

According to Hoyle...

Cross-Platform Software Development from a Macintosh Perspective: Converting Legacy Frameworks

by Jonathan Hoyle

jhoyle at maccompanion.com

macCompanion

July 2006

For these past 10 months, we have spent this column examining a wide number of cross-platform frameworks and development environments available from a Macintosh perspective. For those just catching up with us now, here is what we have covered so far:

- **Intro:** http://www.maccompanion.com/archives/september2005/Columns/According_to_Hoyle_1.htm
- **Qt:** <http://www.maccompanion.com/archives/october2005/Columns/AccordingtoHoyle.htm>
- **wxWidgets:** <http://www.maccompanion.com/archives/november2005/Columns/AccordingtoHoyle.htm>
- **CPLAT:** <http://www.maccompanion.com/archives/december2005/Columns/AccordingtoHoyle.htm>
- **REALbasic:** <http://www.maccompanion.com/archives/january2006/Columns/AccordingtoHoyle.htm>
- **Runtime Revolution:**
<http://www.maccompanion.com/archives/february2006/Columns/AccordingtoHoyle.htm>
- **AMPC:** <http://www.maccompanion.com/archives/march2006/Columns/AccordingtoHoyle.htm>
- **Java compilers:** <http://www.maccompanion.com/archives/april2006/Columns/AccordingtoHoyle.htm>
- **Basic compilers (Part I):**
<http://www.maccompanion.com/archives/may2006/Columns/AccordingtoHoyle.htm>
- **Basic compilers (Part II):**
<http://www.maccompanion.com/archives/june2006/Columns/AccordingtoHoyle.htm>

All of these articles have been discussing the various options of starting a new project. However, a large number of you already have projects in place but need to update their codebases. This month, we shall tour some of the most widely used frameworks and development environments and look at cross-platform tool choices, which best fit them.

Metrowerks PowerPlant

Until relatively recently, the dominant C++ framework used by Macintosh developers was Metrowerks' *PowerPlant*, an object oriented framework which came free with their C++ compiler CodeWarrior. *PowerPlant* served the needs of both Classic and Carbon developers alike for nearly a decade. Unfortunately, its parent company, Motorola, spun Metrowerks off to Freescale, where it was mismanaged and starved for resources. In a short two-year span, Freescale turned a dominant marketshare into insolvency. Left high and dry are the thousand upon thousands of Macintosh software projects, which were heavily invested into the *PowerPlant* framework.

One solution, which has recently become more popular is the C++ framework *CPLAT* [http://www.ksoft.net/cp_home.htm]. *CPLAT* is an object-oriented framework with an API very similar to *PowerPlant's*. Although not as fully featured as other solutions (such as *Qt*), *CPLAT* does the majority of what most applications need. It is not free, but its \$50 price tag is very reasonable, especially when you consider that you will be able to compile for both Windows and Macintosh. For a full review of *CPLAT*, see this article:

<http://www.maccompanion.com/archives/december2005/Columns/AccordingtoHoyle.htm>

Microsoft Foundation Classes (MFC)

Just as PowerPlant was the dominant framework for Carbon C++ developers, so too is MFC for Windows C++ developers. There is a huge base of MFC applications, virtually all of which are Windows specific. Porting these projects to a cross-platform framework is best done with wxWidgets [<http://www.wxwidgets.org/>]. wxWidgets' API is modeled to be very similar to MFC. Best of all, wxWidgets is open source, so there is no charge to you the developer. Using wxWidgets, your project can be compiled for Mac OS X, Classic, Linux and many other operating systems. See this article for a full review of this framework:

<http://www.maccompanion.com/archives/november2005/Columns/AccordingtoHoyle.htm>

Cocoa and Interface Builder

For those using *Cocoa* to create Mac OS X applications, Interface Builder is an integral part of creating the GUI. Unfortunately, Interface Builder is Macintosh-specific, and is thus unsuitable for cross-platform development. However, Interface Builder need not be dropped if your project is architected properly. In particular, using a design pattern called *MVC (Model-View-Controller)*, you separate out your business logic (the *Model*) from Interface Builder .nibs (the *View*) and its associated interaction (the *Controller*). Following this pattern, you would keep all of your Objective C code inside the *Controller* files, as they would be Mac-specific. Your model files should remain in C or C++. At this point you can then use a different rapid application development package to create a GUI for another operating system, e.g.: *Visual C# for Windows*.

For more information on *MVC*, see: http://en.wikipedia.org/wiki/Model_view_controller .

Another solution that will become available is with *REALbasic*. REAL Software has announced that *Cocoa* bindings will become available in a future release. When this happens, you will be able to replace Interface Builder with *REALbasic's* cross-platform GUI generator. Although the *Cocoa* bindings themselves will remain Mac-specific, it will allow you an easy access to drop in Windows-specific code where needed.

Microsoft Visual Basic

Visual Basic is the dominant Basic compiler available today, regardless of platform. The power and elegance of its environment is very attractive, and so most *VB* users are not inclined to leave it just to include Macintosh and Linux customers. Fortunately, there is a cross-platform product, which is very much like it: *REALbasic* from REAL Software [<http://www.realbasic.com/>]. The look and feel of *RB 2006* is very much like *VB*, so users will not have much of a problem making the transition. *REALbasic's* programming API is similar to classic Visual Basic, so those transitioning from *VB 6* and earlier will find it fairly straightforward. Those using *VB.NET* will find that they will need to convert their .NET system calls. REAL Software has documentation on *Porting VB Applications to Linux and Mac OS X* [<http://www.realbasic.com/support/whitepapers/portingvisualbasic/>], and it is definitely worth beginning the process with this document. For the full review of *REALbasic*, go to:

<http://www.maccompanion.com/archives/january2006/Columns/AccordingtoHoyle.htm>

Another choice available to classic *VB* users is *KBasic* [<http://www.kbasic.com/>]. *KBasic* is a low cost downloaded Basic compiler designed to closely mirror the *VB 6* API. Although not as full featured as *REALbasic*, *KBasic* is still powerful, and less expensive. Note however that its Macintosh target is relatively new and available only in the Professional version of *KBasic*.

Visual C# and .NET

Visual C# was introduced by Microsoft when they delivered their first .NET programming environments. Despite the occasional rumors that Microsoft will port *C#* to the Macintosh, it is highly unlikely that this will ever transpire. So what would be the best path forward for *C#* projects? The solution involving the least amount of work would be to simply port the *C#* project to *Java*. *C#* is essentially *Java* with extras, and so the conversion process is a lot easier than to, say, *C++*. The biggest work will be in replacing .NET calls with native *Java* calls, such as the *Swing* API.

Another solution is to use one of the many *C#* to *VB* converters and then translate into *REALbasic*. This would be ideal for those who find *Basic* to be a more suitable language than *Java*. The *Java Swing* API is very different from .NET, so again most of the work will be in translating API calls.

MacApp and other Macintosh frameworks

Older Macintosh frameworks such as *MacApp* and *Think Class Libraries* do not have easy transition options. These frameworks are similar enough to *PowerPlant* that you may wish to follow the recommendations for it. However, serious consideration ought to be given for rewriting the app from scratch. If money is no object, the expensive *Qt* is probably the best framework choice from a feature perspective [see <http://www.trolltech.com/>]. For those on a budget, the fastest way to develop cross-platform software today is by using *REALbasic* as the GUI front end, while the back end can be kept in *C/C++* housed within a *DLL*. Another approach is to port the entire application to *Java*. Regardless of which direction you choose to go, rewriting the application completely is likely to be your penalty for having waited so long.

Next Month: *C++* Framework Update! See you in 30!