Making Peace With Windows

Allen Prescott <allen@clanprescott.com>
10665 Baxter Avenue Los Altos, CA 94024

I am still trying to decide which was more difficult for me. Leaving my first wife or deciding to start programming Windows. However, it is a good idea in today's market to develop as many marketable skills as possible so I finally decided to take the plunge. It was, and continues to be, a difficult thing. When you are ready to take the plunge you will have to confront some disturbing realities. The worst of which is that you are no longer master of the universe. I started programming the Mac when it came with 512K of RAM. Many years later I felt like there was no situation that I could not handle. I knew all the right tools for each job. I was fluent in 68K and PPC assembler. I knew what to expect from the system software. I knew all the little tricks that made my life as a programmer simple. It is not that all of those years of programming did not help me in Windows land. On the contrary they helped me a lot. However, I had a whole new world to conquer and at first it seemed very large and unfriendly. It did get better as I learned more. The unfortunate thing is that you cannot simply sit down for a month, read everything that there is to read, and come away knowing everything that you need to know. You've simply got to dive in and do it for awhile. Eventually it all starts to work for you.

Picking Hardware

First you have to equip yourself. Picking out hardware is pretty easy in Mac land. Just get the biggest, baddest machine available at the time and hope that it does not become obsolete before your first project is done. In Windows land there are many more issues. First you have to decide what OS you are planning to target. Windows NT is the future but Windows 95 is the present. If you are planning on writing consumer applications then Windows 95 will be really important to you. If you want to focus on server applications then you will want to skip Windows 95 and go straight to NT. Either way you will probably want to run both at one time or another. This means that you will need a dual boot setup. This can be a problem. None of the major vendors that I spoke with when I did my first system would certify a dual boot system (in fact some would not even certify an NT system). You should ask your vendor specifically about a dual boot configuration. I have found that the best thing to do is to get an NT system and set up the dual boot yourself. The thing to make sure of here is that there are both Windows NT and Windows 95 drivers available for your devices, particularly your CD-ROM drive. Drivers readily available on the web are a big plus. Ask the vendor and verify for yourself.

Once the new hardware is on your desk make a boot floppy that has CD drivers on it so that you can boot from this floppy and read your CD. This will be a lifesaver in the future. I found it easier to install Windows 95 then install Windows NT on top of it. NT installs nicely on top of 95 but installing 95 on top of NT is much more trouble. So I recommend that you buy a Windows NT setup, reformat your drive, install Windows 95, and then install Windows NT on top of it. You should find a contact that has the appropriate Windows hardware battle scars and pick a day when your contact is available. Then plan on spending the entire day wrestling with your configuration. There is an off chance that you will install everything and it will just work, but that has been very unusual in my experience.

Hardware requirements are another issue. Windows NT and a development environment will run marginally in 64MB of RAM but do yourself a favor and get 128MB up front. That will improve performance a lot. You should also get a minimum of 4GB of disk space, but I recommend at least 6GB. Dual boot systems and development tools will eat up a whole lot of disk space (not to mention if you decide to install a video game or two). In the matter of disk space and memory more is always better. Sound card and monitor choices will depend on your target application. For the sound card make sure that you get one that has readily available drivers for both NT and 95. If you have the money, an extra processor is a nice touch; a dual-processor box is generally much more responsive and performs better. I am not sure that it is really worth the extra money for a user box but it is nice. If you are planning to do multi-threaded programming, then a dual processor system is a higher priority.

Don't try to "save money" by buying cheap parts unless you are really familiar with hardware and drivers. The time that you lose wrestling with drivers and incompatibilities will far outweigh the few dollars that you save. Along with the lost time will come frustration and mistrust. A developer has enough worries without having to constantly wonder if a problem is in the code being developed or the configuration.

Picking Software

So now you have wrestled with vendors, anguished over hardware choices, and wasted a weekend getting your new box all set up. Assuming that you did not damage it permanently in a fit of rage over configuration issues, you will now need developer tools. No matter where you are going from here you will probably want to join the Microsoft Developer Network (MSDN). You should first visit Microsoft on the web at http://www.microsoft.com. You can then navigate your way to the MSDN site and see the available options. The Universal MSDN subscription comes with Visual C++, Visual Basic, Visual J++, Windows NT Server, the Office Suite, SQL Server, all of the SDKs and device driver kits as well as various other tools. See the MSDN site for a full listing. If you need all of this, then the Universal Subscription is a good deal. If you are planning to use a single compiler or already have the BackOffice products, then one of the lower-cost services will do just fine. Most of the stuff on the CDs is also available on Microsoft's web site somewhere, but it can be nice to have the CDs, particularly if you do not have a high speed Internet connection. If you go for the MSDN subscription, be prepared for a **lot** of CDs. You will get multiple boxes with piles and piles of CDs. Finding what you are after in this pile of CDs can be very challenging. They will come with a printed index of what products live on which CDs. Do yourself a big favor and keep this index handy. If you use the index and keep the CDs in the proper boxes and in the proper order, you will save yourself quite a lot of time and frustration digging information out of the pile.

You will have some choices in compilers. The MetroWerks offering is very nice and will be a friendly place for you if you were using that environment on the Macintosh. My experience with them is that they make very good tools. I have not used the Windows version extensively but a friend with some experience claims that the Windows version has a few slings and arrows.

The most prolific compilers are the Visual family from Microsoft. If you are looking for the latest and greatest of technology then you will probably want to go with Visual. Microsoft tends to deliver bleeding edge things integrated into this environment. I found it very full featured

and easy to use. The integrated debugger/editor/compiler is very nice. You can customize pretty much everything. Overall, it is a good environment, though they did seem to go with the 80/20 rule in quality control. You will find plenty of bugs. For example, when you try to compile the following code:

```
for (int inx = 0; inx < 3; inx++) {}
for (int inx = 0; inx < 3; inx++) {}
```

you will discover that it does not compile because inx is multiply defined. This will let you know that the compiler does not conform to the latest C++ specification. This is a polish thing that is easy enough to work around. I am just used to the folks at MetroWerks dotting those i's and crossing those t's. Overall you will find it more than adequate for most application-level development.

The Visual environment includes a resource editor and debugger that will do very nicely for all but the most demanding of situations. If you are planning on sophisticated icon design or other artistic work, you might want to consider a special-purpose program. I have no experience with any of those programs so I will leave you on your own there.

The object browser included is full-featured but buggy beyond use in my opinion. You might want a copy of a product such as Object Master from Altura Software (http://www.altura.com) to supplement the environment.

The editor is nice and full featured. I am perfectly happy with it, but a lot of people that I know prefer to use CodeWright from Premia (http://www.premia.com). This seems particularly true of old-time Windows folks.

The debugger is nice. You can do source or assembler debugging in your own code. You can set break points and conditional break points. There is an expression evaluator and a raw memory viewer. It is fairly weak in expression coercion but you can get by with the memory window. It is also pretty good about attaching to DLLs and other code resources to allow source level debugging. If you want to do lower-level debugging into other people's code, then you might want to get SoftIce from NuMega (http://www.numega.com). It is a little pricey, and I have never used it personally, but I have heard a lot about it and there have been several occasions where I really wished that I had a lower-level debugger.

These are by no means the only products available or even necessarily the best. These are just the ones that I have been exposed to and like. You should take a trip to your local software store and see for yourself.

Reference Materials

When I first conceived of this notion I thought that I would surely need to recommend books that you can read that will help get you started. The problem was that I never read any of the books myself. You can go down to your local bookstore and find hundreds of books on programming Windows that I am sure are fine volumes full of useful information. However, there is a lot of documentation and sample code that comes with the MSDN subscription and more on the Microsoft web site. So I just picked a starting project, surrounded myself with online references, and dove in. Thinking that I was surely alone in this tactic, I composed an email asking several professional programmers that I know to recommend books that they had read and found useful. To my surprise I did not get a single recommendation back. It seems that all of them got started pretty much the same way that I did. This at least proves that it can be done. It may ease your burden to spend some time reading books to get familiar before you get started, but it isn't essential.

The one book that a lot of people know, but no one seems to have read a lot of, is Charles Petzold's *Programming Windows 95* (Microsoft Press, ISBN: 1556156766). I have a copy of his Programming Windows 3.1 book but I have never read it. If you are going to be doing Win32 applications with lots of UI you may want to check this one out.

COM, OLE, and ActiveX

COM, OLE, and ActiveX are terms that are frequently used interchangeably. In reality COM is the underlying object technology layer and OLE/ActiveX is the higher level layer on top of that technology. It all started out as an OpenDoc-type document object system called OLE. The goal was to integrate applications more easily by providing black-box code sharing. As time went on, and the Internet became more popular, the goal evolved and the name was changed to ActiveX. It is now targeted more at distributed web objects.

If you need to imbed a component into someone else's application, such as Excel, then ActiveX is the technology for you. This technology is based upon COM (Component Object Model). COM is an object-broker type technology. It lacks some of the more sophisticated features of other object-broker technologies, but is adequate for most applications. It provides location transparency, named lookups, parameter marshaling, and primitive life-cycle management. If you want to expose features in your application to outside code, it is fairly easy to make functions available to the world via COM. This opens the door for scripting as well as other kinds of code sharing opportunities. COM also makes it easy to mix and match code from different languages such as C++ and Java. With DCOM (Distributed COM), it is also easy to mix and match components from different machines over a network.

COM life-cycle management is primitive. It is basically just a reference count variable. You are responsible for keeping the reference count correct and disposing of an object when the count reaches zero. Do not underestimate the importance of life-cycle management. It can be a real pain to keep those reference counts accurate, but the penalty for not doing so can be very severe. ATL (Active Template Library) is a set of foundation classes provided with Visual C++ that provides really nice wrappers around COM objects for C++ programmers. If you are planning to get serious about COM, I strongly recommend it. It will ease a lot of the problems of life-cycle management and method invocation.

COM provides good opportunities to hang the system. If you are planning to use COM, you would be well advised to read up on it and understand the requirements imposed by the apartment models before writing any code. This is particularly true of the single-threaded apartment model where invisible windows are created on your behalf that may not always do everything that is required of them. If you are planning to do multi-threaded programming, then you should avoid the single-threaded apartment model. In either case, my experience has been that you should stick with one model or the other. Mixing models in the same address space leads to unpredictable behaviors.

When using COM, I suggest you look for reference books. COM is very complex, particularly if you have not done a lot of component-level programming before. I have been doing object-oriented programming for a long time and thought that surely that would make a nice entry into COM. It helps, but is far from enough. The COM documentation available online is not terribly helpful. It tends to be terse and widely dispersed. The definitive source is *Inside OLE* by Kraig Brockschmidt (Microsoft Press, ISBN: 1556158432). This is

available online with your MSDN subscription. The one that I own and really like is *Inside COM* by Dale Rogerson (Microsoft Press, ISBN: 1572313498). It is very readable and has a lot of good information. It was written by a Microsoft programmer and contains one of my favorite Microsoft quotes: "At Microsoft, we always feel we can improve on a standard." Pretty much says it all doesn't it?

Microsoft has adopted COM very widely in their own programming efforts. The Internet Explorer browser has a very sophisticated set of COM interfaces. Other parts of the system are evolving COM interfaces as well. They are truly eating their own dog food in this regard. That makes one believe in the technology.

MFC

MFC is a big part of the development picture. It provides fairly full-featured implementations of most technologies. Your decision on deployment of MFC is almost the same as your decision to deploy PowerPlant or MacApp in MacOS land. With MFC you get a lot of features but the code is big and slow. I also question the quality of it sometimes. I was debugging in MFC once and I dropped into a section of code that used multiple GOTO statements for flow control. Now I'm not one of those elitists who claim that folks who use GOTO are intrinsically evil. On the other hand I do believe that, in a language such as C++ that has a rich set of flow control and error handling constructs, it is the tool of a lazy designer at best and a bad designer at worst. This was never a major issue with PowerPlant or MacApp in my opinion.

A second important issue is distribution. To use MFC in your application you must make sure that the user has the appropriate MFC DLLs on their machine. They will frequently be present already but you must make provisions to deliver them just in case. For desktop applications delivered on CDs with lots of room, this may not be much of an issue. However, if you are planning network delivery then the extra megabyte or so can be a big issue for you. There is also an option to statically link all of the MFC code into your application, thereby avoiding the separate distribution issue. This is simpler, but makes it more difficult to take advantage of MFC updates in the future.

A good strategy is to use MFC for rapid prototyping, and then go away from it when you are ready to get serious about commercial code.

Al Evans recommends that you read *Mfc Internals* : Inside the Microsoft Foundation Class Architecture by George Shepherd and Scot Wingo (Addison-Wesley, ISBN: 0201407213). This is not a book for the faint of heart, but contains lots of useful information.

Modern OS?

Windows NT is a more modern operating system than what you are used to with MacOS. It has protected memory, preemptive multi-tasking, real virtual memory, and all that cool stuff that modern operating systems need. It holds up well under a lot of abuse. Included with the Win32 SDK is a program called Kill. You can almost always use it to kill a rogue program and get on with your life in places where the reset button was the only answer in MacOS land. Abusing the system is much more difficult as well. In MacOS land it is not uncommon to be forced to re-install system software while making new code. I have never had to do that with NT. I also rarely have to reboot because of program errors. Problems such as overwriting arrays, double freeing pointers, and writing into someone else's address space are not so serious. That is my biggest beef with the MacOS. You are constantly rebooting because of things like that. On Windows NT

you just kill the application and get on with it, which is liberating.

However, it is not a panacea. There are still issues that land it somewhere in between MacOS and UNIX as far as reliability goes. In UNIX land, a crash is a big event to be investigated. On NT you just hit the reset button and move on. It also tends to get into a state where parts of the system are unstable but there is the illusion of stability. I have spent quite some time debugging phantom crashes that would not behave consistently and then went away after rebooting.

There are also cases where a single rogue application can bring the system to its knees. A good example of this is the Visual C++ linker. While you are doing a long link, system performance degrades to the point of being unusable. This is true even on dual-processor systems. So, the moral of this story is: Reboot often and do not get alarmed when the system grinds to a halt.

Possibly the most frustrating thing of all is the overall design, or lack thereof, in the Windows APIs. There are almost always multiple ways to do the same thing, and it is frequently unclear what the difference is. There is also no Gestalt call. Discovering things about the system and the hardware can be challenging. As a Macintosh programmer, you may find the Windows APIs scattered and incoherent. It takes some getting used to.

The Tao

So, you may now ask if I think that it was worth it. My answer would have to be that it depends. In today's climate it is certainly easier to feed your family if you can program Windows. There is definitely something missing for me though. I began programming the Macintosh for the fun of it, and it soon became a passion. I was consumed by it. I lived and breathed the Macintosh, and it became an integral part of who I am. I was hoping for a similar experience with Windows, but it was not to be. I have found it generally lacking in the sex appeal that drove me ever onward in the Macintosh world. I do it because I am paid: were it not for the money, I would not do it. So, my suggestion is to move forward with the Macintosh if that is a viable option for you. I would be willing to bet that you will be happier in the long run.