

While this debate will not be solved here, it is worthwhile to keep in mind the two views.

5.4 Dual-Use Licensing

Certain projects are dually licensed. This gives the copyright owner's

greater flexibility in monetizing their software. The idea is to allow certain users open source access to the code under one type of FOSS license while being able to provide the code to another (paying) party without any of the responsibilities of the FOSS license. In a nutshell, a dual-use license says: if your project is open source, you can use the library for free. If you wish to keep your project proprietary, you must pay to use the library.

This type of licensing has become increasingly popular recently. The advantage for the library owner is that the source code is open and available to a lot of developers. This results in bug fixes and improvements that the owner would not otherwise have.

At the same time, the owner can license his or her software for a proprietary project and obtain some remuneration.

5.5 Responsibilities Under FOSS Licenses

You may have obligations by using or modifying a project licensed as FOSS. Depending on the license, you may be required to mention the use of the project somewhere, include the original copyright in your code, publish source code modifications that you have made to the library, or even publish the source code to your entire project.

Be aware of your responsibilities. You should not use any library for which you do not understand the ramifications of the library. The result could be contract or employment termination, a lawsuit, or the forced open-sourcing of your code.

5.6 Strategies on Explaining FOSS to Clients / Managers

If you make use of a FOSS licensed library, you will likely find yourself explaining to your manager or your client what FOSS is. Do not attempt to hide this topic or avoid having the discussion.

While the term “open source” has lost some of the stigma placed on it by leery business types, it is still something can easily sour a discussion. This is largely due to ignorance regarding the licensing details. Many people unfamiliar with software understand there to be exactly two licenses: “normal” and “open source”, and the latter is something they won't have anything to do with.

The task is difficult. One can't simply suggest that the other party is ignorant. The situation can be delicate.

I have found it best to explain what won't happen to your code. If you

are linking to a BSD or LGPL license, you can explain that none of their code will be published.

Do not forget to inform your boss or client about the costs of using FOSS either. In order for this to be a forthright discussion, a frank discussion of the costs and responsibilities under FOSS licenses must be made absolutely clear. Not doing so will likely land you in hot water later on, with a strong reluctance to use FOSS again by your boss or client.

5.7 Contracts

Contract programming involves the use of a contract program that specifies a particular application or project to be written. Contracts will spell out provisions for the projects such as minimum requirements, development schedule, target platform, etc.

The contract you sign with a client should explicitly state any responsibilities you have under FOSS licenses to re-publish code. Clients are reluctant to see any of the code they paid for showing up on a web-site, even though this may be your contractual obligation due to a FOSS license. It is imperative to

5.8 Further Information

As stated at the beginning of this chapter, the topic of software licensing is complex. The developer will likely have to delve further into this topic than can be detailed here. Further information can be found from the following sources:

- *Software Licensing Handbook,* by Jeffery Gordon [Gordon06].
- *Understanding Open Source & Free Software Licensing,* by Andrew M. St. Laurent [St. Laurent04].
- Open Source Initiative (OSI)

  http://www.opensource.org/
- Microsoft

  http://www.microsoft.com/licensing/default.mspx

  http://www.microsoft.com/slps
- Free Software Foundation

  http://www.fsf.org/licensing
- Wikipedia

http://en.wikipedia.org/wiki/EULA

- Creative Commons
  http://creativecommons.org/license/

6

Platform Specific Issues

This chapter deals with issues that relate to dealing with different platforms. The issues raised in this chapter are independent of programming, and therefore independent of the C++ language. However, the developer must be aware of these issues and handle them accordingly. The following chapter deals with C++ specific platform issues.

6.1 Prefer File-Based Preferences and Configurations

6.2 File Systems

6.2.1 Spaces and Special Characters in Names

6.2.2 Filename Length

6.2.3 My Documents, home, Mac(?), etc.

6.2.4 Case-sensitivity

Unix-based (including Macintosh) and Linux systems are case-sensitive. Windows is not case-sensitive. The compiler will thus find or fail to find included headers files based on the correct use of case within a file name.

Fortunately for compiling, this is a compile-time issue. However, files that are loaded at run-time with names that are hard-coded in source

(or read from an external data source) will encounter the a similar error. This error is much more pernicious, since the error can only be detected at run-time.

Windows case-insensitivity is compounded by its desire to “make users' lives easier” by automatically adjusting certain file names. When creating a new all-caps directory or filename in Windows, the user may be surprised to find the the name is suddenly all lower-case. Despite the user's best efforts, that filename cannot be changed back to all caps. Thus a header file whose name is an abbreviation and might logically be all-caps (for example, “GUI.h”), will be kindly renamed “gui.h” for you. Your include dependencies in Windows will function perfectly, while Unix-based dependencies will cry foul.

6.3 I/O

6.4 Sources of Platform-Dependent Behavior

Despite all of the above (and following) advice, the programmer may run into platform-dependent behavior. This may be simple to track down, or it may be as elusive as the most difficult run-time error to catch. Unfortunately, it tends to be more difficult than easy. The following lists some of the more common sources of platform dependence.

6.4.1 Order of Initialization

The order in which static and global variables is an important issue and must be treated carefully and thoroughly.

7

C++ Specific Issues

This chapter deals with issues of platform dependency as related to the C++ language specifically. As opposed to the previous chapter, which dealt with language independent differences between platforms, this chapter

7.1 16, 32, and 64 Bit Platform Issues

7.2 Determining the OS

Determining which operating system on which the program is currently executing can be of use. The more often this is needed, however, the less portable the code is. As that is the ultimate goal of this book, it should be stated that as a rule of thumb, determining the OS should be done as little as possible. This is the only sure way to guarantee portability.

7.2.1 Compile Time Determination

Compile time determination of the OS is accomplished via pre-processor definitions.

7.2.2 Run Time Determination

7.3 Platform-Specific System Calls

7.4 File Systems

7.4.1 Use Boost::filesystem

7.4.2 Case-sensitivity

Unix-based (including Macintosh) and Linux systems are case-sensitive. Windows is not case-sensitive. The compiler will thus find or fail to find included headers files based on the correct use of case within a file name.

Fortunately for compiling and linking, this is a build issue. However, files that are loaded at run-time with names that are hard-coded in source (or read from an external data source) will encounter a similar error. This error is much more pernicious, since the error can only be detected at run-time.

Windows case-insensitivity is compounded by its desire to “make users' lives easier” by automatically adjusting certain file names. When creating a new all-caps directory or filename in Windows, the user may be surprised to find the the name is suddenly all lower-case. Despite the user's best efforts, that filename cannot be changed back to all caps. Thus a header file whose name is an abbreviation and might logically be all-caps (for example, “GUI.h”), will be kindly renamed “gui.h” for you. Your include dependencies in Windows will function perfectly, while Unix-based dependencies will cry foul.

7.5 Memory

7.5.1 Memory Size

Being cognizant of the likely and (often unknown) absolute amount of memory can be important. This is especially true when dealing with porting to embedded devices, some which may not provide the emergency “pressure release value” of virtual memory.

7.5.2 Virtual

7.6 I/O

7.7 Linking and Libraries

7.8 Link vs. Binary Portability

7.9 Conditional Code and the Preprocessor

7.10 Platform-Specific Versions of Source Files

7.11 Multi-threading

7.12 Sources of Platform-Dependent Behavior

Despite all of the above (and following) advice, the programmer may run into platform-dependent behavior. This may be simple to track down, or it may be as elusive as the most difficult run-time error to catch. Unfortunately, it tends to be more difficult than easy. The following lists some of the more common sources of platform dependence.

7.12.1 Order of Initialization

The order in which static and global variables is an important issue and must be treated carefully and thoroughly.

8

Development Environment

Environment intro

8.1 Standardizing on One Environment

8.2 Emacs / vi + tools

8.3 Eclipse

8.4 Others

8.5 Issues

8.5.1 Tabs

9
Make

Make Intro

9.1 make and automake

9.2 CMake

9.3 Jam

9.4 NMake

10
Compilers

Compilers intro

10.1 Advantages of Using Multiple Compilers

10.2 Non-Compatibility Issues

10.3 gcc

10.4 Microsoft Visual Studio

10.5 Borland

10.6 On-line Compilers

11

User Interface Programming

UI intro

11.1 Console and curses

11.2 wxWidgets

11.3 QT

11.4 VCL

11.5 XUL and Browser-Based UI Toolkits

11.6 Others (FLTK)

12
Embedded Programming

Embedded Programming Intro

12.1 Current Popular Embedded Platforms

12.1.1 WinCE

12.1.2 Palm

12.1.3 Symbian

12.1.4 Linux embedded

12.1.5 ARM

12.1.6 WindRiver

12.2

12.3 Licensing Issues

12.4 Specific Embedded Issues

12.4.1 Memory constraints

12.4.2 Limited / no viewing screen

12.4.3 Real time / performance constrained

12.4.4

13
Cross-Platform Libraries

Libraries intro

13.1 Boost

13.2 Threading

13.3 Other libraries that isolate sub-systems (time, memory, etc)

13.4 Unicode

13.5 Peripheral abstraction

13.6 Microsoft's Services for UNIX (Interix)

14
Databases

Database intro

14.1 Methods of Connectivity via C++

14.2 Database Connectivity and Licensing

14.3 Cross-Database Programming

14.4 SQL Standardization and Fracture

14.4.1 Popular FOSS Databases

14.4.2 MySQL

14.4.3 PostGreSQL

14.4.4 SQLite (embeddable)

14.4.5 BerkeleyDB

14.4.6 RemStar

14.4.7 Other Fully OpenSource RDBs

14.5 Popular Commercial Databases

14.5.1 MySQL

14.5.2 Oracle

14.6 DTL as Another Layer of Portable Abstraction

15

Internet & Web Specific Issues

Internet and Web Specific Issues

15.1 Embeddable Browsers

15.2 Sockets

15.3 CGI

15.4 CORBA / MPI

15.5 SOAP

15.6 Flash and Ming

16
Graphics and Sound

Graphics and Sound intro

16.1 OpenGL

16.2 OpenSceneGraph and G3D

16.3 Ogg and MP3

16.4 Static Graphics

16.5 Browser Based Graphics

16.5.1 Vector Graphics

16.5.2 Flash

17
Programming Tools

Programming Tools intro

17.1 GUI Designers

17.1.1 wxWidgets (wxDesigner and DialogBlocks)

17.1.2 QT Designer

17.1.3 Techniques for using designers

Copying out code, member variables, documenting, directory for projects, etc.)

17.2 UML

17.3 Code Profiles

17.4 Code Generators

17.4.1 YACC

17.5

18
Concurrent Versioning

Versioning intro

18.1

18.2 CVS

18.3 SubVersion

18.4 BitKeeper

18.5 Visual SourceSafe

18.6 Git

18.7 Hosting Possibilities

19
Testing

Testing intro

19.1 Necessity for Cross-Platform Testing

19.2 Difficulty for Cross-Platform Testing

19.3 Resolution, Font, Layout

19.3.1 Sizer Based Layout

19.4 Strategies for Automated Testing

19.4.1 Non-UI (should be mostly standard C++ anyway)

19.4.2 UI – command structure

19.4.3 File system / network related

19.5 Testing tools (lint, etc)

19.6

20

Debugging and Bug Tracking

Debugging intro

20.1 gdb

20.2 Other Debuggers

20.3 Bug Catching tools

20.4 Bugzilla

20.4.1 Advantages

20.4.2 Why not Visual Intercept

20.4.3 Hosting possibilities

21

Installation and Documentation

Intro

21.1 Installers

21.2 Various Help Doc Systems

21.3 Programming Documentation

21.3.1 Doc++

21.3.2 DOxygen

21.4 DocBook

21.5 Technique for using same config files on multiple platforms

21.6

22

Web Services and Web Applications

Intro

22.1 Web Services Overview

22.2 Web Services vs. Web Applications

22.3 .NET

22.4 J2EE

22.5 Mono

22.6

23

Interfacing with Other Languages

Intro

23.1 PPP Scripts (particularly via CGI)

23.2 Java / JNI

23.3 C

23.4 Fortran

23.5 Basic (Power, VB, etc.)

23.6 Calls Through Memory

23.6.1 How-to

23.6.2 Dlls, libs

23.6.3 issues with memory

24
XML

XML Intro

24.1 Reasons for use

24.2 Why it aids in X-platform development

24.3 Data Display via XSLT

25

Emulators, Wine, and Virtualization

Intro

25.1 Not actually cross-platform development

25.2 Wine

25.2.1 lib vs. emulation

25.2.2 Acceptable for already-written programs

25.2.3 Limitations (available for Win32-based programs)

25.2.4 Darwine

25.2.5 WineX / CrossOver

25.3 Other Emulators

25.4 Virtualization

25.4.1 VMWare

25.4.2 Parallels

25.4.3 VirtualBox

25.4.4 Linux Virtualization

25.4.5 Connectix (Win95/98 on Linux)

A

Cross-Platform Libraries and Web Sites

Intro

A.1 Libraries

A.2 Web Sites

A.3

A.4

B

Cross-Platform Productivity Applications

In order to be able to work comfortably on two platforms, not only does one need the use of programming tools, but the rest of the work environment must also be present on multiple platforms. This appendix outlines non-programming applications that can be used on multiple platforms.

B.1 Office

B.2 Internet and Mail

B.3 Graphics

B.4

C

Selected FOSS Cross-Platform Applications

This appendix provides the user with selected FOSS applications that make use of some of the libraries included in this book. This allows the user to inspect the source code for cross platform applications.

C.1 Office

C.2 Internet and Mail

C.3 Graphics

C.4 Audio

D
Where to Obtain Help

Where to obtain help.

D.1 Books

D.2 Web sites

Bibliography

Sutter05: Herb Sutter and Andrei Alexandrescu, *C++ Coding Standards*. Addison-Wesley, 2005.
Meyers96: Scott Meyers, *More Effective C++*. Addison-Wesley, 1996.
Gordon06: Jeffery I. Gordon, *Software Licensing Handbook*. Lulu.com, 2006.
St. Laurent04: Andrew M. St. Laurent, *Understanding Open Source & Free Software Licensing*. O'Reilly, 2004.

Contributors

The following people have contributed to the creation and distribution of this book. The contributors are loosely ordered by the chronological order in which their contributions were received (not necessarily the importance or my valuation of the contribution!)

I would like to once again thank all the contributors to this project.

Franz Von Asche

During a dinner in Paris discussing some technical issues, it was my conversation with Franz regarding the cross-platform possibilities of C++ that sparked the idea to write this book.

Jean Hollis Webber

Jean is the author of OpenOffice.org Writer, The Free Alternative to Microsoft Word, which was used to guide me through the process of writing this book in OpenOffice Writer. She personally answered some questions regarding some formatting how-tos.

Robert D. Peavy

As well as being my father, Bob was the first to begin proof-reading the text. Many grammatical and spelling mistakes were first eradicated by Bob. He has also given me valuable advice on writing style and striving for conciseness.

Contributors 2

Contributors 3

Acronyms

In order to help you better navigate the alphabet soup of TLA's, I have provided a list of acronyms used in this book.

API - Application Programming Interface

ABI - Application Binary Interface

BSD - Berkeley Software Distribution

CGI - Common Gateway Interface

CPU – Central Processing Unit

FOSS – Free and Open Source Software

GNU - GNU's Not Unix

GPL – General Public License

GUI - Graphical User Interface

IDE - Integrated Development Environment

KDE - K Desktop Environment

LGPL – Lesser General Public License

MFC – Microsoft Foundation Classes

OS - Operating System

POSIX - Portable Operating System Interface

TLA – Three letter acronym

UI - User Interface

Glossary

Aqua

The windowing theme used in the OS X operating system.

API

Application Programming Interface – the interface that a software library or service provides to an external user for interaction.

Berkeley Software Design / Distribution (BSD)

The

BSD License

The software license that governs the BSD project. The license is extremely permissive in how the code may be used, mandating only that copyright notices be maintained within the code.

Cocoa

The OS X's native object-oriented UI toolkit.

Common Gateway Interface (CGI)

A

Cross-Platform

Programmed for more than one platform; Multi-platform.

Curses

A text based UI based on ...

FreeBSD

A flavor of BSD

Gnome

One of the two most popular windowing interfaces for the GNU/Linux operating system, the other being KDE.

GNU

GNU is a self-referencing acronym which stands for “GNU's Not Unix”.

Graphical User Interface (GUI)

A graphical means for interacting with a user.

Integrated Development Environment (IDE)

A software package used for software development. IDEs incorporate many discrete programming tools such as an editor, a compiler, a linker, and a debugger. Other tools such as a versioning tool, a graphical dialog editor, and a file differencing program may also be included. Some of the more common IDEs include Microsoft Visual Studio, Eclipse, and DevC++.

KDE

One of the two most popular windowing interfaces for the GNU/Linux operating system, the other being Gnome.

Linux

An

Longhorn

The codename for Microsoft's Vista operating system.

POSIX

A US government backed initiative to standardize the Unix operating system.

Recursive

See Recursive.

Unix

A good operating system.

User Interface (UI)

A means of interacting with the user. A UI may be graphical in nature (GUI) or text based such as the Windows console or curses.

Vista

The name for Microsoft's latest operating system.

Win16

Windows

Win32
Windows

X Windows
The X Windows windowing management system.

Index