Cross-Platform Approaches from a Macintosh Perspective

Jonathan W. Hoyle
jonhoyle@mac.com

Abstract

One of the many difficulties that Macintosh software developers must face is how to justify to their management the expense of development on the Mac when less than 1/10 of users are on it. Cross-platform development seems to be the only economically feasible way, yet so many of these approaches give you "lowest common denominator" solutions. This paper will detail some of the many approaches to cross-platform development from a Macintosh perspective, looking at the pros and cons of various frameworks and modern RAD applications. Highlights and pitfalls will be addressed for environments such as REALbasic and Java, but most emphasis will be placed on a C++ development perspective. Frameworks such as CPLAT, wxWidgets (formerly wxWindows) and Qt will be looked at, as well some legacy frameworks. Design principles which expedite a cross-platform strategy (such as MVC) will also be discussed. In the end, some recommendations can be made for best practices and viable approaches, plus downloadable sample projects will be made available.

Outline

1. Motivation
2. A Word About Java
3. Development Considerations
    a. C/C++ Compilers
    b. Mac OS X on Intel
    c. Architecting Using Model-View-Controller
4. Legacy Cross-Platform Frameworks
    a. Visual C++ Macintosh Cross-Compiler
    b. Yellow Box for Windows
    c. Mac2Win
    d. PowerPlant for Windows
5. Modern Cross-Platform Frameworks
    a. CPLAT
    b. wxWidgets (formerly wxWindows)
    c. Qt
    d. Other Cross-Platform Frameworks
6. REALbasic with a C/C++ Dynamic Library
    a. Creating the REALbasic GUI
    b. Creating the C++ Library
    c. An Example: The C++ Code
    d. An Example: The REALbasic Code
7. Five Rules for a Successful Cross-Platform Project
8. Summary

1. Motivation

The Macintosh is an excellent computer to use, primarily due to its ease of use and the power it gives to the user. Unfortunately, due to the Macintosh's smaller user base, it becomes difficult for software developers to justify to management why a given application should be ported to the Macintosh. Even with the more powerful tools available on the Mac, the ratio of expected Macintosh sales to development dollars is usually much lower than on Windows. One practical solution to this is the use of a cross-platform framework or development environment. Unfortunately, not all have strong support for the Macintosh. We will examine some of the most popular cross-platform approaches, with a strong eye on how well it works on Mac OS X. We will focus primarily on Mac OS X and Windows as our required platforms, and not consider any strategy that does not at least include these; Mac OS Classic and Linux will be mentioned in passing when a particular solution supports them. Furthermore, we will consider only those approaches which will be compatible with Apple's transition to Intel-based Macintoshes.

2. A Word About Java

Nowhere else is cross-platform development as important a priority than in the Java programming environment. This is arguably its most important contribution to the development community. In the past 10 years, Java has gained more market share and has had more books published about it than any other programming language. Furthermore, Apple itself has stated that Java is their official cross-platform strategy, leaving C++ positioned for platform specific development of Mac OS X. One might therefore ask: Why would anyone consider anything but Java for cross-platform development?

Although Java is a powerful approach for cross-platform development, the following are some important points to consider before excluding C++ from your set of options:

- There are two very different technologies both named Java: the front-end programming language and the back-end bytecode compiler. What makes Java cross-platform is the latter of these two. Although Java the language and Java the bytecode are usually intertwined, this is merely a matter of convention. Other languages can produce Java bytecode, and Java may be compiled into platform-specific assembly language[a].
- Java programs often suffer in performance due to the intermediary bytecode layer that requires translation by the Java Runtime Environment.
- The future of Java remains uncertain, as Microsoft's continued attempts to compete with it may eventually overwhelm the technology, particularly with the advent of Microsoft's competing C# language and .NET framework[1].
- Java programs often have a "lowest common denominator" look and feel. For example, some Java programs attach a menu bar to the application window instead of at the top of the desktop.
- Java's promise of being cross-platform has not always been fulfilled, as seen by the many Java applications which run only on Windows. Applications generated by Microsoft Visual J++ in particular are known for this, with incompatibilities on non-Windows targets being so extreme that it prompted Sun Microsystems to launch a lawsuit[2].
- Ultimately, Java and C++ are not mutually exclusive choices, as JNI (Java Native Interface) allows a Java application to link with C/C++ code.

Recommendation: If Java does become all or part of your cross-platform solution, it is important to choose a development environment which properly supports the Macintosh. Although Metrowerks has

[a] For example, an early version of Metrowerks CodeWarrior shipped with a Java language compiler which could compile directly to PowerPC assembly without Java bytecode. For the reverse, AMPC (Axiomatic Multi-Platform C) is an ANSI C compiler which generates Java bytecode [see http://www.axiomsol.com], thus bypassing the need for the Java programming language.

dropped support for Java since CodeWarrior 8.3[b], there are still a number of Java IDE's which are very powerful and Macintosh friendly. Xcode is an obvious first choice as it comes free with Mac OS X; its main negative is of course that it is Mac OS X-hosted only, making Windows debugging more difficult. Of other free Java compilers, Eclipse[c] stands alone as the best. For commercial products, Borland's JBuilder[d] is adequate for Mac OS X, but the strongest recommendation goes to IDEA[e] from Intellij, as it is clearly the best and most powerful modern Java environment today, on any platform for any price.

3. Development Considerations

Many of the C++ frameworks examined here are distributed as source files which need compiling on each platform, so we will consider the various compiler options. We will also consider how a cross-platform strategy my be impacted by Apple's move to Intel. Finally, we will discuss the Model-View-Controller architecture useful for successful cross-platform project.

3a. C/C++ Compilers

There are essentially two Macintosh C/C++ compilers of note today: Metrowerks CodeWarrior and Apple's Xcode. The former has held the dominant market share for the past 10 years; as of March 2004, 90% of all shipping Macintosh applications were developed using CodeWarrior[3]. However, use of Xcode has grown dramatically since Apple began shipping Xcode free with Mac OS X 10.3 Panther in 2003, with 56% of the top Mac developers now using it[4]. Xcode is based on gcc, so it relies heavily on the experience and knowledge of the Open Source community. CodeWarrior and Xcode are both excellent compilers, each offering their own advantages and disadvantages. Both compilers integrate well with Apple's GUI design program, Interface Builder. Metrowerks CodeWarrior has the better user interface, a more optimized compiler, faster compile times and better conformance to ANSI standards. Xcode, on the other hand, offers such options as Distributed Builds, Fix & Continue, better CodeSense and others.

Unfortunately, CodeWarrior has a very uncertain future since Metrowerks was spun off from Motorola to the apathetic Freescale. Meanwhile, Xcode has closed the gap against many of CodeWarrior's advantages by updating to a more ANSI compliant gcc 4, as well as improving the user interface to allow a more CodeWarrior-like "condensed" workspace. For those needing to continue support on Classic, CodeWarrior is required, as Xcode builds Mac OS X-only applications. For those wishing to take advantage of newer technologies, such as G5 optimizations, 64-bit compilation or Universal Binaries, Xcode may be the only option, as CodeWarrior currently does not support any of these (and perhaps never will). In the past year, the migration away from CodeWarrior has been inspired as much from Metrowerks' lack of commitment as it has from Apple's advances with Xcode[f].

Although there are many more choices for C/C++ compilers on Windows, Mac developers tend to gravitate to one of two options: Microsoft Visual Studio and Metrowerks CodeWarrior for Windows. Visual Studio is obvious because it is the largest and most common Windows compiler, as well as being the most supported by cross-platform frameworks. CodeWarrior for Windows, although not very well known among Windows developers, is especially favored by Mac programmers. Using CodeWarrior for both platforms, a single project file can contain both Macintosh and Windows targets off a single source

---

[b] http://web.archive.org/web/20011204194719/www.metrowerks.com/desktop/java/
[c] http://www.eclipse.org
[d] http://www.borland.com/us/products/jbuilder/index.html
[e] http://www.jetbrains.com/idea/
[f] Since 2003 and the release of 9.0, Metrowerks has seemingly ended Mac development on CodeWarrior, aside from maintenance updates. In perhaps a larger statement, Metrowerks refused to even attend Apple's Worldwide Development Conference in 2004, nor has it yet announced any plans for a next version of CodeWarrior.

base. Up through version 9.4, there is Win32 cross-compiler available in the Mac-hosted tools, thus allowing the developer to use his Mac for Windows development. Unfortunately, Metrowerks very recently sold off their x86 compiler technology, so this product is no longer available.

Recommendation: CodeWarrior 9.4's ability to compile for both Macintosh and Windows targets on the same project file, along with its superior user experience, would make it the ideal choice for a cross-platform compiler. Unfortunately, Metrowerks' continued support for the Macintosh platform remains highly questionable at this time. With Metrowerks' selling of its x86 tools, their Windows compiler is no longer available, nor does it appear that Metrowerks will be able to offer an x86 Macintosh compiler. Those using CodeWarrior for a pre-existing project may continue to do so, but it is recommended that they also investigate Xcode. New development projects should use Xcode from the start.

3b. Mac OS X on Intel

The selection of a cross-platform strategy should take into account the recent announcement by Apple that future Macintoshes will contain Intel chips. Mac OS X on PowerPC and on Intel are not actually separate platforms from a software perspective, as both are programmed to the same API (Application Programming Interface). Despite byte-swapping issues, the hardware move to Intel may be a relatively small effort for the software developer. Xcode 2.1 is available now and can compile Universal Binaries. REALSoftware has announced support for Intel on future versions of REALbasic. Java applications should be able to run completely unmodified. It appears that the only development environment that may be negatively impacted by the move to Intel is Metrowerks CodeWarrior[g]. This is a particular problem for applications needing to support Classic Mac OS, as Xcode does not support Classic. Mac OS X for Intel will apparently not support the Classic Environment, as Apple documentation on Rosetta emulation specifically states it will not run Mac OS 8 or 9 applications.[5]

3c. Architecting Using Model-View-Controller

Model-View-Controller (MVC)[6] is a way to architect software which separates an application's core data (the *Model*) from its user interface (the *View*). Typically, the model is written in a platform-independent manner, so MVC allows the developer to create a separate View for each platform[h]. The code which interfaces between the Model and the View is called the *Controller*. Both Apple[7] and Microsoft[8] have documentation for best practices with regard to using MVC.

A well-architected MVC application does not even require a cross-platform framework for porting. By using available RAD (Rapid Application Development) tools, such Interface Builder for Mac OS X, and Visual C# for .NET on Windows, thin platform-specific GUI apps can be created very easily, which then interface with model code. Modern RAD tools are dynamic and easy to learn; such tools are often more efficient than learning a new, complicated framework API.

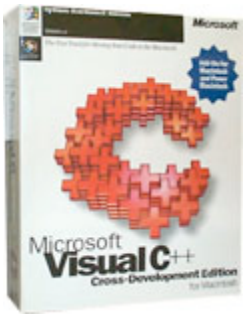
4. Legacy Cross-Platform Frameworks

Before describing modern cross-platform frameworks, it is useful to look at some legacy frameworks that were popular in the past. Note that these older frameworks are either discontinued or are no longer suitable for modern development and mentioned here for historical context.

---

[g] In an amazing combination of stupidity and bad timing, Metrowerks sold its 10-year held x86 compiler technology to Nokia, mere weeks before Steve Jobs announced the Macintosh's transition to Intel. As of this writing, Metrowerks has refused to state publicly what its commitment is for Mac on Intel.

[h] MVC is often used to make multiple views on the same platform; for example, it often useful to have both a GUI version to ship to the customer and an in-house command line version for unit testing.



4a. Visual C++ Macintosh Cross-Compiler by Microsoft (1995-1997)

http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/dnarvc/html/msdn_mfcmac.asp

For a brief time during the mid-1990's, Microsoft weighed in on Macintosh development by essentially porting MFC to the Mac. Within very specific limitations, a developer could create an MFC application for Windows and then cross-compile it for the Mac with this product. There were, however, several severe failings which eventually doomed this product from the start:

1. It was Windows NT-hosted. This is a major disadvantage to overcome, as most Mac developers prefer to work on the Mac. (Even Microsoft's defunct QuickBasic for the Mac was Mac-hosted.) Moreover, this required remote debugging for the Macintosh on Windows.
2. This was not a stand-alone compiler but an add-on to the Windows tools. Once principal development in Visual C++ began, the project was then cross-compiled with the add-on.
3. Not all of MFC functionality was ported, so `NOT_IMPLEMENTED` errors occurred frequently.
4. When it was first released, around the time the first PowerMac's were being introduced, it was 68K only. By the time Microsoft did include a PowerPC compiler with their tools with version 4, it was too late, as Metrowerks CodeWarrior had taken the Mac compiler market by storm.
5. It created notoriously slow and clunky apps. Microsoft Word 6 was the best example of that, with such poor performance that many Mac users stayed with version 5.1 and refused to upgrade.
6. It was obscenely expensive, costing $1999. And that was just for the add-on; you still had to plunk down the initial $495 for the standard Visual C++ compiler.

By 1996, Microsoft discontinued any further development on their Macintosh cross-compiler. They slashed the price to $199 to clear out remaining inventory. Microsoft has not produced Macintosh development tools since.

4b. Yellow Box for Windows by Apple Computer (1997-1998)

http://internetnews.com/dev-news/article.php/51871

At the 1997 Apple Worldwide Developer's Conference, Apple announced its plans for a new operating system: *Rhapsody* (a port of NeXTStep to the Macintosh). To take advantage of this new OS, it was required of developers to rewrite their Macintosh applications from scratch, using a new API called *Yellow Box* (essentially NeXT's old Objective C-based OpenStep API). Yellow Box applications would not only run natively on Rhapsody, but they could be easily recompiled to run on *Yellow Box for*

Windows, a set of libraries that ran the Yellow Box API on various flavors of Windows[i]. However, the requirement for developers to rewrite all their applications from scratch proved to be too much, and the Mac community rejected this new approach. The following year, Apple changed course once again by announcing Mac OS X (a merger of Rhapsody technology with the original Mac OS) and introduced another native (but more transitional) API called Carbon. Rhapsody's Yellow Box lives on today as the Cocoa API for Mac OS X.

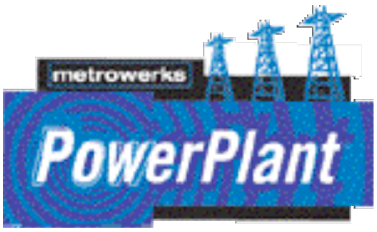
What was lost was Yellow Box for Windows. As flawed as the Rhapsody initiative was in 1997, it did provide an Apple-sponsored cross-platform framework. The promise of writing to a single API to create native apps on both the Mac and Windows was one of the few bright spots that year. Armed with a prerelease CD of Rhapsody and ProjectBuilder for Intel, many developers were excited by the prospect of creating these new cross-platform applications. Unfortunately it was never to be, as Yellow Box for Windows was cancelled by Apple.

4c. Mac2Win by Altura Software (1990-present)

http://www.altura.ro/Mac2Win/mac2win_frame.htm

Essentially, Mac2Win is a port of the original Mac OS API to Windows. By adding the Mac2Win libraries with your project, your application can be compiled and run on Windows. The appeal of this for the Macintosh developer was that he could develop his project in the standard Mac Toolbox and compile for Windows without a large rewrite. Unfortunately, only about 80% of the System 7 Toolbox was ported. And of the functions that were supported, many behaved differently on Windows than on the Mac, requiring extensive debugging.

More recently, a minimal set Carbon calls have been ported to allow Mac2Win based applications to be Carbonized. Unfortunately, these do not include many modern API's like Carbon Events. Although technically still alive as a product, it is not adequate for modern development. Mac2Win remains an expensive, royalty-based product, well outside the price range of the small developer. However, many well-known Macintosh applications got their initial Windows ports via Mac2Win[j], including Claris Works, Macromedia Director,[9] 4th Dimension[10] and Metrowerks CodeWarrior[11].



4d. PowerPlant for Windows by Metrowerks (2002-2004)

http://web.archive.org/web/20040207055356/www.metrowerks.com/MW/Develop/Desktop/PowerPlant.htm

Not long after CodeWarrior took the Macintosh development world by storm, its PowerPlant framework was quickly embraced due to its smooth design and usability. As CodeWarrior had been including a Win32 compiler, it did not take long before developers started requesting a Windows version of

[i] Yellow Box for Windows was initially supposed to be freely deliverable but was later priced at $249 per end user install.

[j] It is rather easy to determine where it is used: pressing `Alt-U-R-A` in a Mac2Win-ported application will bring up the Altura Secret About Box, giving the copyright date and Mac2Win version.

PowerPlant, thereby facilitating cross-platform developing. For years however, Metrowerks insisted that PowerPlant was a Mac-only framework, and there would never be a PowerPlant for Windows[11]. That was until Metrowerks purchased the Latitude Porting technology; by 2001, Metrowerks was finally ready to introduce PowerPlant for Windows.

PowerPlant for Windows seemed to get underway with a strong start when it was publicly embraced by Adobe[12]. The appeal was quite obvious to the Mac developer: use the PowerPlant framework (already the dominant one on the Mac), and with this library you get a Windows port as well. It was exactly what Macintosh developers had been begging for since CodeWarrior was first introduced. However, greed killed the initiative. At first launch, PowerPlant for Windows cost anywhere from $20,000 to $50,000. By early 2003, Metrowerks dropped the price to $15,000 plus 1% of the total product sales above $1.5 million, capping at a maximum of $150,000. Clearly not for the faint of heart. This price structure strangled any hope of survival for this product. Unlike Visual C++ and Mac2Win, Metrowerks never bothered to drop the price to a reasonable range before killing it, which they finally did in early 2004.

5. Modern Cross-Platform Frameworks

As a simple Google search will show, there exist a large number of cross-platform frameworks available for the C++ programmer. However, these frameworks will vary in degrees of quality, price and Macintosh usability. Due to the limitations of a single paper, it is not possible to look at every cross-platform framework that exists. Therefore, this list should be considered as only a starting point for investigating possible solutions.

CPLAT II

5a. CPLAT II by ksoft

URL: http://www.ksoft.net/cp_home.htm
Cost: $50
Platforms: Mac OS X, Mac OS Classic, Windows, (Linux under development)
Supported Compilers: CodeWarrior (Mac & Win), Xcode, Visual C++
Mac OS X on Intel Support: Upcoming release, expected August 2005

CPLAT, which compares very favorably to the others, is an amazing achievement by Ken Stahlman, the creator this framework. CPLAT is a low-cost cross-platform framework supporting Mac OS 9, Mac OS X and Windows, with Linux still under development. For a mere $50, this C++ application framework is powerful and very reminiscent of PowerPlant, giving a good Macintosh user experience. One of the problems with some other cross-platform frameworks is that Macintosh support tends to be secondary. This is not the case with CPLAT, as it clearly has the Macintosh as the primary platform in mind.

The current version as of this writing is CPLAT II (a complete rewrite of CPLAT I) and will compile on CodeWarrior 8.x and 9.x for both Macintosh and Windows; Xcode for the Mac and Visual C++ for Windows are also supported. One nice aspect of CPLAT is its ability to work with .nib files created from Interface Builder. Once the GUI has been generated with Interface Builder, the resulting .nib files can be converted into XML via the CP_NibToXML tool provided with CPLAT. The resulting XML can then be used to generate CPLAT GUI for Mac, Windows and even Linux. Macintosh applications created by CPLAT can be compiled for either CFM or Mach-O, and will run on Mac OS X 10.2 or higher (Classic is supported on OS 9 via CarbonLib). Windows applications will run on 98, 2000 or XP.

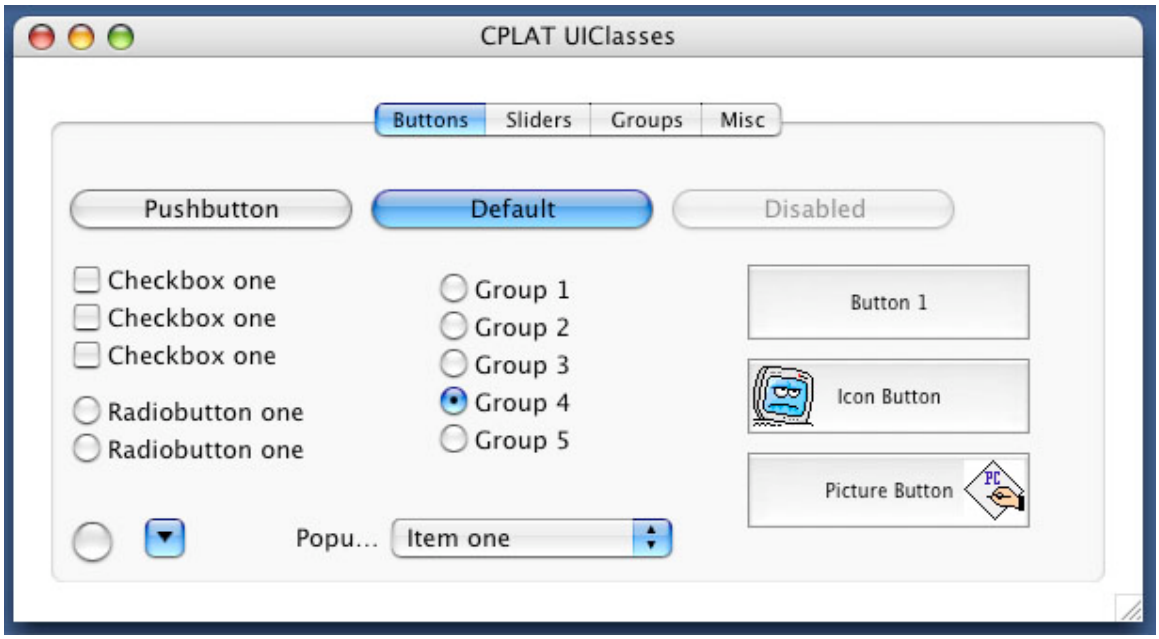


Figure 1a: A CPLAT dialog on Mac OS X 10.3 Panther

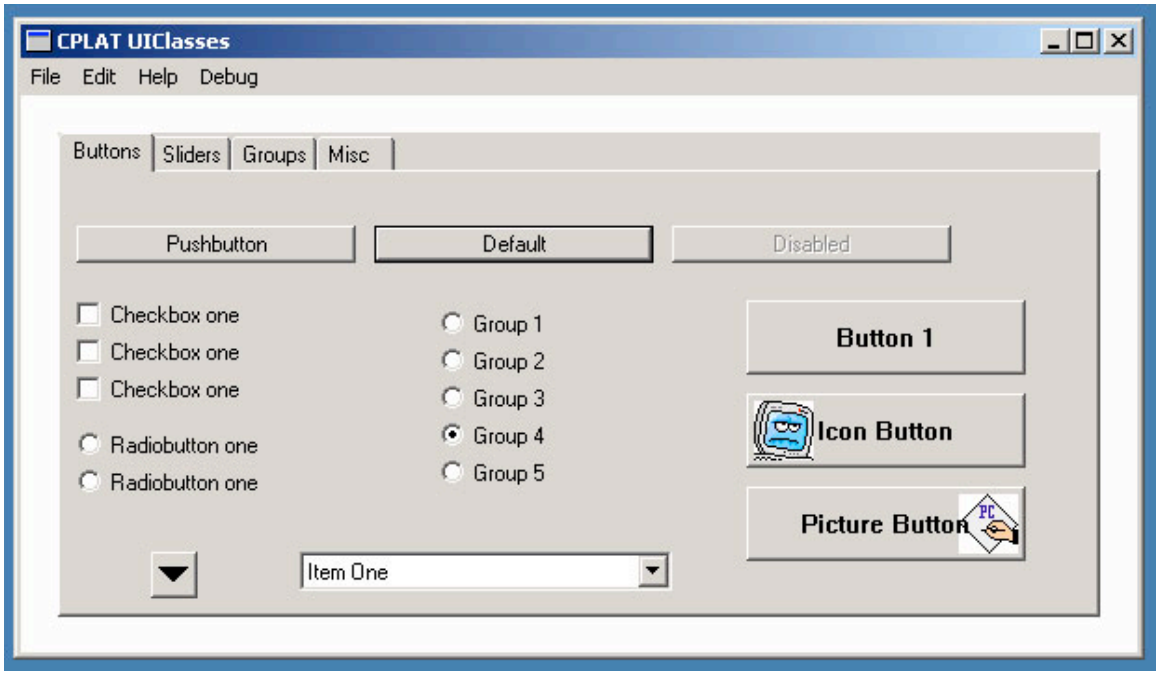


Figure 1b: The same CPLAT dialog in Windows 2000

Recommendation: The main negative to CPLAT is that kSoft is a one-man operation, and thus CPLAT support is subject to Ken Stahlman's efforts to keep it current. Support, however, has always been one of CPLAT's strong suits, and it isn't likely to end soon, as Ken has promised to update CPLAT for Intel-based Macs. CPLAT may not be as sophisticated (or as expensive) as Qt, but it is a top contender to consider for a cross-platform project. Although it's not free like wxWidgets, its relatively low price of $50 makes it very reasonable. If your background is PowerPlant development with CodeWarrior and you would like to begin cross-platform development, CPLAT is likely to be your most comfortable fit.

5b. wxWidgets (formerly wxWindows)

URL: http://www.wxwidgets.org
Cost: Free (Open Source)
Platforms: Mac OS X, Mac OS Classic, Windows, Linux
Supported Compilers: CodeWarrior (Mac only), Xcode, Visual C++, Borland C++, others
Mac OS X on Intel Support: Upcoming release, version 2.6.2

Due to a name change motivated by a frivolous lawsuit from Microsoft[13], wxWidgets remains a popular cross-platform framework. wxWidgets is Open Source and has a very large community of supporters. It also has one of the largest number of platforms supported: Mac OS 9/X, Win16, Win32, Win64, Linux x86, Solaris, AIX, HP-UX, Irix, SCO Unix, FreeBSD, OpenVMS, and even OS/2[k]. Additionally, there is an initiative to port to to embedded systems, including Windows CE and Palm OS. There is already a large number of products written with wxWidgets, including AOL Communicator[l]. With it being Open Source, the community will post "bounties", fees for specific bugs they'd like to see fixed.

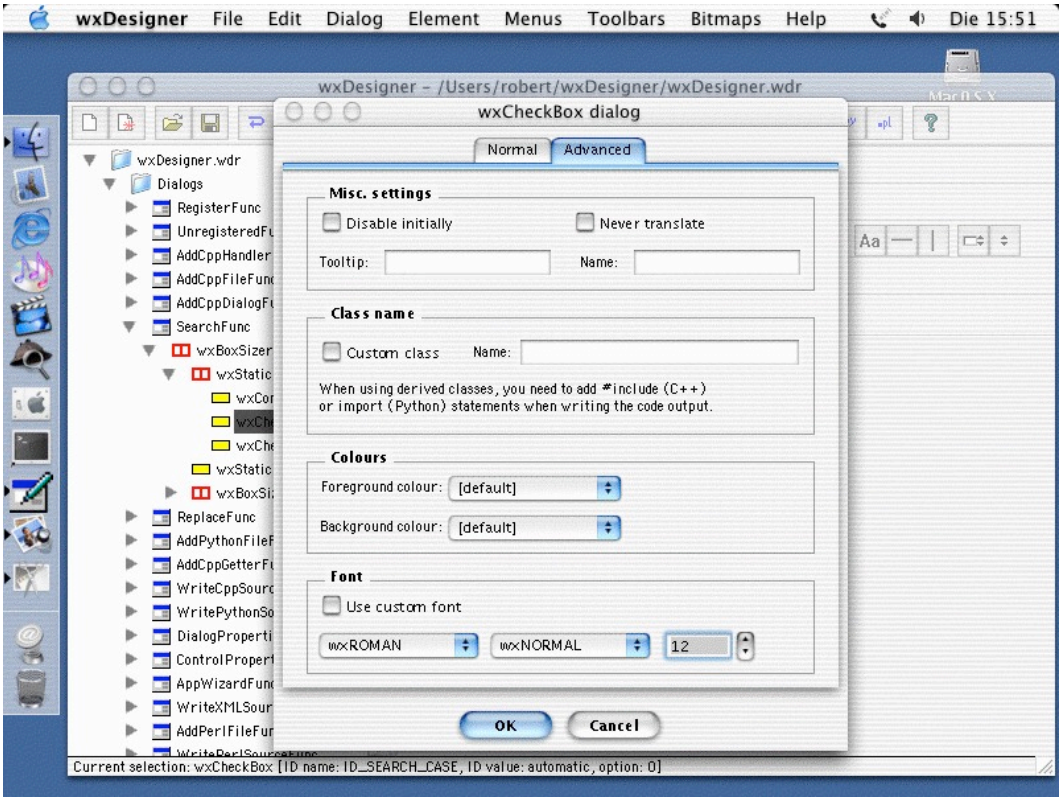


Figure 2: wxDesigner GUI program for wxWidgets

wxWidgets' Macintosh support has historically been poor but has been improving recently. The Macintosh version (called wxMac) supports both Mac OS 9 and X, although the wxMac FAQ[m] states that the Mac OS X port is still "in progress". Another annoying factor is that the Classic and OS X versions

[k] For a complete list of wxWidget's supported platforms, go to: http://www.wxwidgets.org/platform.htm
[l] For a list of other applications created by wxWidgets, go to: http://www.wxwidgets.org/users.htm
[m] http://www.wxwidgets.org/faqmac.htm

are separate codebases. The wxMac FAQ appears to be out of sync with other parts of the website, as it states that the compilers it supports for Classic are CodeWarrior Pro 5.3 and 6, and for OS X CodeWarrior Pro 6 and Xcode; yet elsewhere it says that the supported compilers are CodeWarrior 8.3 for Classic and Xcode for OS X. This kind of synchronicity error is not uncommon for Open Source projects, but it is particularly noticeable for wxWidget's Macintosh support. There are several commercial GUI generators compatible with wxWidgets, including wxDesigner[n] and DialogBlocks[o], but there is no ability to integrate with Interface Builder.

Recommendation: wxWidgets has an awkward feel from a Macintosh user's perspective, reminiscent of MFC, and not as smooth as the inexpensive CPLAT. It does however continue to improve with time, particularly due to growing support from the Open Source community. It is free and functional, with no limitations on commercial development, which is not true of either CPLAT or Qt. For developing freeware, the Open Source version of Qt edges it out with superior Mac OS X capabilities. But if it's a free cross-platform framework you desire for general purpose development, wxWidgets is not only the best game in town, it's essentially the only one.

5c. Qt by Trolltech

URL: http://www.trolltech.com
Cost: Tiered pricing[p]: $1790 Professional license, $2880 Enterprise license, free for Open Source development
Platforms: Mac OS X, Windows, Linux
Supported Compilers: Xcode, Visual C++, gcc
Mac OS X on Intel Support: Unknown as of this writing

Qt is arguably the most powerful cross-platform framework available today. Although Trolltech arrived a bit late to the Macintosh party (primarily deploying Windows and Unix), the introduction of Mac OS X inspired Trolltech to introduce an impressive Macintosh porting product. Without the burden of supporting Classic, Qt is a very modern and up-to-date framework. It weighs in at about 400 C++ classes, covering simple dialogs, controls, sophisticated OpenGL classes, database connectivity and more. Unfortunately, some of these powerful features are available only in the more expensive Enterprise edition. Included with the cheaper Professional license are RAD-like tools, such as QtDesigner for creating GUI's and QtLinguist for localization. The documentation is also very impressive, with a separate tool called QtAssistant which manages it nicely. The KDE environment for Linux was written with Qt[14] as were the Mac OS X versions of KOffice[15] and PostgreSQL GUI Client[16].

[n] http://www.roebling.de/
[o] http://www.anthemion.co.uk/dialogblocks/
[p] Qt pricing shown above is for a single developer; Trolltech's pricing scheme discounts in groups of two or three developers. For detailed pricing information, go to: http://www.trolltech.com/products/qt/pricing.html .

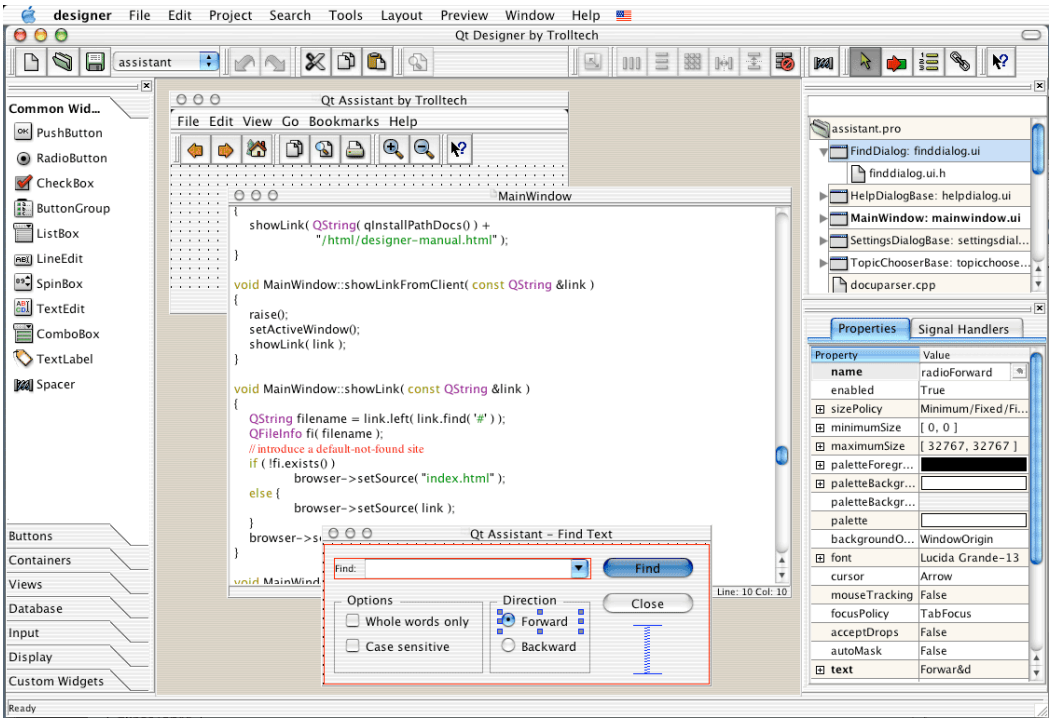


Figure 3: QtDesigner for building GUI's

In addition to the Professional and Enterprise editions, there is also an Open Source version of Qt free to download, and it is provided under the GNU General Public License. This means that you cannot commercially sell any application built with Qt/Open Source; furthermore, the license requires you to provide source code with every application distributed. Trolltech was wise in creating an Open Source edition as it becomes a vehicle to entice developers into using Qt: once a developer learns the framework for his own freeware project, he can then recommend it to his employer for commercial development.

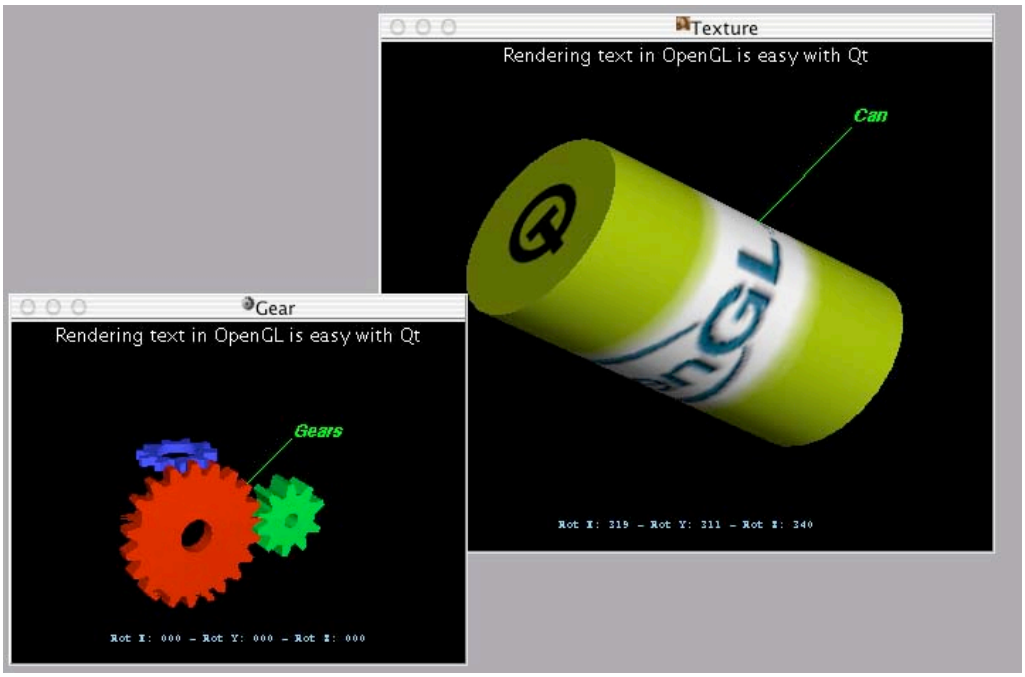


Figure 4: Graphical examples created by Qt

For more detailed information on using Qt, read Scott Collins' *Cross-Platform Development with Qt* also published at AdHoc/MacHack 2005[17].

Recommendation: As impressive and powerful as Qt is, its pricing is daunting and would place it outside the reach of the private developer. Although it is an excellent solution for Open Source development, it is an expensive route for commercial development. It also shows its Windows and Linux heritage, as the Mac OS X port still feels more like an afterthought. For example, there is no facility to work with .nib files created by Interface Builder, to which a Macintosh developer would be accustomed.

5d. Other Cross-Platform Frameworks

There are a number of cross-platform frameworks (many of which are free), and they are of varying levels of Macintosh support. As it is beyond the scope of this paper to evaluate them, they are listed below so that others may continue the investigation:

1. CroPL II *(Cross-Platform Library)*: http://www.crystalfiresw.com/products/cropl.html
2. YAAF *(Yet Another Application Framework)*: http://www.yaaf.org
3. FLTK *(Fast Light Toolkit)*: http://www.fltk.org
4. Whisper: http://www.sourceforge.net/projects/whisper2
5. ZooLib: http://zoolib.sourceforge.net

6. REALbasic with a C/C++ Dynamic Library

URL: http://www.realbasic.com
Cost: $99 Standard Edition per platform, $399 Professional Edition
Platforms: Mac OS X, Windows, Linux
Mac OS X on Intel Support: Upcoming release

One of the deficiencies of many of the aforementioned cross-platform frameworks is that they lack the RAD capabilities available to those using Interface Builder, Apple's design tool for GUI's. .nib files created from Interface Builder have a dynamic nature to them, allowing you to make changes without requiring a complete regeneration of the associated code. What would be desirable is a cross-platform equivalent to Interface Builder for rapid application GUI development.

This is precisely what REALbasic offers. REALbasic is an application development environment very similar to Visual Basic, except that it can compile for the Macintosh and Linux as well as for Windows. The GUI design portion of REALbasic is one of the most intuitive and powerful RAD tools available. Those familiar with VB would feel right at home with it; those familiar with Interface Builder will find it straight forward and easy to work with. And because it's the Basic programming language, it is not very difficult to learn.

How is this useful to the C/C++ programmer? Since REALbasic has the ability to call functions written in C and C++, the GUI can be easily generated, while the remainder of core functionality can still be written in C. REALbasic can essentially be viewed as a cross-platform version of Interface Builder. Architecting with MVC, REALbasic can easily create views need for your C++ model.

6a. Creating the REALbasic GUI

Those familiar with Microsoft Visual Basic for Windows will find REALbasic trivial to learn. Dragging and dropping GUI elements is straightforward, similar to Interface Builder. Double-clicking on any such element will take you to its instance methods where you can override or replace default functionality. Support ranges from standard controls to QuickTime movies and OpenGL. For additional functionality, there are a large number of REALbasic plugins, adding to the already impressive list of objects available. REALbasic documentation is fairly good, but there are still a number of behavioral holes which require a trip to the news:comp.lang.basic.realbasic newsgroup.

For more information on REALbasic, I recommend trying the demo version of the product by downloading it from their web site at http://www.realbasic.com .

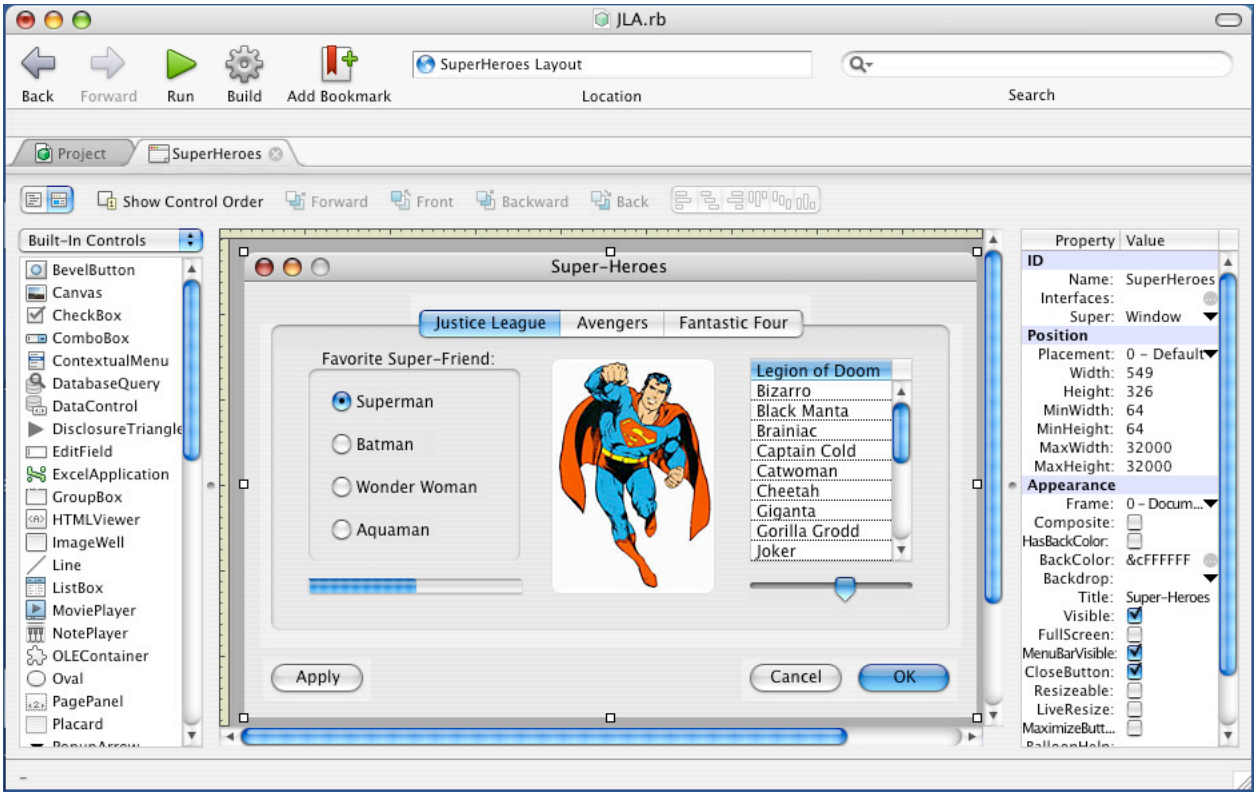


Figure 5: Designing GUI with REALbasic 2005

Note that beginning with REALbasic 2005, the designing GUI tool has changed to an all-in-one window format, similar to that found in Visual C++ and other Windows IDE's. Versions 5.5 and earlier used a more Mac-like multiple window paradigm, similar to Metrowerks CodeWarrior.

6b. Creating the C++ Library

Once you have created the Model classes for your application, they need to be bundled into a dynamic library which will export your Controller functions. Note that only those pieces of your C++ model code that interface with the GUI will need to be exported; once control has been passed in from the REALbasic application into the C++ library, the remaining portions of your model are available to you.

On Windows, the dynamic library type is called a DLL (Dynamic Linked Library and can be made with any number of compilers, such as Visual C++ and CodeWarrior for Windows. On the Macintosh, there

are two types of libraries which can be used: CFM/PEF-based (called a Shared Library) and Mach-O based (called a dylib). Shared Libraries can be used with either Classic or Carbon/CFM-based applications. If you wish to have your REALbasic application run in both Mac OS 9 and Mac OS X, encase your model inside a Carbon Shared Library built with Metrowerks CodeWarrior. If your REALbasic application is built as Mach-O, you must instead house your functions inside a Mach-O dylib, which you can build with either CodeWarrior or Xcode[q]. If you wish to have your C++ implementation to be 64-bit compatible, you will need to use Xcode. In general, Carbon/PEF libraries are far easier to work with and are much less fragile to work with. Jonathan Johnson has written an excellent article on creating Mach-O dylibs with Xcode and using it in a REALbasic application[18].

To avoid C++ name-mangling issues, we will export our model code as a library of C functions, using the `extern "C"` declaration[r]. Your standalone utility functions will just need a simple wrapping and you are done. For C++ classes and methods, mapping them to C equivalents can be done in a straight forward manner, such as using the `ClassName_MethodName` form. The implicit `this` pointer must also be passed in, which for us will be treated as an opaque handle.

6c. An Example: The C++ Code

Below is an example of sample code in the C++ library. The class name is `MyModel` containing a constructor, destructor and two methods `foo` and `bar`. We then create C-wrappers for them:

```
// A C++ model class used in the library's implementation
class MyModel
{
  public:
    MyModel();
    virtual ~MyModel();
    void foo(int parm1, double parm2);
    int bar(const char *parm);
};

// Exported functions associated with the MyModel class
#ifdef __MACH__
  #define export
#else
  #define export __declspec(dllexport)
#endif

extern "C"
{
  export int MyModel_Create();
  export void MyModel_Destroy(int modelHdl);
  export void MyModel_Foo(int modelHdl, int parm1, double parm2);
  export int MyModel_Bar(int modelHdl, const char *parm);
}
```

[q] Mach-O dylibs can also be accessed by CFM applications built with REALbasic 2005 using *Soft Declares*.

[r] If you know your compiler's decoration mechanism for C++ name mangling, you can bypass the extern C-wrapping and call C++ methods directly. However, the decorations can make code look really ugly: instead of calling `MyModel_foo`, you would be calling something looking like `?foo@MyModel@@QAEHHH@Z:NEAR`. Worse still, name mangling differs from compiler to compiler (and from version to version within a compiler), so you will not be able to call these mangled function names in a platform-independent way. For these reasons, using extern C-wrappers makes life much easier.

```
// Exported function implementations
int MyModel_Create()
{ return (int) new MyModel; }

void MyModel_Destroy(int modelHdl)
{ delete ((MyModel *) modelHdl); }

void MyModel_Foo(int modelHdl, int parm1, double parm2)
{ ((MyModel *) modelHdl)->foo(parm1, parm2); }

int MyModel_Bar(int modelHdl, const char *parm)
{ return ((MyModel *) modelHdl)->bar(parm); }
```

Most C++ methods will be able to be thunked down to C in this fashion. You can see from the example above that we used the `int` type to hold our class pointer, so we are assuming a 32-bit library for this particular example[s]. Our REALbasic application will treat `modelHdl` as an opaque reference and not be concerned with the fact that it is a pointer to memory.

The `__declspec(dllexport)` directive before the function declarations is required for PEF and Win32 libraries. It tells the compiler that these method names are available to callers of the library. Functions without this declaration are not exported. Mach-O libraries, on the other hand, export every non-static function by default, so no `__declspec(dllexport)` directive is needed.[t]

6d. An Example: The REALbasic Code

Once you have your functions exported from the dynamic library, they are now available to be called by your REALbasic application. At the top of a REALbasic method in which you call a library routine, you must `Declare` the exported function before using it. If our exported C function were of the form:

```
extern "C" ReturnType FcnName(pType parm, ...);
```

then the REALbasic declaration must be of the form:

```
Declare Function FcnName lib LibName(parm as pType, ...) as ReturnType
```

If `ReturnType` is `void` , then the word `Function` is replaced with `Sub` and `as ReturnType` is dropped. For Mach-O, `LibName` is a complete pathname to the dylib, which can be relative to the Unix executable (note that the Unix executable live two levels beneath the app's bundle). For non-Mach-O targets, just the library name will do if the library is in the same path as the application. REALbasic source code for a lib may look like this:

```
#if TargetCarbon
  const ModelLib = "MyModel Library"
#endif
#if TargetMachO
  const ModelLib = "@executable_path/../../../libMyModel.dylib"
#endif
```

[s] The simplicity of using a 32-bit `integer` to house a pointer as our opaque handle is just for ease of demonstration. If a 64-bit compatibility is desired, a better mechanism would need to be implemented.

[t] This sample code assumes that you are interfacing with REALbasic 5.5 or higher. REALbasic for Windows version 5.2 and earlier could not import `declspec` functions from DLL's. For these earlier versions of REALbasic, exported library functions had to be declared as `__stdcall` and their names be listed in a separate .def file.

```
#if TargetWin32
  const ModelLib = "MyModel.dll"
#endif
#if TargetLinux
  const ModelLib = "libMyModel.so"
#endif

Declare Function MyModel_Create lib ModelLib() as integer
Declare Sub MyModel_Destroy lib ModelLib(modelHdl as integer)
Declare Sub MyModel_Foo lib ModelLib(modelHdl as integer,
                                     parm1 as integer, parm2 as double)
Declare Function MyModel_Bar lib ModelLib(modelHdl as integer,
                                          parm as Cstring) as integer

Dim modelHandle as integer
Dim barValue as integer

modelHandle = MyModel_Create()
MyModel_Foo(modelHandle, 12, 3.0)
barValue = MyModel_Bar(modelHandle, "Hello, World")
MyModel_Destroy(modelHandle)

return barValue
```

For some downloadable sample projects, check out http://www.jonhoyle.com/MacHack/ .

Recommendation: REALbasic's claim that it is "cross-platform that really works" is only partially true: the $99 Standard Edition compiles for only a single platform, and you must purchase a separate copy for each platform you wish to support. The $399 Professional Edition does however allow you to cross-compile to each of the other platforms. Moreover, the Professional Edition offers extended databasing capabilities, the creation of console applications on most platforms, etc. For this reason, the Professional version of REALbasic is recommended for advanced development. Overall, REALbasic is the simplest, most powerful way of creating cross-platform applications.

7. Five Rules for a Successful Cross-Platform Project

Regardless of which cross-platform approach is taken, a project can be quickly undermined by a poor process. Through experience, I have learned a number of simple principles, which if followed, will help you avoid the pitfalls that many cross-platform projects fall into.

1. Design your application with a Model-View-Controller (MVC) architecture. This is perhaps the single best key to success in any cross-platform project.
2. Have both Mac & Windows developers working together from the start. If you are serious about being cross-platform, you need to have the expertise of both engineering groups early in the design and implementation[u].

[u] This important lesson was lost on a development team I had worked with in 1999. The project was a properly MVC layered application with the model written as a C++ DLL and the view being a Java Swing application via JNI, all written in CodeWarrior for both Mac OS & Windows. Sounds good so far, right? Well, the Windows team developed first and handed it off to the Macintosh team afterwards, which they found out only later that the application was written to the Java 2 specification (which was not supported on the Mac at that time). Worse still, the source filenames themselves exceeded the 31 character limitation of the Finder, so project files would not compile due to broken `#include`'s. Management decided to split the project into separate Windows and Macintosh branches, each of which remained laden with the weight of an unused cross-platform design. All this waste could have been avoided had there simply been Mac developers advising the team from the start.

3. Use a single code branch for source control. Separate code branches often have the undesirable effect of spinning out of sync very quickly[v]. Use `#ifdef`'s when you need to do something platform-specific within your codebase.
4. Write ANSI-compliant C and C++ code, using standardized ANSI and STL libraries. Since many cross-platform projects use multiple compilers, it is important to minimize the errors and warnings that may crop up on the other platform. To further this, make it a point to turn the warning level high and require all checked in code have no warnings.
5. Put both a Mac and a PC onto each of your developers' desks. There will be times when a developer on one platform will want to see how his code has affected the other platform. If the developer does not have easy access to the other machine, it makes it more likely that check-in's will break on the other platform.

8. Summary

There are essentially three modern cross-platform frameworks available to the C++ programmer today which support Mac OS X and Windows: CPLAT II, wxWidgets and Qt:

- CPLAT wins for being the most Macintosh-friendly. Those accustomed to developing with PowerPlant on the Macintosh will find it a familiar feel; for a mere $50, CPLAT is an excellent value and will likely serve most all of your needs for crossplatform development.
- With no cost or development restrictions at all, wxWidgets is a cross-platform framework which is Open Source. Although it acquits itself adequately for the Macintosh development, it has an MFC/Windows feel to it.
- At the high end, Qt is certainly the most powerful framework but it's also the most expensive. Lacking Classic support, Qt is able to embrace Mac OS X fully without the legacy of backward compatibility. Although producing superb results, it is priced outside the range of the individual developer; the GNU version however is free for developing Open Source products.

As an alternative to heavy frameworks with a new API, architecting software using MVC allows the developer the flexibility of using more modern RAD tools. By housing your data model inside of a C/C++ dynamic library, many GUI tools become available for the View. Arguably the easiest and most powerful of these is REALbasic, which can generate Macintosh, Windows and Linux GUIs all from the same project file, which in the end is this author's recommendation.

For more information, check for updates at http://www.jonhoyle.com/MacHack/ .

Further Reading

REALbasic:

1. *REALbasic, the Definitive Guide, 2nd Edition*, M. Neuburg, 2001, ISBN: 0596001770
2. *REALbasic for Macintosh*, M. Swaine, 2002, ISBN: 0201781220
3. *REALbasic for Dummies*, E. Tejkowski, 2001, ISBN: 0764507931
4. *Learning REALbasic through Applications*, C. Clayton, 2002, ISBN: 1584502061

Qt:

1. *C++ GUI Programming with Qt 3*, J. Blanchette et al, 2004, ISBN: 0131240722
2. *Programming with Qt*, M. Dalheimer, 2002, ISBN: 0596000642
3. *Teach Yourself Qt Programming in 24 Hours*, D. Solin, 2000, ISBN: 0672318695

[v] On a project I once led, management made the decision to branch the code so that the Mac version could be released sooner. Although a few months were saved in doing so, the cost was a year and a half before for two branches to be remerged.

wxWidgets:

1. *Cross-Platform GUI Programming with wxWidgets*, J. Smart et al, 2005, ISBN: 0131473816

Metrowerks CodeWarrior:

1. *Metrowerks CodeWarrior Professional Book*, D. Sydow, 1997, ISBN: 1566047331
2. *CodeWarrior Software Development Using PowerPlant*, J. Harrington, 1999, ISBN: 0123264227
3. *Metrowerks CodeWarrior Programming, 2nd Edition*, D. Sydow, 1996, ISBN: 1558515054
4. *C++ Programming with CodeWarrior*, J. Harrington, 1995, ISBN: 0123264200
5. *Mastering CodeWarrior for Windows 95/NT*, J. Trudeau, 1997, ISBN: 0782120571

Xcode:

1. *The Mac Xcode 2 Book*, D. Cohen et al, 2005, ISBN: 0764584111
2. *Step into Xcode*, F. Anderson, 2005, ISBN: 0321334221

Bibliography

[1] DrPizza, *Microsoft .Net*. **Ars Technica**, 2/02 < http://arstechnica.com/paedia/n/net/net-4.html >
[2] Armstrong, E. and J. Niccolai. *Sun Granted Injunction Against Microsoft*. **JavaWorld** December 1998 < http://www.javaworld.com/javaworld/jw-12-1998/jw-12-injunction.html >
[3] Tyler, M. *CodeWarrior Development Studio 9*. **MacAddict** Mar 2004 < http://www.macaddict.com/issues/0403/rev.codewarrior.html >
[4] D. Sellers, *Be Careful of the Bridges You Burn*. **Macsimum New**, 6/17/05 < http://www.macsimumnews.com/index.php/archive/5416/ >
[5] Fried, I. *Intel Deal May Mean End to OS 9 Support*. **ZDNet News** 6/6/05 < http://news.zdnet.com/2100-1040_22-5734410.html >
[6] *Model-View-Controller*. **Wikipedia** 7/1/05 < http://en.wikipedia.org/wiki/MVC >
[7] Apple Computers, *The Model-View-Controller Design Pattern*. **Apple Developer Connection** 5/27/04 < http://developer.apple.com/documentation/Cocoa/Conceptual/AppArchitecture/Concepts/MVC.html >
[8] Microsoft Corporation. *Model-View-Controller*, **MSDN Library** < http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/DesMVC.asp >
[9] Winer, D. *Quorum and Altura*. 1/27/97 < http://davenet.scripting.com/discuss/msgReader$300 >
[10] 4th Dimension Consulting, *4th Dimension XML Keys*, < http://www.4d-consulting.com/Manuals/4D_DOC_HTML_2004/XML_Keys_US/000_Legal_Title.html >
[11] Mark, D. *From the Factory Floor (Interview with Greg Dow)*. **MacTech** Apr 1996 < http://www.mactech.com/articles/mactech/Vol.12/12.04/Apr96FactoryFloor/ >
[12] *Adobe Uses Metrowerks for Mac/Windows Transfer*. **Austin Business Journal**, 2/7/02 < http://www.bizjournals.com/austin/stories/2002/01/07/daily3.html >
[13] Loli-Queru, E. *Microsoft Intervention Pushes wxWindows to become wxWidgets*. **OS News** 2/20/04 < http://www.osnews.com/story.php?news_id=6100 >
[14] Dalheimer, M. *Programming with Qt, 2nd Edition*, **O'Reilly & Associates**, 2/02
[15] Turner, D. *'Grunt Work' Brings Early Version of KOffice to Mac OS X*, **eWeek**, 1/5/04 < http://www.eweek.com/article2/0,,1426578,00.asp >
[16] *OpenRPT – Open source GUI Report Writer and Renderer for PostgreSQL*, **PostgreSQL News**, 4/26/05 < http://www.postgresql.org/about/news.314 >
[17] Collins, S. *Cross-Platform Developent with Qt*. **AdHoc/MacHack** 7/05
[18] Johnson, J. *Creating a DyLib on Mac OS X and Declaring into it*. **NilObject** 2/27/05 < http://www.nilobject.com/?p=184 >