

# An Introduction to Macintosh Programming for Windows Programmers

This document provides a brief overview of the Macintosh for Windows programmers. It provides general tips, tools and tricks for porting your application and explains some of the similarities and differences between the Macintosh and other platforms. This document is useful to developers who are just starting to investigate Macintosh development.

## Tutorial Contents

### [Chapter 1 - Introduction](#)

### [Chapter 2 - Who Are Your Users?](#)

### [Chapter 3 - Programming Tools](#)

#### [3.1 - Integrated Development Environment](#)

#### [3.2 - PowerPlant Framework](#)

#### [3.3 - Editors/Code Browser](#)

#### [3.4 - Development Process](#)

#### [3.5 - Resource Editors](#)

#### [3.6 - Source Code Control Systems](#)

### [Chapter 4 - Processors](#)

#### [4.1 - 68K and PowerPC](#)

#### [4.2 - Windows Emulators](#)

### [Chapter 5 - Operating System](#)

#### [5.1 - Versions](#)

- [5.2 - File and Application Management](#)
- [5.3 - Console](#)
- [5.4 - Application structure](#)
- [5.5 - Components](#)
- [5.6 - Shared libraries](#)
- [5.7 - Shipping Families of PowerPlant Applications](#)
- [5.8 - Desktop Icons](#)
- [5.9 - Help Online](#)
- [5.10 - Help Tool tips](#)
- [5.11 - Y2K Problems](#)
- [5.12 - “Hot” Plug and Play](#)
- [5.13 - Java Support](#)

## [Chapter 6 - Programming Model](#)

- [6.1 - APIs](#)
- [6.2 - Events](#)
- [6.3 - Localization](#)
- [6.4 - Memory model](#)
- [6.5 - Text Editing](#)
- [6.6 - Inter-Application Communication and Scripting](#)
- [6.7 - Multitasking and Multi-threading](#)

## [Chapter 7 - Files](#)

- [7.1 - File Types](#)
- [7.2 - File Names](#)
- [7.3 - File Structure](#)
- [7.4 - Application Structure](#)
- [7.5 - Document Structure](#)
- [7.6 - Bitmap Files](#)
- [7.7 - Text Files](#)
- [7.8 - Foreign Volumes](#)
- [7.9 - Strings](#)

## [Chapter 8 - Graphics](#)

- [8.1 - 2D Graphics](#)
- [8.2 - Multiple-Monitor Support](#)
- [8.3 - 3D Graphics](#)
- [8.4 - Multimedia](#)

## [8.5 - Game Support](#)

### [Chapter 9 - GUI](#)

#### [9.1 - Human Interface Guidelines](#)

#### [9.2 - Menus](#)

#### [9.3 - Windows](#)

#### [9.4 - Documents](#)

#### [9.5 - Controls](#)

#### [9.6 - Drag & Drop](#)

#### [9.7 - Keyboard](#)

#### [9.8 - Mouse Buttons](#)

#### [9.9 - Palettes](#)

#### [9.10 - Printing](#)

### [Chapter 10 - Resources](#)

#### [10.1 - Creating Resources](#)

#### [10.2 - Visual Resources](#)

### [Chapter 11 - Summary](#)

#### [11.1 - Things To Do](#)

#### [11.2 - Things to Remember](#)

### [Chapter 12 - Some Sample Programs](#)

#### [12.1 - Overview](#)

#### [12.2 - Anatomy of a Macintosh Application](#)

### [Further References](#)

The Apple Store	Hot News	Products	Design & Publishing	Developer	
	About Apple	Support	Education	Where to Buy	

[Search Tips](#) | [Site Map](#) [Extended](#) [Index](#)

[The Apple Store](#) | [Hot News](#) | [About Apple](#) | [Products](#) | [Support](#)  
[Design & Publishing](#) | [Education](#) | [Developer](#) | [Where to Buy](#) | [Home](#)

[Contact Us](#) - [Developer Site Map](#)

Copyright © 2000 Apple Computer, Inc. [All rights reserved.](#)

# An Introduction to Macintosh Programming for Windows Programmers

---

[Table of Contents](#)[Next Chapter](#)

---

## Chapter 1—Introduction

The Mac is quite similar to Windows. The Mac is quite different than Windows. Which statement is true? They both are, of course. Compared to using MS-DOS, UNIX, LINUX, or a Palm Pilot, Macintosh and Windows 95/98 are quite similar. Users of one system can learn the basics of using the other system fairly quickly, and applications present an almost identical user experience on each platform. You will even find that programming for Macintosh is fundamentally the same as programming for Windows—both systems are based on an event-driven Graphical User Interface (GUI) that uses document windows, dialog windows, controls, menus, and so forth. Both systems use resources to describe user interface elements, and both systems present similar sets of APIs for managing windows, menus, dialogs, graphics, and so forth. Both systems even have at least one popular application framework available to make it easier to manage an event-driven GUI.

On the other hand, the systems are different in thousands of details, so porting an existing Windows program to the Macintosh is not an automatic or trivial process. You will have to modify both your C++ code and the resources used to describe user interface elements. You may also have to modify the user interface itself, the way you deal with files, the way you handle separate threads of execution, and other program elements. The easiest programs to port will be those designed with a clear separation between the parts of the program that model the problem domain and the parts associated with the GUI. If, for example, you have written an online banking application, the C functions or C++ classes that represent customers, bank accounts, transactions, and so on should be platform-independent. If the member functions in those classes do not make any calls to

the Windows APIs, then you should be able to port those to the Macintosh very easily. On the other hand, the menus, push buttons, dialogs, et al., displayed on the screen are based on the Windows APIs, your code will have to be rewritten for the Mac OS. It is our job to help you with that process.

[Back to top](#)

---

[Table of Contents](#)

[Next Chapter](#)

[Further References](#)

---

The Apple Store	Hot News	Products	Design & Publishing	Developer	
	About Apple	Support	Education	Where to Buy	

[Search Tips](#) | [Site Map](#)   [Extended](#)   [Index](#)

[The Apple Store](#) | [Hot News](#) | [About Apple](#) | [Products](#) | [Support](#)  
[Design & Publishing](#) | [Education](#) | [Developer](#) | [Where to Buy](#) | [Home](#)

[Contact Us](#) - [Developer Site Map](#)

Copyright © 2000 Apple Computer, Inc. [All rights reserved.](#)

# An Introduction to Macintosh Programming for Windows Programmers

---

[Previous Chapter](#)[Table of Contents](#)[Next Chapter](#)

---

## Chapter 2— Who Are Your Users?

Before we dive into the technology, let's talk about the reason you develop applications: end users. You probably know that the Macintosh is strong in certain markets such as Education, Consumer, Small Office Home Office, and Content Creation (graphics designers, multimedia producers, professional musicians, web-page authors, and so forth). Don't think of Macintosh users as simply buyers in those markets. You should also realize that Mac users are somewhat different than Windows users in their attitude toward computers.

What does this mean for your software? It means that you should pay attention to every little detail. Macintosh users are not very tolerant of software that is hard to install, or hard to use, or has a non-standard user interface. Your software should look and feel like was designed for Macintosh users. If they think it is a warmed-over port of something from another platform, they are unlikely to buy it. Your mission, should you decide to accept it, is to make your Macintosh software a thing of beauty. Take pride in providing a seamless, productive experience for every user. Make use of Apple's [Human Interface Guidelines](#) book while doing the design of your application. Your users will thank you for taking the time to make your application look like a Macintosh application. In return, Macintosh users will love your software and your company. Remember that Macintosh users are among the most loyal consumers in the galaxy. Making great software will pay off on the bottom line!

[Back to top](#)

---

[Previous Chapter](#)

[Table of Contents](#)

[Next Chapter](#)

[Further References](#)

---

The Apple Store	Hot News	Products	Design & Publishing	Developer	
	About Apple	Support	Education	Where to Buy	

[Search Tips](#) | [Site Map](#) [Extended](#) [Index](#)

[The Apple Store](#) | [Hot News](#) | [About Apple](#) | [Products](#) | [Support](#)  
[Design & Publishing](#) | [Education](#) | [Developer](#) | [Where to Buy](#) | [Home](#)

[Contact Us](#) - [Developer Site Map](#)

Copyright © 2000 Apple Computer, Inc. [All rights reserved.](#)



# An Introduction to Macintosh Programming for Windows Programmers

---

[Previous Chapter](#)[Table of Contents](#)[Next Chapter](#)

---

## Chapter 3—Programming Tools

The documentation delivered with Microsoft Visual C++ describes how to use the Microsoft Foundation Classes (MFC) framework to manage the GUI part of your Windows application. This is recommended because an application framework can take care of many complex parts of the GUI code, making your application code smaller and easier to maintain. The same logic applies when creating a Macintosh program; instead of doing all the low-level work yourself, you should consider using an application framework to help with event handling, scrolling, printing, and other potentially messy coding.

### 3.1 - Integrated Development Environment

The first application framework for developing Macintosh applications was called MacApp. Although MacApp is still available from Apple, it is not the framework we recommend for new projects. For the last few years, the preferred framework for new projects has been PowerPlant, shipped as part of the CodeWarrior Integrated Development Environment (IDE) from [Metrowerks Corporation](#).

CodeWarrior is a cross-platform IDE, with development tools for both Mac and Windows

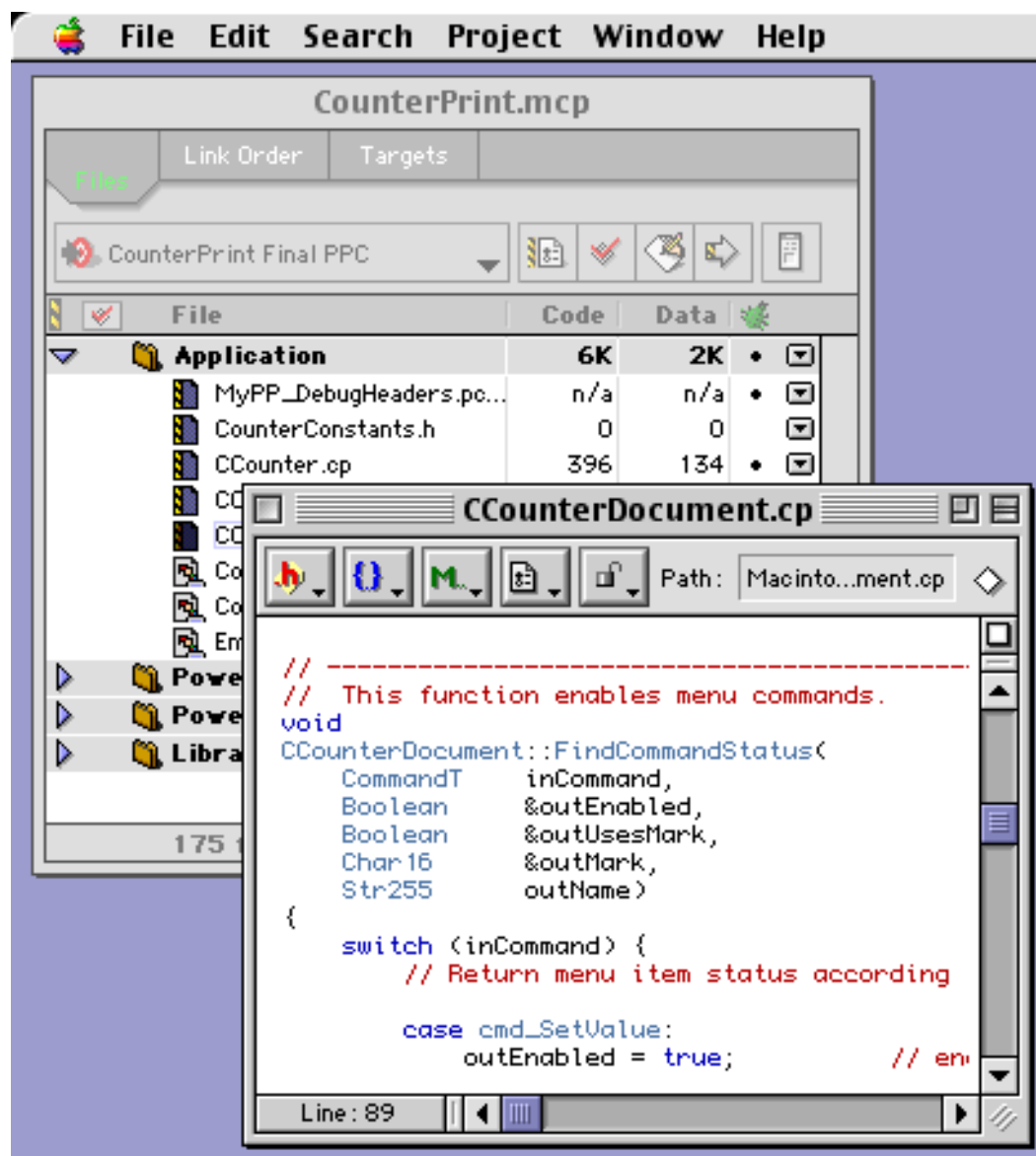
shipping with the Professional version of the product. In fact, the Pro version ships with:

- PowerPlant for C++ Mac programmers.
- A complete set of Java development tools for both platforms.

CodeWarrior is a project-based environment that lets you specify multiple compile targets in a single project. This makes it very nice when simultaneously developing versions of your application for 68K processors and PowerPC processors. The IDE also allows you to open multiple projects simultaneously, which is convenient when stealing code from older projects or sample programs for use with your next project.

### Figure 3.1 - The CodeWarrior IDE

A CodeWarrior project file with one open source file



CodeWarrior also ships with good source-level, symbolic debuggers for each language. They allow you to set breakpoints, single-step, inspect variables and objects, and so forth. CodeWarrior comes with complete documentation, including tutorials, reference documents, online help, etc.

## 3.2 - PowerPlant Framework

PowerPlant is a very sophisticated application framework, with support for basic event-driven user interfaces, plus a wide range of extra goodies. It differs from MFC in a number of ways. One difference is that in MFC, all C++ classes are derived from the common base class `CObject`. There is no common base class in PowerPlant. There are many trees in the framework hierarchy, and many `mixin` classes are defined. A `mixin` class is one that adds additional properties to another class using multiple-inheritance. You will see multiple inheritance used throughout PowerPlant, and you will often use it in your programs.

In case you get the urge to try some of the sample programs shipped with PowerPlant, some of the interesting GUI classes are mentioned in Table 3.1.

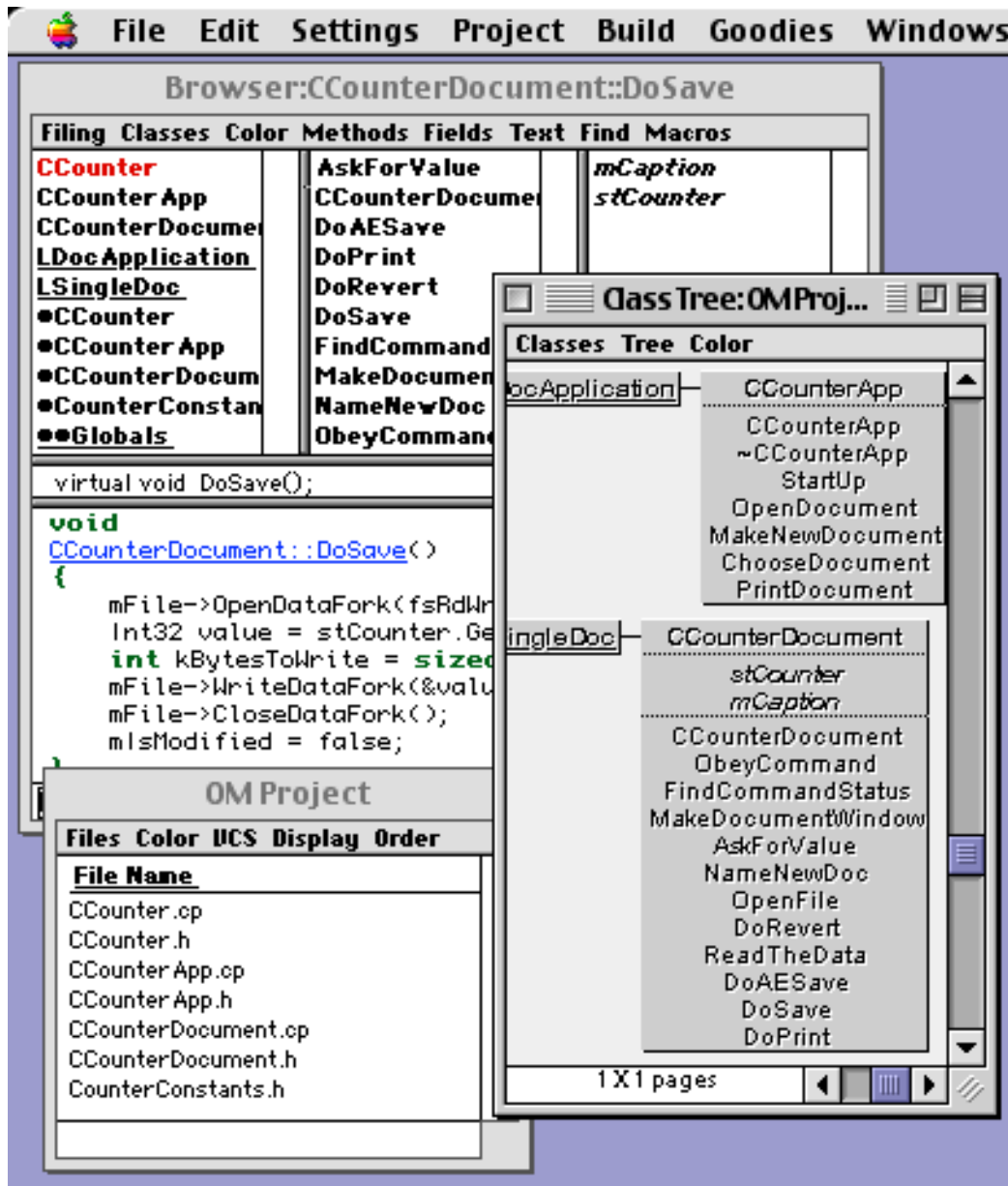
**Table 3.1 - Some important PowerPlant GUI classes**

Class	Description	Comments
<code>LPane</code>	Lighweight component.	Cannot have subpanes.
<code>LControlPane</code> that can broadcast events.	Inherits from <code>LPane</code> and <code>LBroadcaster</code> .	Most buttons, checkboxes, and controls with which the user interacts.
<code>LViewHeavyweight</code> component.	Inherits from <code>LPane</code> .	Pane that can have subpanes. Use a subclass of a view to scroll the contents, or to embed panes inside a container view.
<code>LWindow</code> A window.	Inherits from <code>LView</code> .	A normal or floating window.
<code>LDialogBoxModal</code> dialog.	Inherits from <code>LWindow</code> .	You can put all sorts of controls inside.

## 3.3 Editors/Code Browsers

CodeWarrior comes with a color-coded, syntax-directed editor that can be used to edit individual source files, or can be used as a hierarchical class browser. You may also want to consider using BBEdit by [Bare Bones Software, Inc.](#) or Object Master from [Altura Software](#). Object Master can parse Pascal, C, C++, and Java code and display source files, class browsers, and extremely useful class diagrams, as shown in Figure 3.2.

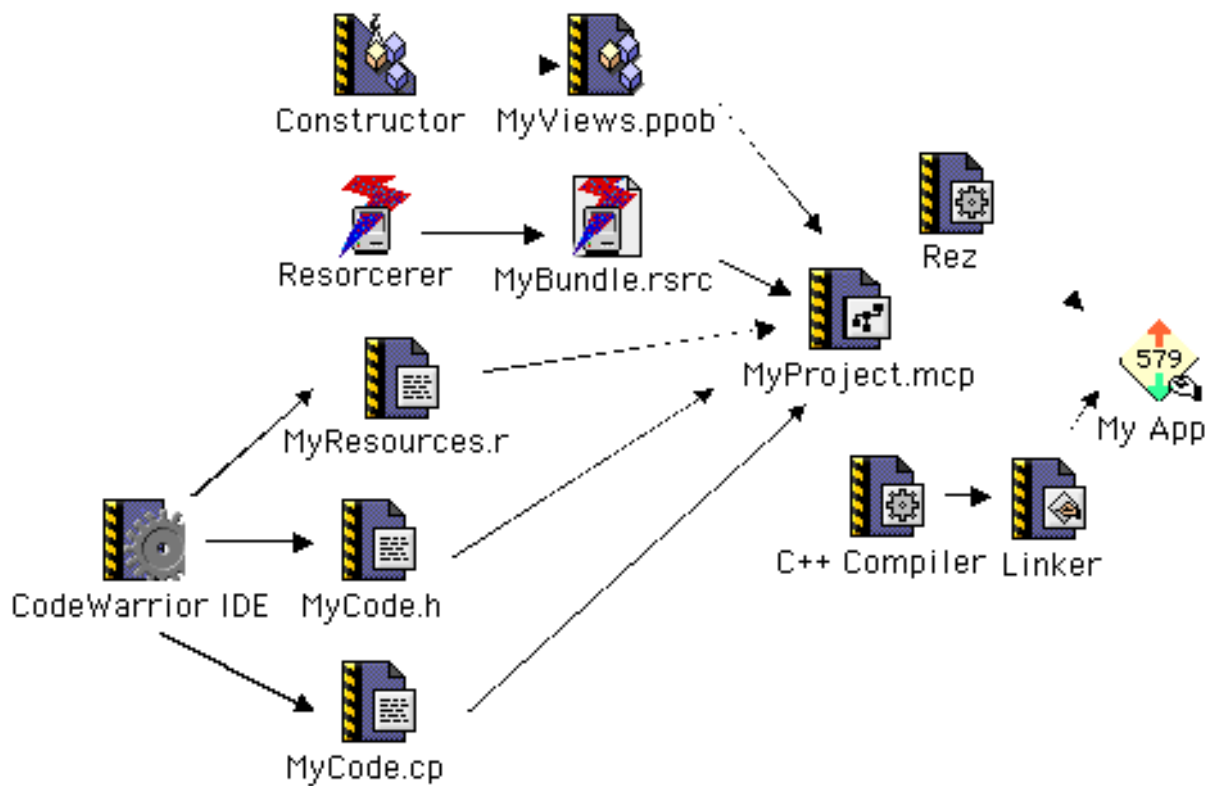
**Figure 3.2 - The Object Master editor/class browser**



## 3.4 - Development Process

Figure 3.3 shows a rough approximation of the application process that you might consider using with CodeWarrior. This probably looks a bit confusing, so let's go more in detail. You can use CodeWarrior's text editor to create your C++ source files and, optionally, text descriptions (i.e., files) of some of your resources. These files are put into a CodeWarrior project file. You can use Metrowerks Constructor resource editor to create resource files with resources, and use the Resorcerer resource editor to create additional resource files. These compiled resource files also go into the project file. When you compile the program, CodeWarrior uses the Rez compiler to compile text-based resources, and uses its C++ compiler to compile your C++ code. Finally the Linker assembles all this into your application. Table 3.2 shows comments about the various files involved in the development process.

**Figure 3.3 - The Development Process**



**Table 3.2 - Files used in development process**

File	Comments
MyViews.ppob	Visual resources created by Constructor resource editor.
MyBundle.rsrc	Other resources created by Resorcerer resource editor, such as Finder icons.
MyResources.r	Text description of some resources, such as menus.
MyCode.h	C++ header file.
MyCode.cp	C++ implementation file.
MyProject.mcp	CodeWarrior project file that manages all of the source files.

## 3.5 - Resource Editors

Resources are described in more detail in [Chapter 10](#), but we'll mention some of the resource editors used in the typical development process in this Chapter.

### Constructor for Visual Resources

CodeWarrior comes with a special editor called Constructor for creating and editing visual resources used to describe windows, dialogs, and controls such as buttons and menus. You should use Constructor for these types of resources. There are many other non-visual resource types, such as text strings, Finder information, and so forth that cannot be edited by Constructor.

### Editors for Other Resource Types

There are two popular resource editors available for these other resource types. One is

called [ResEdit](#), available free from Apple. The other is called Resorcerer, available from [Mathemaesthetics](#). Use these editors for non-visual resources such as text strings, Finder information, and so forth.

## Text-based Resource Descriptions

There is another way to create resources that involves creating text file descriptions of each resource, and compiling these descriptions into binary resource files using a resource compiler called Rez, which ships with CodeWarrior. This latter approach is less intuitive than using a mouse-driven editor such as Resorcerer, but has the virtue that the text-based version is easy to print as part of the source code, and is easy to manage in any kind of source code control system.

## 3.6 - Source Code Control Systems

The Metrowerks web site lists seven version control systems that can be fully integrated into the CodeWarrior IDE, including:

- CVS plug-in by [Electric Fish, Inc.](#)
- CW Projector by [Electric Fish, Inc.](#)
- Visual SourceSafe for Macintosh by [Metrowerks](#)
- Perforce by [Perforce Software](#)
- Projector plug-in by [Metrowerks](#)
- PVCS Version Manager on the Macintosh by [Synergex](#)
- VODOO by [UNI SOFTWARE PLUS](#)

We suggest that you investigate these products if you require source control in your development process.

[Back to top](#)

---

[Previous Chapter](#)

[Table of Contents](#)

[Next Chapter](#)

[Further References](#)

---

The Apple Store	Hot News	Products	Design & Publishing	Developer	
	About Apple	Support	Education	Where to Buy	

[Search Tips](#) | [Site Map](#) [Extended](#) [Index](#)

[The Apple Store](#) | [Hot News](#) | [About Apple](#) | [Products](#) | [Support](#)  
[Design & Publishing](#) | [Education](#) | [Developer](#) | [Where to Buy](#) | [Home](#)

[Contact Us](#) - [Developer Site Map](#)

Copyright © 2000 Apple Computer, Inc. [All rights reserved.](#)

# An Introduction to Macintosh Programming for Windows Programmers

---

[Previous Chapter](#)[Table of Contents](#)[Next Chapter](#)

---

## Chapter 4—Processors

### 4.1 68K and PowerPC

Older Macintosh computers were based on Motorola's 68000 series of processors, but all machines made in the last few years use the PowerPC chip. Your first big decision is whether to create two versions of your software: one for 68K and one for PowerPC.

Apple's newest system software (Mac OS 9.0 and the upcoming Mac OS X) works only with PowerPC. If your software is targeted at power users or new users (such as iMac purchasers), then you can safely forget about 68K and develop just for PowerPC.

On the other hand, if you want to reach millions of users of older computers (for example, your Mom or many school districts), then you may have to build a version for each processor. For the most part, the programming APIs are the same for each processor, so you can write one set of sources that applies to both processor families. Moreover, you can handle slight target differences with conditional compile directives. When your application is finished, you can package both versions into a single installer. The user can then choose to install one of three versions:

1. The 68K version for older Macs. (The PowerPC system software contains an emulator for the 68K, so 68K-based software will also run on a PowerPC-based Mac. It will, as you might expect, run quite a bit slower than code compiled native for the PowerPC.)
2. The PowerPC version for newer Macs.



3. A fat version that will run on either processor.

If you are not sure what to do, start by developing a PowerPC version, since this will run best on your PowerPC-based development system.

## 4.2 Windows Emulators

We should also mention Intel's x86 series of processors. Software written for those processors, such as normal Windows 95 applications, can also be run on a Macintosh in one of two ways. First, there are plug-in cards for the Mac that contain a Pentium processor. You can use these cards to run Windows programs in one partition, while simultaneously running Macintosh programs in another partition. These cards cost many hundreds of dollars, however, so many Mac users choose the lower-cost alternative of using a Windows software emulator. The two major emulators each cost less than \$200, and allow users to run DOS or Windows applications on a Macintosh, however the results are not always speedy. Unfortunately, neither solution is satisfying to most Mac owners. The x86-based hardware solution is expensive, while the software emulator is relatively slow. What Mac users really want is Macintosh software running native on their PowerPC-based Macintosh. That's where you come in. A genuine Mac application not only provides better performance than a Windows emulator, but also provides the user experience that Mac users prefer.

[Back to top](#)

---

[Previous Chapter](#)

[Table of Contents](#)

[Next Chapter](#)

[Further References](#)

---

The Apple Store	Hot News	Products	Design & Publishing	Developer	
	About Apple	Support	Education	Where to Buy	

[Search Tips](#) | [Site Map](#) | [Extended](#) | [Index](#)

[The Apple Store](#) | [Hot News](#) | [About Apple](#) | [Products](#) | [Support](#)  
[Design & Publishing](#) | [Education](#) | [Developer](#) | [Where to Buy](#) | [Home](#)

[Contact Us](#) - [Developer Site Map](#)

Copyright © 2000 Apple Computer, Inc. [All rights reserved.](#)

# An Introduction to Macintosh Programming for Windows Programmers

[Previous Chapter](#)
[Table of Contents](#)
[Next Chapter](#)

## Chapter 5—Operating System

### 5.1 - Versions

The first Macintosh was released in 1984, and as far as users could tell, it didn't have an Operating System. It booted to a "desktop" on which the user could manipulate icons, but that did not feel like a traditional operating system. There was one underneath, of course, but it had no name-- now it does. In a burst of creativity, it was named the Mac OS. Major generations and generic information about them are shown in Table 5.1.

**Table 5.1 - Mac OS versions**

Version	Comments
1 - 6	Obsolete.
7.x	Still used by millions of people. For both 68K and PPC. Additional APIs.
7.6.1	Best version of System 7.
8.0 and 8.1	PPC and 68040. Millions of copies sold in 1998. More APIs.
8.5 and 8.5.1	PPC-only. Still more APIs.
8.6	PPC-only. Yet more APIs.
9.0	PPC-only, latest version. Yet more APIs.
X ("ten")	Will use Carbon APIs. (The new Carbon APIs are discussed in <a href="#">Chapter 6.1.</a> )

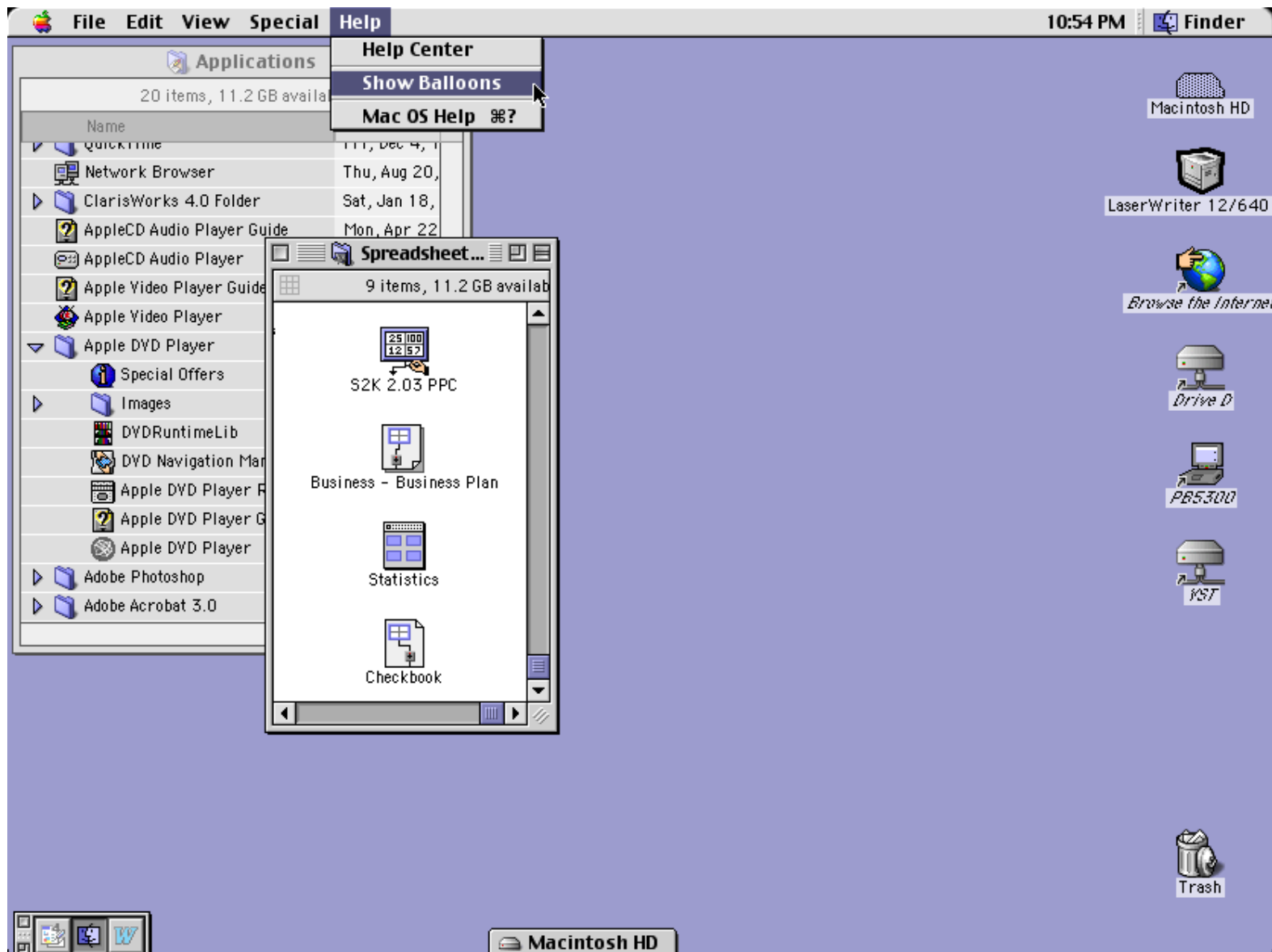
You might think of Mac OS X as analogous to Windows 2000, while previous versions of Mac OS are more akin to Windows 95/98. What does this mean to you? If you are going to target power users and/or new users, then you can safely use all the APIs available in System 7 and later. Many applications out there have gone slightly further by requiring the use of System 7.6.1 or later. If you are targeting users of iMacs or the G3 Towers, you can assume they are running some version of Mac OS 8 or later.

If you are developing heavy-duty server-based applications, you might consider targeting them at Mac OS X, which will provide demand-paged virtual memory, protected address spaces, and preemptive threading.

### 5.2 - File and Application Management

Most PCs are configured to boot to Windows 95/98/NT. All Macs are configured are configured to boot into the Finder of the OS, which is the GUI shell application. The Finder is analogous to the combination of File Manager and Explorer. It manages the desktop, displaying icons for each file, mounted volume, and other devices. A typical desktop display is shown in Figure 5.1.

Figure 5.1 - Typical Macintosh Desktop



This is similar to what you see in a Windows desktop, but there are some significant differences, including:

- An icon with an italicized name is an alias to the real file or volume. An alias is like a Windows shortcut, but more sophisticated. An alias finds its associated target using APIs defined in the Alias Manager. (API calls defined for the Mac OS are grouped into packages called managers. For example, the APIs for manipulating windows will be found in the Window Manager Chapter of the documentation. There other Chapters for Dialog Manager, File Manager, and so forth. Not everything is called a manager, however. The graphics routines are simply called QuickDraw, the multimedia routines are called QuickTime, and so on.) If the user moves the target file to another directory, the alias is automatically updated, so users don't "lose" the document or application when it is moved. You can use Alias Manager calls in your software to create and manipulate very robust references to files.
- The menu bar is always at the top of the desktop. Unlike the Windows taskbar, the Mac menu bar cannot be moved to the sides or bottom of the screen.
- Only the desktop has a menu bar—individual windows do not. The menu bar changes to show a different set of menus when a different application becomes active.
- The Apple menu is somewhat like the Windows Start menu, but different customs are used. For example, Windows applications usually install a shortcut to themselves in Windows' Start folder. You could install an alias to your application in the Apple menu, but it is considered bad form.

### 5.3 - Console

Windows 95 and NT users can choose to bypass Windows Explorer's graphical user interface and open a console window to manipulate files and launch applications. The Mac OS does not provide any end-user console window. There are GUI-based ways to manipulate groups of files, and AppleScript provides a GUI-based alternative to batch files, but there is no OS-based text-based console window for end-users. As a programmer, you can get a text-based command environment by using Apple's Macintosh Programmers Workshop (MPW). MPW provides a command-interpreter shell, and a complete UNIX-like command language that allows everything from file manipulation to very sophisticated text-manipulation using regular expressions.

You can even provide a text-based console window inside any particular application, using Metrowerk's SIOUX library (SIOUX = Standard Input Output UniX, pronounced "sue"). However, just because you can provide a console window does not mean you should. Console windows provide a very poor interface for normal people to use. You may often use them for displaying debug messages, but we do not recommend that developers include a console window in a shipping application.

### 5.4 - Application structure

While Windows applications are designated with an "exe" extension, Mac applications cannot be recognized by their file names. Instead, applications are recognized by the operating system because they have a file type of 'APPL' (see [Chapter 7.1](#) for details on file types). Users do not know about file types, so they recognize an application by its icon. Applications icons typically look different than document icons, as we'll see in [Chapter 5.8](#).

### 5.5 - Components

Active-X and COM allows Windows programmers to build or buy small, reusable GUI components from which more complex user interfaces can be assembled. These are typically created using either Visual BASIC or Visual C++. The Mac OS does not have standard component technology like this, so this reuse mechanism is not available. However, there are many other reuse mechanisms with application frameworks like PowerPlant.

### 5.6 - Shared libraries

The other type of reusable component available to a Windows programmer is the dynamic-link library. The Mac OS also provides a re-entrant shared-library technology, supported by the Code Fragment Manager APIs. You can easily compile any set of functions or classes into a "CFM" shared library, and then call upon the services of this library from any application. This is particularly valuable if you are creating a family of programs, since you can put common routines into a shared library, thereby saving both disk space before it is loaded, and memory footprint when being used by multiple applications. CFM is a PowerPC technology, while the companion CFM-68K technology supports code fragments on 68K processors.

#### Shipping Families of PowerPlant Applications

As one example, if you were to build a number of applications based on the PowerPlant framework, you could compile the framework itself into a shared library that could be accessed by each application. You do have to be careful, because object-oriented frameworks have a "fragile base class" problem, so a new version of the framework will require recompiling programs based on that framework. This means that if you (1) get a new version of PowerPlant, (2) compile it into a new shared library, and (3) build a new application based on that new version, then all applications based on that shared library must be recompiled, and then reinstalled.

#### Warning:

Shipping shared libraries is not encouraged. Mac users would prefer to avoid the problems of DLL proliferation and version conflicts that face many Windows users. We recommend shipping your application as a single executable file. This makes it easy to install, and even allows users to uninstall it without fear. You should only ship part of your code as a shared library if you are delivering a large family of programs and the code sharing represents a significant resource saving.

Finding a Shared Library

When an application tries to open a shared library, the OS first searches for it in the same directory as the application, then searches in a “magic” directory called the Extensions folder located in the active “System Folder.”

Note: The “System Folder” is identified by the System software as the one with the active “Finder” and “System” files. It does not actually have to be named “System Folder”.

5.7 - Compound Documents and Embedding

The Mac OS currently does not actively emphasize the use of OLE-type embedded objects. It still supports the OpenDoc compound document technology, but no longer plans enhancements to the technology nor encourages developers to use it. Microsoft ported the OLE technology to the Mac platform a few years ago, so you might be able to use that in a pinch.

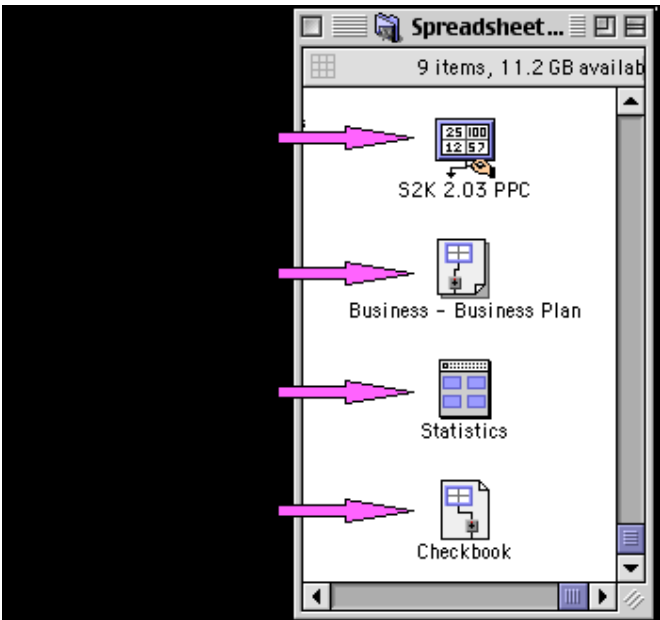
5.8 - Desktop Icons

When you create your new application, programming skills are not enough. You will also need to be a bit of an artist, so you can design useful icons for the application and for every kind of document that can be produced by your application. Users need the visual feedback from these desktop icons to help them understand how to use the program, so there are certain design guidelines to follow. The various types of icons are summarized in Table 5.2, while Figure 5.2 shows an example of such a set of icons.

Table 5.2 - Various types of desktop icons

Icon type	Comments
Application	Should show a hand holding a pen; should hint at the type of data being created by the application.
Normal document	Should look like a piece of paper with the top-right corner folded; should hint at the type of data contained within.
Stationery document	Should look like a pad of paper. These are prototype documents that can contain data. They are cloned by the Mac OS when opened by the user.
Special documents	Design should hint at their use.

Figure 5.2 - Desktop icons for an application and documents



As an example of these ideas, again consider the icon examples in Figure 5.2. These icons are from an application called Spreadsheet 2000, which uses a dataflow visual programming language for defining math operations. The icons (designed by the programmer, Steve Wilson) build on this metaphor. The application icon shows a grid containing numbers, since the application is a type of spreadsheet. The normal and

stationery document icons show a grid object connected to a math operator, just as the user will do when using the program. The “Palette” document icon stores the data for a user-defined floating palette of math operators, so the icon looks a like a floating palette window. This kind of attention to detail when designing icons enhances the visual appeal and helps the user better understand how to use the program.

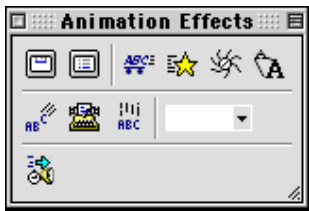
## 5.9 - Help Online

Windows users expect each application to provide a Help menu that can be activated by the F1 function key. This leads to a hypertext-linked set of online help files. Macintosh users expect to have similar online help support, also accessed through the Help menu. In the past, Mac programmers were encouraged to build their online help using a technology known as Apple Guide, but that is being replaced by a system known as [HTML Help](#).

## 5.10 - Help Tool tips

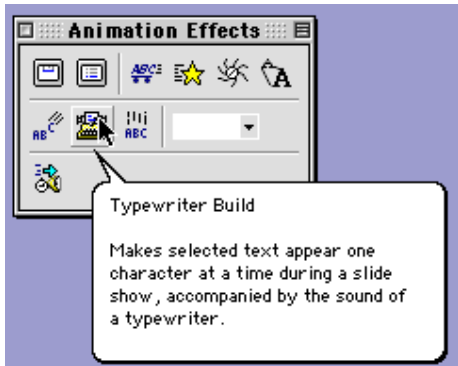
There is nothing more intimidating to many users than a tool palette full of small-sized icons, such as shown in Figure 5.3. If you intend to subject your users to this type of palette, then you better provide an easy way for users to determine the meaning of each icon.

**Figure 5.3 - A window that needs Help**



The most common Windows technique is to provide tool tips for each program element. The Mac OS does not have tool tip APIs, but does provide a similar technology called Balloon Help. The user can request little balloons (containing text and/or pictures) to pop up as the mouse moves over various program elements, as shown in Figure 5.4. Balloon Help is supported by the [“Help Manager” APIs](#).

**Figure 5.4 - Balloon help in action**



## 5.11 - Y2K Problems

The Mac does not have a Y2K problem. Right from the beginning, the Mac system clock was designed to work correctly up until the year 2019. If you use the Mac OS APIs for date and time manipulation, your program’s dates should work. See the [Year 2000 web site](#) for more detailed information.

## 5.12 - “Hot” Plug and Play

Users do not want to have to reboot just because they change their computer’s configuration. The Mac OS allows them to dynamically do things like this:

- Connect or reconnect to a network.
- Change the relative location of monitors.
- Move the menu bar from one monitor to another.
- Hot plug a USB/FireWire device

- Hot swap a hard disk into or out of their PowerBook laptop.

Be sure that your application can handle this type of dynamic behavior.

## 5.13 - Java Support

There are a number of Java Virtual Machines (JVMs) available on the Mac platform, both as standalone JVMs and embedded in browsers. The long-term plan is to have both applications and all Mac browsers using Apple's JVM, known as the Macintosh Runtime for Java (MRJ). See our [developer web site dedicated to MRJ](#) for more detail.

[Back to top](#)

---

[Previous Chapter](#)

[Table of Contents](#)

[Next Chapter](#)

[Further References](#)

---

The Apple Store	Hot News	Products	Design & Publishing	Developer	
	About Apple	Support	Education	Where to Buy	

[Search Tips](#) | [Site Map](#) [Extended](#) [Index](#)

[The Apple Store](#) | [Hot News](#) | [About Apple](#) | [Products](#) | [Support](#)  
[Design & Publishing](#) | [Education](#) | [Developer](#) | [Where to Buy](#) | [Home](#)

[Contact Us - Developer Site Map](#)

Copyright © 2000 Apple Computer, Inc. [All rights reserved.](#)

# An Introduction to Macintosh Programming for Windows Programmers

[Previous Chapter](#)
[Table of Contents](#)
[Next Chapter](#)

## Chapter 6—Programming Model

### 6.1 APIs

When the first Macintosh was introduced in 1984, the Inside Macintosh documentation described about 400 Pascal procedures and functions. These routines were collectively called the Mac User Interface Toolbox. You will still see these Mac OS routines referred to as Toolbox calls. At the time, these were considered to be a very large, complex set of APIs, so they were organized into related groups called Managers. The first set of seventeen Managers included the ones shown in Table 6.1.

**Table 6.1 - Some original Mac OS technologies (Managers)**

#### Original APIs

Manager	Purpose
Memory Manager	Heap-based memory allocation.
Resource Manager	Manage structured data called resources.
QuickDraw	2D graphics
Font Manager	Font management



Event Manager	Handling events
Window Manager	Handling windows
Control Manager	Buttons, scroll bars, etc.
Menu Manager	Menu bar, menus, and menu items
TextEdit	Simple text manipulation (up to 32K characters)
Dialog Manager	Model and modeless dialog windows
Scrap Manager	Clipboard Cut, Copy, and Paste
Utilities - various categories	Miscellaneous stuff: dates, times, bit manipulation, string manipulation, etc.

## Newer APIs

The original Managers shown in Table 6.1 still exist today, but they have been enhanced many times over the years, and many new technologies have been added. Just to give you a flavor for the variety, a few of the newer ones are listed in Table 6.2. In addition, the APIs are now focused on the needs of C and C++ programmers rather than Pascal programmers. There are about 8,000 C functions, spread over about 240 header files, defined in the “Universal Headers” for these basic APIs.

**Table 6.2 - A few of the newer Mac OS technologies**

Manager	Purpose
Navigation Services	Provides user interface for managing files - replaces Standard File Manager.
Translation Manager	Helps translate files from one format to another.
Apple Game Sprockets	Sound, imaging, input devices, network gaming.
Apple Information Access Toolkit	For indexing, searching, and summarizing of text across a large set of documents.
Find By Content API Sherlock application	Find By Content is a new system-level search facility implemented as a Code Fragment Manager library. The Sherlock application is a client of Find By Content and uses its search facilities for performing content-based searches, both locally and on the Internet. Because each Internet search site has its own particular format for query and response information, the Sherlock application uses plug-ins that describe data formats used by these sites for formatting queries and parsing response data.
Unicode Utilities	Allow applications and text service components (such as input methods) to perform various operations on Unicode text, for example, Unicode key translation.

AppleScript	An end-user scripting language that enables you to control Macintosh applications and system software directly, without the use of input devices such as the mouse or keyboard. The AppleScript Language Guide, Finder Guide, and other documents can help you write scripts to manipulate images, move files, maintain Web site content, and provide automation of work flow type business processes.
Multiprocessing Services	Routines that allow your application to create separate threads of execution called tasks. Tasks are preemptively scheduled on the available processors in the system, even if there is only one processor. Third-party hardware is available with multiple processors.
Thread Manager	Provides threads of execution within your application.
Network Services Location Manager	A protocol-independent way for applications to discover network services that are available in the local network. Your application can use the NSL Manager to obtain a list of network services, display the list, and allow the user to choose and connect to a particular network service.
Speech Manager	Support for synthesized speech in applications. The Speech Manager is under evaluation for Carbon.

## The Carbon APIs

[Carbon](#) represents the core set of APIs you will be able to use to build [Mac OS X](#) applications that can also be deployed on Mac OS 8. Using Carbon, you will be able take advantage of preemptive multitasking, memory protection, and dynamic resource allocation in Mac OS X while maintaining maximum source code compatibility with your current applications. Any new projects started in 1999 should use the Carbon APIs.

Carbon will include most of the Mac OS functions that developers rely on today, while adding a small number of new functions to support Apple's goal of providing a more robust, responsive, and productive computing experience. You should become familiar with the [Carbon Porting document](#) when designing your application for maximum compatibility.

When running in Mac OS X, Carbon applications gain these advantages:

- Greater stability. Preemptive multitasking and protected address spaces will help prevent errant applications from crashing the system or other applications.
- Improved responsiveness. Each application will be guaranteed processing time through preemptive scheduling, resulting in a more responsive user experience.
- Efficient use of system resources. Applications will dynamically use memory and

other system resources based on actual needs rather than predetermined values.

## 6.2 Events

Event handling in the Mac OS is somewhat different than in Windows. In both cases, your application must handle events to respond to user input. The differences are in the philosophy and the details of how events are handled.

### Windows OS Event Model

The Windows programming model focuses on the OS sending messages to windows. All interesting user interface elements are treated as windows, including icons, menus, scroll bars, etc.

In the Mac OS, icons, menus, and scroll bars are not windows. Furthermore, events must be retrieved from the OS event queue by your application, and it is the application's job to handle the events and to request behavior from windows, scroll bars, icons, etc. as necessary. While there are around 400 different message types defined in the Windows APIs, there are only 16 types of events defined in the Mac Toolbox.

Conceptually, the Windows OS “pushes” events to each window's message queue, while a Mac application's event loop must “pull” events from the operating system.

### Mac OS Event Model

The event-handling model is the central focus of any Macintosh application. Your application must handle events to respond to user input, to redraw and activate/deactivate windows when necessary, and even to respond to messages from other applications or from the scripting system software. Furthermore, you must properly handle events to ensure that cooperative multi-tasking works correctly, so other applications get processing time as they need it. To quote from Apple's [Inside Macintosh](#) programmer documentation:

At the core of every Macintosh application is the application's event loop. The event loop is that piece of code in an application that processes and responds to user actions and other events. You can use the Event Manager to retrieve information about these actions. For example, you can get information that tells your application whether the user pressed a key or the mouse button, whether one of your application's windows needs updating as a result of the user moving windows, or whether some other hardware or software action requires a response from your application.

You should structure your application so that it can respond to events and so that the user is able to perform tasks in any order. For example, a user should be able to type text in a window, select a graphic and copy it, open a new document, paste in the graphic, open another document, and then go back to the first window to select text and change its typeface, size, or style.

Your application should respond to events in a way that lets the user switch between your application and others whenever the user chooses to do so (for example, by clicking in a window belonging to another application). Your application should also yield time to other applications when it isn't busy. System software provides a cooperative multitasking environment that allows

users to switch between many open applications and that allows applications to receive available processing time when other applications aren't using the processor. System software coordinates the scheduling of processing time between your application and other applications.

You can also let your application communicate with other applications in order to request services or information from another application or to provide services to other applications. You can use the Event Manager or Apple Event Manager to do this.

## **Event-Handling in PowerPlant**

All this sounds scary, and it would be if we didn't have frameworks to make life easier. Just as MFC handles most of the details for a Windows application, PowerPlant will do most of the event handling for your Mac application. PowerPlant code runs the main event loop, gets each event, figures out what kind of event it is, and dispatches it to the appropriate PowerPlant object. You can therefore write a simple Mac application that handles windows, menus, push buttons, scrolling, and so forth without ever touching an actual Mac OS Event Record. What could be easier?

## **6.3 Localization**

The Mac OS provides a wide range of technologies to support localization of your program, including:

- Script Manager, which oversees script systems and gives you access to their features. This is relevant if you are writing a multiscrypt text-handling application.
- Text Services Manager, which requests actions and information from applications, and sends data to them. This is important if you want your application to support text input in a 2-byte script system. Your application will then work with multiple script systems and many input methods.
- Apple Type Services for Unicode Imaging, which your application can use to draw and print Unicode text. ATSUI automatically handles many aspects of text display, including simple and sophisticated typography and line layout.
- Date, Time, and Measurement Utilities, which provide information about the current date, time, location, time zone, and units of measurement in a local format.
- Keyboard and International Resources, which specify how keyboard input is converted to text, and help with number and date formatting for each locale.

## **6.4 Memory model**

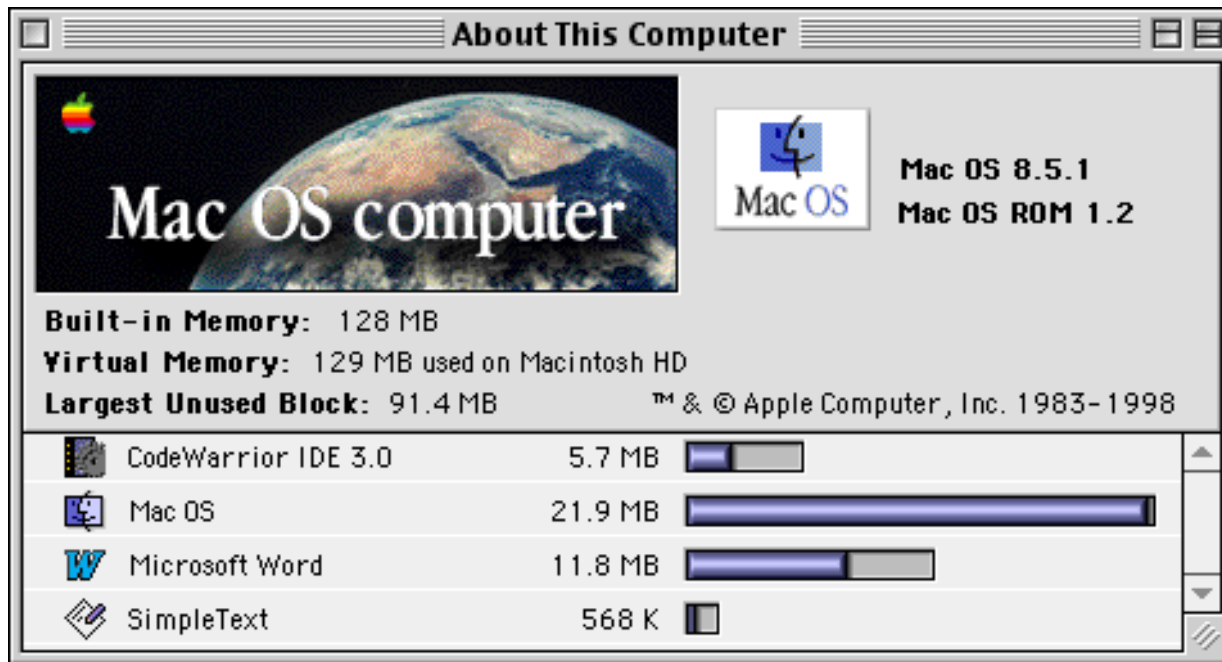
The Macintosh memory model differs from that of Windows in significant ways, including:

- Memory allocation among applications
- Virtual memory
- Memory protection
- Management of data structures on the heap

### **Memory Allocation Among Applications**

The current Mac OS memory model partitions the address space among each of the running applications. You can get an idea how this works by simply choosing the first item (About This Computer) on the Finder's Apple menu. This will display a memory allocation window, as shown in Figure 6.1.

**Figure 6.1 - Memory allocation for each application**



This example shows that the system software is using about 22 MB of the computer's 128 MB of RAM, and also shows the memory partitions allocated to the three applications that are currently running. For example, CodeWarrior is running in a 5.7 MB partition, although it has filled less than half of that so far.

When you create your application, you must test thoroughly to determine the preferred amount of RAM, and then specify that number in a resource of type 'SIZE'. If you request too much, then the user may not have room to launch other applications, which annoys users to no end. If you request too little, then your application may run out of heap space, leading to various runtime problems. Power users can adjust an application's partition size before it is launched, but it is better if the developer makes a good estimate when creating the program. (Select the application icon with the mouse, and choose "Get Info..." from the file menu. You can set preferred partition size in the resulting dialog.)

A common situation is that your application may temporarily need a large block of memory. If you need it all of the time, you will simply have to run your application in a large partition. If you only need memory for occasional short-term use, then you can use Mac OS routines for allocating in temporary memory. Temporary memory is taken from memory that is reserved for, but not yet used by, other applications. As soon as your application is finished with the need for that allocation, it should call Memory Manager routines to free the memory. When your applications quits, the operating system will free any temporary memory still being used by the application. A good use for temporary memory is to store data associated with open documents. Then if memory gets crowded, the user has the option of simply closing one or more of those documents.

## Virtual Memory

“But wait!” you exclaim, “Windows has demand-paged virtual memory, so the user doesn’t have to worry about running out of memory. Doesn’t the Mac OS have virtual memory?” The Mac OS does have virtual memory, so the user can ask for a logical memory that is bigger than physical memory using the Memory Control Panel. What the Mac OS does not yet have is demand-paged virtual memory. That feature is coming soon in Mac OS X, which will make both users and developers feel much better. We should mention that if the user turns on the current Mac virtual memory, then Code Fragment Manager shared libraries are already file-mapped and use demand-paged virtual memory. Applications will have to wait for Mac OS X.

## Allocating Memory on the Heap

Because Mac programs have to work with a fixed memory allocation, Mac programmers must worry about managing heap space efficiently. When the Mac system software was first developed, heap space was at premium, so various schemes were developed by the system software engineers for allocating blocks on the heap. The result is that many Mac OS data structures are allocated on the heap as non-relocatable blocks accessed by pointers, while other important data structures are allocated on the heap as relocatable blocks accessed by handles, which are indirect pointers.

NOTE: Windows handles are not the same as these Mac handles, although they are conceptually similar. Windows handles are references to Windows API objects. Certain Windows functions will return these handles, while others take them as arguments. Mac handles are specifically pointers to pointers. It is common for Mac programmers to dereference a handle to get a pointer, or double-dereference a handle to get to the data structure on the heap.

If the heap gets filled, the Mac OS Memory Manager routines will compact the heap, moving relocatable blocks to make more room for new data structures.

What does this all mean to you? It means that you will often find functions that take various types of handles as arguments, or that return handles as function results. You must then double-dereference the handle to access the data structure itself. You often have to lock the data structure so it won’t move while you access the data, since a structure referenced by a handle is expected to be relocatable. Remember that the Memory Manager may move handle-based structures during heap activity. (See [Memory Madness](#), by Peter N. Lewis.)

You do not have to worry about relocation for C++ objects allocated with the new operator. These objects are non-relocatable and accessed by direct pointers, just as with C++ on Windows.

## Memory Protection

Up through Mac OS 8.5, the memory heap is not protected, and page zero is not read or write protected. This means, unfortunately, that you can overwrite memory that belongs to another application or to the system software, which leads to very bad things. You should therefore be very careful about memory allocation, and dereferencing pointers and handles. Programmers who use the PowerPlant framework generally have much less trouble in this area than do programmers who write directly to the Mac Toolbox APIs,



which is one reason why we recommend using PowerPlant. There are various tools to help you diagnose memory management problems, including [Onyx Technology's](#) QC and Spotlight, and the ZoneRanger tools included with CodeWarrior.

## 6.5 Text Editing

The Mac OS provides many kinds of text-handling support, from displaying simple characters to complex, multi-language text processing. It includes routines to:

- Draw characters, strings, and lines of text.
- Work with fonts in any size, style, and language.
- Edit simple, multi-language text paragraphs.
- Format numbers, dates, and times.
- Convert formats among different countries or regions.
- Handle text in any language.
- Input text input in Asian languages such as Japanese.
- Handle text-encoding conversions.

(This Chapter was summarized from Apple's Inside Macintosh: Text documentation.)

### TextEdit

The [TextEdit APIs](#) support basic text formatting and editing capabilities, including text display in multiple scripts. TextEdit manages fundamental text processing tasks on text limited to 32 KB. You can use the TextEdit routines in many kinds of applications, such as spreadsheets, online (data-entry) forms, online advertising programs, simple programming-language or text-file text editors, electronic mail programs, drawing and painting programs with simple text-editing features, and electronic note cards. However, TextEdit was not designed to be used to implement word-processing applications that manipulate lengthy documents.

There are, however, third-party libraries that support rich text editing with millions of characters, embedded graphics, scripting with AppleScript, and even integration with PowerPlant. One example is the [CWasteEdit library](#), which is very easy to use.

### QuickDraw Text Routines

You can use the [QuickDraw](#) text routines to measure and draw text ranging in complexity from a single glyph to a line of justified text containing multiple languages and styles. In addition to measuring and drawing text, the QuickDraw text routines also help you to determine which characters to highlight and where to position the caret to mark the insertion point. These routines translate pixel locations into byte offsets and vice versa.

### Font Manager

The [Font Manager](#) is a collection of routines and data structures that you can use to manage the fonts your application uses to display and print text. The Font Manager takes care of reading font data from font resources and creating the bitmap images that QuickDraw uses to display text.

### Text Utilities

The [Text Utilities](#) provide you with an integrated collection of routines, ranging from modifying the contents of a string, to sorting strings from different languages, to converting times, dates, and numbers from internal representations to formatted strings and back. These routines work in conjunction with QuickDraw text-drawing routines to help you display and modify text in applications that are distributed to an international audience. There are routines for basic operations such as accessing a string resource and comparing two strings for equality.

### **Text Services Manager**

[Text Services Manager](#) describes how text-processing applications can communicate flexibly and efficiently with utilities that provide services to those applications. Applications that need input methods, spell-checking, hyphenation, and so forth can use the Text Services Manager to search for, obtain information about, and communicate with those utilities. Utilities can use the Text Services Manager to request actions and information from applications, and to send data to them. In particular, if you want your application to support text input in a 2-byte script system, you should use the Text Services Manager. Your application will then work with multiple script systems and many input methods.

### **Text Encoding Converter**

The [Text Encoding Converter](#) is the primary converter for converting between different text encodings, such as converting a file's text from the Windows Latin-1 character set to the Mac OS Roman character set. When you use the Text Encoding Converter, neither the source encoding nor the destination one must be Unicode, although either can be.

### **Unicode Converter**

Use the [Unicode Converter](#) if you are writing applications based in Unicode, such as a word processor, or for a file system that operates in Unicode, and need to convert to or from Unicode.

## **6.6 Inter-Application Communication and Scripting**

As a Windows programmer, you may have used OLE Automation with Visual BASIC for Applications (VBA) to script behavior for an application or a suite of applications. OLE 2.0 has been made available for Macintosh programmers by Microsoft during the last few years, but it has never grown popular on the Mac. VBA has not been made available to Macintosh developers. Mac programmers have instead chosen to use Apple's powerful and elegant [AppleScript](#) technology to support automating complex processes.

AppleScript is built on top of the Apple Event Object Model, which is a very flexible technology for inter-application communication. The PowerPlant framework provides considerable support for making your application scriptable, and we highly recommend that you support this feature. An application that can be scripted is described as being scriptable. With some extra work, you can make your application recordable, which means that a user can create a complex AppleScript by simply turning on a script recorder and then operating the application in a normal manner. This is very powerful and elegant. Recordable applications can be very valuable to power users. Danny Goodman has written a number of good books about end-user scripting with AppleScript, while Inside Macintosh provides a very detailed volume on the underlying technologies and APIs. (See,



for example, Danny Goodman's *AppleScript Handbook*, by Danny Goodman, Random House, 1994, ISBN 0-679-75806-2. and *Inside Macintosh: Interapplication Communication*, by Apple Computer, Addison-Wesley, 1993, ISBN 0-201-62200-9.)

## 6.7 Multitasking and Multi-threading

Imagine how you (a typical programmer) might be using your new, very fast Power Macintosh. You might simultaneously be using a web browser to surf through online documentation, using an FTP utility to download a new set of sample programs, and using CodeWarrior to compile your new application. Furthermore, while the FTP download continues, you might want to switch to CodeWarrior and create and edit a new C++ source file, even as other files in your project are being compiled.

Although Mac OS 8.5.1 and below do not support full, preemptive multitasking, they certainly allow the scenario described above. Everything will work fine, provided that each application developer properly supports cooperative multitasking, and, if necessary, some kind of "multi-threading." Mac OS 8.6 adds support for preemptive threads with the addition of [MultiProcessing Services](#).

At the heart of each Macintosh application is the main event loop, which repeatedly calls the `WaitNextEvent()` function to get the next event from the operating system. You will not have to write this code if you use PowerPlant, because the main event loop code is inherited from the `LApplication` class.

### Cooperative Multitasking

What happens when more than one application is running? There will then be a number of applications that each are repeatedly calling `WaitNextEvent()` to see if an event has occurred. If there is no event pending for an application (it actually receives a null event), then the application can perform some processing, or it can relinquish control for a specified time so that the other (background) applications can do useful work. (You must set an application's `canBackground` flag in the 'SIZE' resource to get background processing time. You can do this with Resorcerer or ResEdit.) While an application is processing an event, it takes over the machine, which means that no other applications can do anything until control is relinquished. A well-behaved application should only spend a short time processing an event, so that other apps can also get processing time. If each application can call `WaitNextEvent()` at least a few times every second, then the user can switch between each of the running applications with no noticeable delay. This makes the machine feel quite responsive.

On the other hand, if a piggish application takes control for many seconds, then the user cannot switch to other tasks, and other applications cannot continue to do their processing. This can make even a fast computer feel really slow.

So what does this mean to you? Consider, for example, a user choosing one of your menu items. The PowerPlant framework will respond by calling an `ObeyCommand()` function in your code. Your code should spend at most a few tenths of a second in that response function, so you must do your work quickly. This will be easy if the menu command is simply changing the color of a box from red to green. A more difficult situation occurs if

the menu command must trigger a long, complex operation such as recalculating a large spreadsheet or doing complex image processing. In that case, you need to break the operation up into short bursts of activity, saving the state of the calculation between stages. To do this, you can use:

- Idle-time processing
- Cooperative threads
- Preemptive threads (only if using Multiprocessing services)
- Idle-time Processing

The PowerPlant framework has a hook for calling a routine every time a null event is detected. PowerPlant also has another hook for calling a routine every time any type of event is processed. In either case, your idle routine must complete its work quickly - ideally in less than 0.1 seconds. If the idle routine takes too long, then normal event processing will be delayed, and the user will notice sluggish response to mouse and keyboard events.

### **Cooperative Threads**

The Mac OS Thread Manager performs creation, scheduling, and deletion of threads. It allows multiple independent threads of execution within an application, each having its own stack and state. Your application can change the scheduler or context-switch parameters to optimize for a particular usage pattern. All scheduling occurs in the context of the currently executing application. When the application gets processing time, the application's threads get time via the Thread Manager. Applications which are sleeping do not get their threads executed. Threads are per-application: when the application gets time, its threads get time. Threads are not given a priority and are scheduled in a round-robin fashion or as dictated by a "yield to" call or a custom scheduler.

PowerPlant provides Thread class wrappers to make it easy to use the Mac OS Thread Manager.

### **Preemptive Threads**

Mac OS X will provide operating system-level support for preemptive threads. There will still be limitations in their use, however, since not all of the Toolbox calls are safe to call from a thread that can be preempted. In the meantime, there is one way to get preemptive threads: [Multiprocessing Services](#).

The Apple Multiprocessing API provides a set of calls that allow an application to create separate threads of execution called tasks. Tasks are preemptively scheduled on the available processors in the system, even if there is only one. See [Technote 1104: Interrupt-Safe Routines](#) for more information.

### **Warning:**

Many Toolbox calls cannot safely be called from preemptive threads. It is safe to call pure C++ algorithms, but many user interface Toolbox functions are not safe to call. Safe routines will be designated by the FOR\_SYSTEM8\_PREEMPTIVE flag in the Mac OS 8 interface files.

## Summary

As a good citizen, you should design your event-handling code to do your processing in short bursts, thereby allowing other applications to get processing time as well. CodeWarrior provides flexible support for idle-time processing, and even provides Thread classes wrapped around Apple's Thread manager APIs if you want to structure your program to use multiple threads of execution. For vertical-market, high-performance applications, you should consider using multiple-processor Macs and the Multiprocessing Services APIs.

[Back to top](#)

---

[Previous Chapter](#)

[Table of Contents](#)

[Next Chapter](#)

[Further References](#)

---

The Apple Store	Hot News	Products	Design & Publishing	Developer	
	About Apple	Support	Education	Where to Buy	

[Search Tips](#) | [Site Map](#) | [Extended](#) | [Index](#)

[The Apple Store](#) | [Hot News](#) | [About Apple](#) | [Products](#) | [Support](#)  
[Design & Publishing](#) | [Education](#) | [Developer](#) | [Where to Buy](#) | [Home](#)

[Contact Us](#) - [Developer Site Map](#)

Copyright © 2000 Apple Computer, Inc. [All rights reserved.](#)

# An Introduction to Macintosh Programming for Windows Programmers

[Previous Chapter](#)
[Table of Contents](#)
[Next Chapter](#)

## Chapter 7—Files

Macintosh files differ from Windows files in a number of important ways, including:

- The way document files are mapped to their applications.
- Macintosh files have two “forks.”
- File names.
- Data file formats.

We'll discuss these differences in this Chapter.

### 7.1 File Types

Windows file types are specified by three-character extensions, such as “doc” for MS Word files, or “bmp” for bitmap files. Macintosh file types are not indicated by the file name, but rather by an invisible four-character identifier such as 'TEXT', 'PICT', or 'APPL'. These file types are not known to users, but are understood by the Finder, which uses them to match documents to their associated applications. Some common file types are shown in Table 7.1.

#### Table 7.1 - Common file types

File Type	Description	Application
'APPL'	Executable application	Launched by Finder
'shlb'	Shared Library	Managed by Mac OS
'TEXT'	Simple text in the data fork	Many
'PICT'	Graphics (“picture”) file	Many
'URL '	Bookmark file	Netscape Navigator

## 7.2 File Names

On most versions of the Mac OS, file names can contain up to 31 characters. To an MS-DOS user, this represents long file names, since it’s a big improvement over “8.3” file names. To a Windows user, this might sound like short file names, since Windows file names can be up to 255 characters. In truth, Mac files names are long enough for most users to have never run into a problem.

Since the Mac was first introduced in 1984, Macintosh files names could contain any characters in an 8-bit extended ASCII character set, so be prepared for characters such as: '\*+\$%é and even trademark symbols ®, Greek letters >, Japanese Yen symbols ¥, and the symbol for the English pound £. This means you can have a file named “\$#@!&\*^%!!”, even if it sounds like you're cursing.

### Referencing Files

One character not allowed in file names is the colon “:”, since that character is the path separator, equivalent to “\” on Windows. This should not be a major concern to you, however, since program code should not refer to files by manually assembling the complete path name. When creating new files or opening existing ones, the File Manager APIs provide more elegant ways to deal with the files, based on unique file reference numbers.

#### Warning:

Do not refer to files by their hard-coded full path names. The full path specification will break if the user moves or renames the file. It can also get you into trouble because the Mac file system allows the user to mount multiple volumes with the same name. Always refer to an open file using the file system specification record (the FSSpec record). Always use the [Alias Manager](#) to obtain a reference to a file that will be correct even if the file is later renamed and/or moved to another directory by the user.

You should not have to reference full path names in your code because:

- Users find and name files or folders using the [Standard File or Navigation Services](#) APIs that use FSSpec records.
- The operating system will hand your program a file alias containing an FSSpec when the user opens a file from the Finder.

In the rare situation that you need to manipulate files in the special “System Folder”, you can use the `FindFolder( )` function to access those files.

### Improved File System

Mac OS 8.1 includes support for Apple's new volume format, [HFS Plus](#) (also known as Mac OS Extended), with support for long Unicode file names. HFS Plus also provides other benefits, such as larger files, larger volumes, file-size-dependent allocation blocks, and much faster file I/O. Apple has not, however, yet released APIs to let the developer access these HFS Plus longer file names.

If your application needs to manage a large number of files, you should know that the Mac file system is better at managing deep/narrow file system trees, in contrast to Windows which is better at handling shallow/wide trees.

## Volume Names

The Windows OS refers to mounted volumes using drive letters. Macintosh volumes have names, just like files. If your program should need to refer to a volume, however, you should not use the name. You should instead programmatically get the volume reference number from the Standard File or Navigation Service routines.

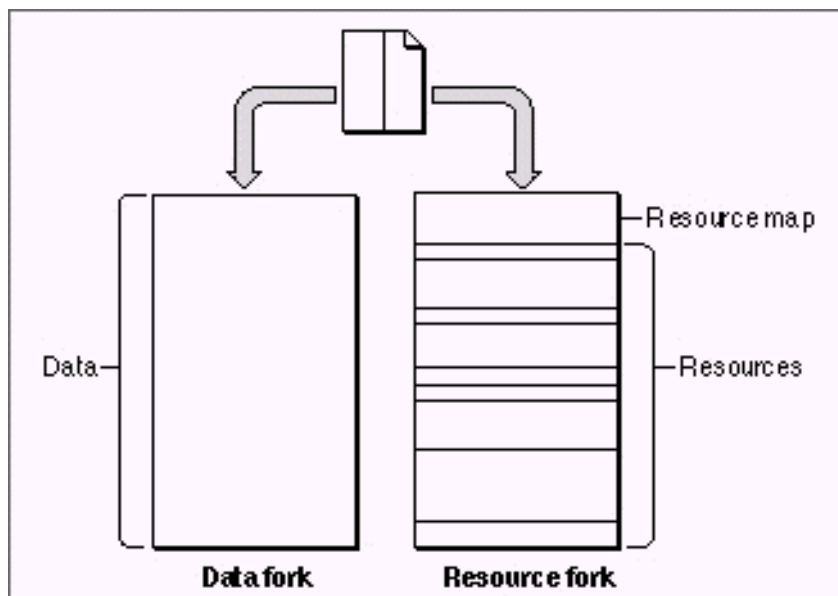
## 7.3 File Structure

To quote [Inside Macintosh](#):

“Many operating systems treat a file simply as a named, ordered sequence of bytes (possibly terminated by a byte having a special value that indicates the end-of-file).” As illustrated in Figure 7.1, however, each Macintosh file has two forks, known as the data fork and the resource fork.

A file's resource fork contains that file's resources. If the file is an application, the resource fork typically contains resources that describe the application's menus, dialog boxes, icons, and even the executable code of the application itself. If the file is a document, its resource fork typically contains preference settings, window locations, and document-specific fonts, icons, and so forth.

**Figure 7.1 - The two forks of a Macintosh file**



A file's data fork contains the file's data. It is simply a series of consecutive bytes of data.

In a sense, the data fork of a Macintosh file corresponds to an entire file in operating systems that treat a file simply as a sequence of bytes. The bytes stored in a file's data fork do not have to exhibit any internal structure, unlike the bytes stored in the resource fork (which consists of a resource map followed by resources). Rather, your application is responsible for interpreting the bytes in the data fork in whatever manner is appropriate. The data fork of a document file might, for example, contain the text of a letter.

Even though a Macintosh file always contains both a resource fork and a data fork, one or both of those forks can be empty. Document files sometimes contain only data (in which case the resource fork is empty). More often, document files contain both resources and data. Applications compiled for the 68K processor family generally contain resources only (in which case, the data fork is empty). Application files can, however, contain data as well.

What does this mean to you? It affects the structure of the application and perhaps the structure of your documents, as described below.

## **7.4 Application Structure**

Applications generally have their user interface elements described in resources that are all located in the resource fork. If you examine an application with a resource editor, you will see resources for windows, dialog items, menus, user-visible text strings, and so forth. Furthermore, applications compiled for 68K processors have their compiled code in 'CODE' resources. On the other hand, applications compiled for PowerPC processors have the binary executable code in “code fragments” in the application's data fork, while the GUI objects are still in the resource fork.

## **7.5 Document Structure**

Documents created by your application should keep their main (non-optional) data in the data fork. For example, simple text files—the equivalent of a “.txt” file created by the Windows Notepad utility program—store only the text in the data fork. Any configuration information, such as the font used to display the text, would be kept in special resources in the resource fork. If your application must handle documents that are used on both the Windows and Mac platforms, then you should not store anything in the resource fork of the document.

## **7.6 Bitmap Files**

Bitmapped graphics files in Windows are commonly of type 'bmp'. This is not a standard Macintosh type. The most common Macintosh image file is of type 'PICT'. A PICT file is basically a collection of QuickDraw graphics instructions saved in a Toolbox data structure called a Picture. These instructions can encode either a bitmap or a collection of structured graphics commands. For example, if the user opens a “paint” document in the popular AppleWorks (formerly called ClarisWorks) application, and draws stuff with various paintbrushes, the resulting collection of bits can be stored in a PICT file. PICT files containing bitmaps are compressed by QuickDraw, and you can even choose the compression scheme if QuickTime is installed.

On the other hand, the user could open a “draw” document and compose an image using



objects such as lines, ovals, polygons, text, etc. This document could also be stored as a PICT file. Both files may look good on the screen, but the structured graphics version will look much better when printed. Structured-graphics files can print at the full resolution of the printer since they contain the original QuickDraw commands to draw the objects and text. One powerful feature of QuickDraw pictures is that your application can add something called PicComments to a picture as it is being created. These comments can be used to provide hints to a printer driver on how to produce optimum output. A PostScript printer driver, for example, may convert PicComments into PostScript code.

There are, of course, many other graphics file types used by graphics professionals on the Mac. These include GIF and JPEG files used to build web pages, and Encapsulated PostScript (EPS) files used by high-end graphics programs like Adobe Illustrator. These high-end programs use PostScript because QuickDraw has limitations, such as no support for curves, or rotation of bitmaps, graphics objects, or text. Apple's cross-platform QuickTime 3 provides APIs for importing a wide range of graphics formats directly into an offscreen graphics world (GWorld). Still image formats supported include BMP, GIF, JPEG, PhotoShop, PICT, PNG, SGI, Targa, and TIFF. QuickTime can also import digital video, digital audio, animation, MIDI, and other file formats. It's very cool.

## 7.7 Text Files

Windows, UNIX, and Macintosh text files differ primarily in the way that each line in terminated, as shown in Table 7.2.

**Table 7.2 - Line terminations vs. OS**

Operating System	Line terminator
Windows	CR + LF
UNIX	LF
Macintosh	CR

This can cause end-users trouble at times, but all good Macintosh programming tools can transparently display text files from any platform. In fact, these tools generally have a simple way of converting text files from one format to another. This means you can painlessly import your Windows source files into Macintosh development tools, and painlessly export them back again as necessary.

## 7.8 Foreign Volumes

The Mac OS can transparently mount foreign file system volumes, such as a floppy disk or ZIP disk formatted for Windows. The Mac OS File Manager routines allow you to find and open those files as easily as if they were Macintosh files, although it is still your problem to determine how to understand the data contained in those files. The [Mac OS Translation Manager](#) APIs may be able to help you in this process, however. As an example of access to a foreign volume, the "Drive D" alias shown in the lower-right side of [Figure 5.1](#) represents an alias to the D Drive of a Dell Latitude laptop, running Windows 95. It behaves like a Macintosh AppleShare server on my Ethernet network



using AppleTalk protocols because the Dell is running Miramar Systems PC MACLAN software.

## 7.9 Strings

Strings can be confusing to programmers who are new to the Mac, due to the history of the Toolbox. The original Toolbox APIs were defined for Pascal programmers, so they used Pascal strings instead of C Strings. Even though the Toolbox APIs are now defined for C/C++ programmers, the function arguments still take Pascal strings for backward compatibility. What does this mean? A Pascal string starts with a size byte specifying how many characters follow (from 0 to 255), while C strings are null-terminated, and can be of any length. For example, the string “dog” would be specified as follows:

```
Str255 pString = "\pDog" ;           // bytes = 3 D o g
char* cString = "Dog" ;              // bytes = D o g null
```

The PowerPlant framework provides a very useful string manipulation class named `LStr255` that allows you to create string objects easily from Pascal strings, C strings, or even string resources. You can then do all sorts of string manipulations, and return ordinary Pascal strings when you need to pass them to a Toolbox call. Many of the PowerPlant methods directly take `LStr255` objects as arguments.

[Back to top](#)

---

[Previous Chapter](#)

[Table of Contents](#)

[Next Chapter](#)

[Further References](#)

---

The Apple Store	Hot News	Products	Design & Publishing	Developer	
	About Apple	Support	Education	Where to Buy	

[Search Tips](#) | [Site Map](#) | [Extended](#) | [Index](#)

[The Apple Store](#) | [Hot News](#) | [About Apple](#) | [Products](#) | [Support](#)  
[Design & Publishing](#) | [Education](#) | [Developer](#) | [Where to Buy](#) | [Home](#)

[Contact Us](#) - [Developer Site Map](#)

Copyright © 2000 Apple Computer, Inc. [All rights reserved.](#)

# An Introduction to Macintosh Programming for Windows Programmers

---

[Previous Chapter](#)[Table of Contents](#)[Next Chapter](#)

---

## Chapter 8—Graphics

---

### 8.1 - 2D Graphics

The standard Macintosh two-dimensional graphics library is called QuickDraw. QuickDraw supports 32-bit color, and has elegant and powerful support for using multiple monitors, as discussed below. It is limited, however, to drawing relatively simple graphics. It provides APIs for drawing lines, ovals, rectangles, polygons, and text. It also provides support for manipulating bitmaps. What it does not provide is support for drawing curves, or applying transformations such as rotation to any of the objects and text. QuickDraw coordinates are based on 16-bit integers. This means that QuickDraw cannot directly handles a view of your data that is longer than 32,676 pixels. (PowerPlant does provide support for larger views, however.) In other words, QuickDraw is limited in much the same ways as the standard Windows 2D graphics system.

### 8.2 - Multiple-Monitor Support

One extremely powerful QuickDraw/Window Manager feature is the support for windows and graphics displayed contiguously across multiple monitors—with little extra work required by the applications programmer. This feature has long been a favorite of Mac programmers, most of whom use two monitors, and sometimes three. This means that you can display source code on one monitor, debugging information on the second, and see the

application running on a third. This is way cool. Furthermore, end-users can do the same thing. This means that users can have three monitors in use simultaneously for a realistic flight-simulator experience, or to display that three-foot wide spreadsheet in all its glory. If you are creating custom software for vertical markets, remember that you might be able to make great use of this multiple-monitor support.

The Mac OS windowing system is able to display the contents of an application window even when the window is shown partially on one monitor and partially on another—even if the monitors are of different size, color capabilities, and bit depths.

It is up to you or your application framework to make sure that the user interface details work properly on multiple monitors, including:

- Window zooming.
- Choosing the right monitor when opening new windows and dialogs.
- Handling dynamic monitor reconfiguration.

See the Window Behaviors Chapter of Chapter 5 of [Apple's Human Interface Guidelines](#) for a detailed description of the proper behavior. Be sure to test this support before shipping your app.

## 8.3 - 3D Graphics

The Mac offers two completely different libraries for modeling and rendering 3D graphics.

### QuickDraw 3D

The first is [QuickDraw 3D](#), which was developed by Apple for both Macintosh and Windows. QuickDraw 3D is an API for creating and rendering real-time, workstation-class 3D graphics. It consists of human interface guidelines and toolkit, a high-level modeling tool kit, a shading and rendering architecture, a cross-platform file format and a device and acceleration manager for plug and play hardware acceleration. QuickDraw 3D is integrated into QuickTime 4, described in the next Chapter.

### QuickDraw 3D RAVE

RAVE is the QuickDraw 3D Renderer Acceleration Virtual Engine, which controls 3D drawing engines (also known as 3D drivers). A drawing engine is software that supports the low-level rasterization operations required for interactive 3D rendering.

RAVE is used internally by QuickDraw 3D. For most 3D drawing and interaction, you should use the high-level QuickDraw 3D APIs. You might, however, use low-level QuickDraw 3D RAVE if:

- You are writing a specialized application (such as a game-development framework) that needs to take advantage of Apple's optimized software rasterizers and any available 3D acceleration hardware.
- You are writing interactive software (such as a game or other entertainment software) that requires the extremely fast 3D rendering that can be achieved with a very low-level, lightweight graphics library.
- You are developing 3D acceleration hardware or software that is to be accessed by any applications rendering 3D images.

## Open GL

Another 3D library you can use is [OpenGL](#), which is now available on the Macintosh. This is an industry-standard library for photorealistic rendering that is available for Windows, UNIX, and now Macintosh. Although QuickDraw 3D is available on both Mac and Windows, Open GL is the preferred platform for getting very fast 3D performance across both Mac and Windows (and UNIX). In general, Open GL will be better choice than QuickDraw 3D for future projects.

By the way, all currently shipping Macs have 3D hardware acceleration, and using either QuickDraw 3D or Open GL will access that acceleration. For example, the blue/white G3 towers introduced in early 1999 ship with the extremely fast ATI Rage 128 3D accelerator, providing state-of-the-art 3D performance.

## 8.4 Multimedia

Apple's [QuickTime library](#) has been an industry standard for a number of years, used in over 11,000 CD-ROM titles and hundreds of new DVD titles. QuickTime includes support for video, animation, digitized sound, MIDI, virtual reality through QuickTime VR, 3D graphics through QuickDraw 3D, and other technologies. As such, there is currently over 7,600 HTML pages (!) of documentation available online. Since QuickTime 4 has been fully implemented on both Mac and Windows, it is an ideal technology to use for developing cross-platform multimedia, music, and graphics titles.

## 8.5 Game Support

Over the last few years, Apple developed the [Game Sprockets APIs](#) specifically to help game developers. Game Sprockets provides libraries for working with sound, imaging, user input, and network gaming. There are similarities between Sprockets and Microsoft's DirectX technology, with Sprockets providing a subset of DirectX capabilities. Game Sprockets provides:

- DrawSprocket for fast drawing to the screen. It supports hiding the desktop and menu bar, double and triple-buffering, control of the monitor's resolution and pixel-depth, frame rates, color management, and so forth.
- InputSprocket for handling input devices, providing the only API support for joysticks and other input devices on USB machines like iMacs.
- NetSprocket for multi-player network gaming, including support for high-performance data transmission, fault-tolerance, and a user interface for hosting and joining games.
- SoundSprocket for simulating 3D sound sources, Doppler motion effects, reverberation, and other goodies.

[Back to top](#)

The Apple Store	Hot News	Products	Design & Publishing	Developer	
	About Apple	Support	Education	Where to Buy	

[Search Tips](#) | [Site Map](#) [Extended](#) [Index](#)

[The Apple Store](#) | [Hot News](#) | [About Apple](#) | [Products](#) | [Support](#)  
[Design & Publishing](#) | [Education](#) | [Developer](#) | [Where to Buy](#) | [Home](#)

[Contact Us](#) - [Developer Site Map](#)

Copyright © 2000 Apple Computer, Inc. [All rights reserved.](#)

# An Introduction to Macintosh Programming for Windows Programmers

---

[Previous Chapter](#)[Table of Contents](#)[Next Chapter](#)

---

## Chapter 9—GUI

---

### 9.1 - Human Interface Guidelines

Apple's Interface Guidelines have been part of the programmer documentation since 1984. Macintosh programmers have painfully learned that Mac users reject programs that violate the expected user interface standards. Mac users expect consistent behavior, including menu placements, document behavior, scrolling, and so forth. Some publishers have successfully survived delivering un-Mac-like user interfaces, because their applications were seen as either indispensable, awesomely innovative, or a game (a flight simulator should behave like an F-16 instead of like a computer). Some of the graphics tools have remained "different" and survived, but even Microsoft has seen the light and delivered the latest versions of Office for Macintosh with a solid, Mac-like interface.

If you are delivering any kind of application except a game, you will increase your acceptance in the Mac market by making your application look and feel like a standard application. You could therefore spend a few days studying the interface guidelines. A better use of your time might be spent using a program like AppleWorks (formerly called ClarisWorks), so that you can see for yourself how things should look and feel.

Even if you are developing a Mac game, you should still provide a proper user interface for setup and configuration. This means that you should not burden users with archaic command-line interfaces, .INI configuration files, magic hidden keystrokes at startup, etc. Your chances for success will be enhanced if your game feels like it was written for the Mac, not ported from Windows.

The next few Chapters call out some areas of particular interest.

## 9.2 - Menus

Macintosh menus behave differently in many details from those of windows. Some differences are listed in Table 9.1. The difference in menu bar location can be seen in Figure 9.2.

**Table 9.1 - Menu differences**

Feature	Windows	Macintosh
Menu bar	Each application window has its own menu bar.	Always at the top of the screen. Only the “main” monitor will have a menu bar if the user is using multiple monitors. Programmers can actually put a menu bar inside an individual window, but it is not common practice, and there is no API support for doing this.
Menus	Drop down and stay down.	Pop up when released before Mac OS 8. With Mac OS 8, menus can remain pulled down.
“About this program” item	Last item on the (right-most) Help menu.	First item in the (left-most) Apple menu.
Keyboard accelerators (command-key equivalents)	Triggered by Ctrl or Alt key.	Triggered by Command (Apple) key.

There are standard keyboard assignments for common menu items. You do not have to provide all of these items, but if you do, it is crucial that you do not change the keyboard equivalents, since practically everyone uses them more or less automatically. Some are listed in Table 9.2.

**Table 9.2 - Keyboard equivalents for some standard menu items**

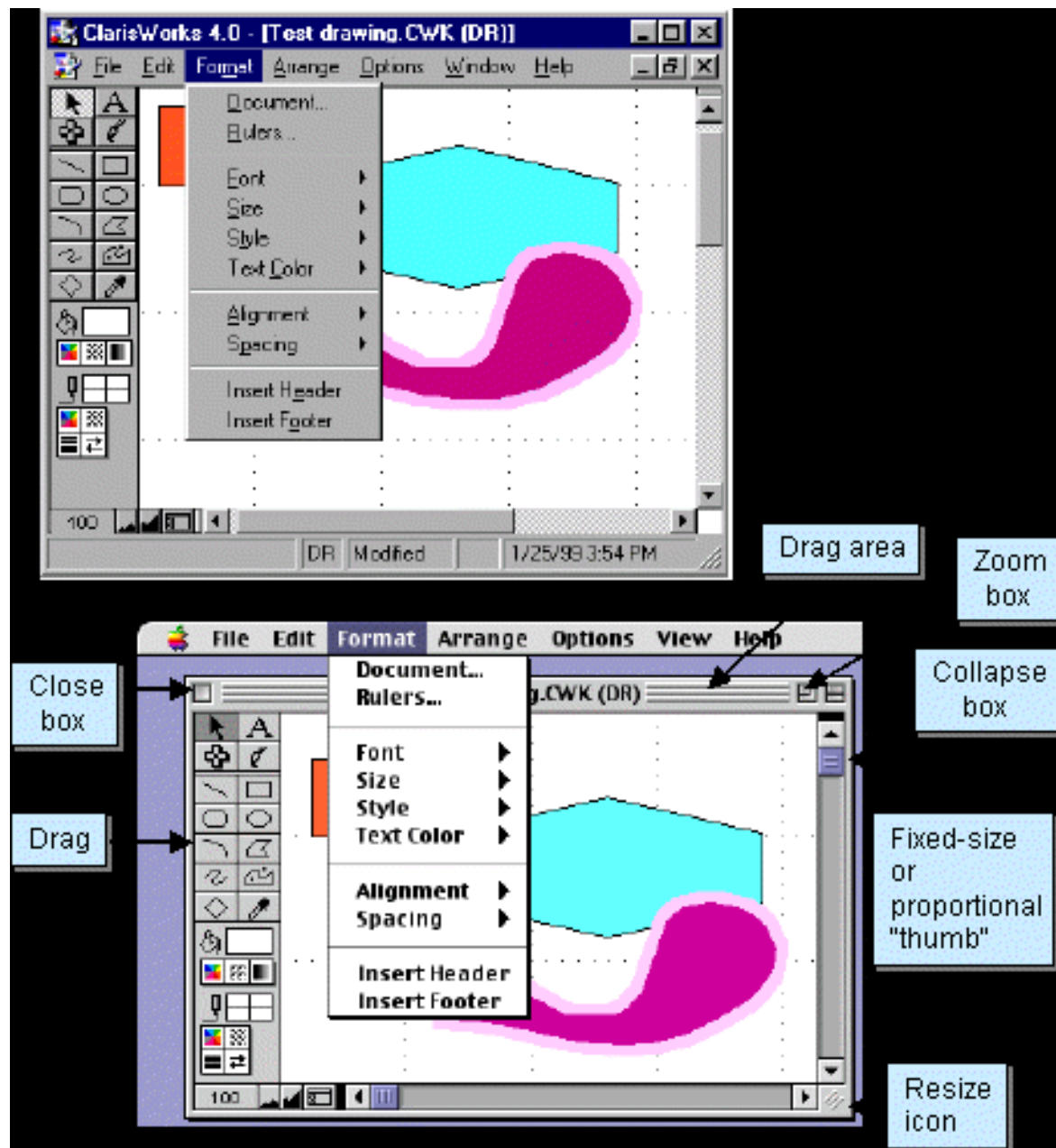
Menu	Menu Item	Command Key
File	New	N
File	Open	O
File	Save	S
File	Close	W
File	Print	P
File	Quit	Q
Edit	Undo	Z
Edit	Redo	No Standard
Edit	Select All	A
Edit	Cut	X
Edit	Copy	C
Edit	Paste	V
Help	Help	?

### 9.3 - Windows

Windows 95/98 and Macintosh provide similar window functionality, but there are many small differences, as shown in Figure 9.1 and summarized in Table 9.3.

**Figure 9.1 - Windows 95 vs. Mac OS 8.5 window.**





**Table 9.3 - Windowing Differences**

Window Behavior	Windows 95/98	Mac OS 8
Drag	Title bar	Title bar or border
Resize	Anywhere on border	Only resize icon
Minimize to button	Minimize icon	Not supported
Maximize	Double-click on title bar	Zoom box
Show only title bar	Not supported	Collapse box
Close	Close box	Close box
Scrolling the contents	Scroll bar thumb changes size to indicate percentage of data that is currently visible.	Scroll bar thumb is fixed in size, or proportional - it is a user option on Mac OS 8.

Most details of how users manipulate windows are transparently handled by the operating

system and/or the application framework. You the programmer will have to decide if a new window should be closable, resizable, etc. The framework will take care of the rest.

## 9.4 - Documents

One of the biggest platform differences has to do with how an application handles multiple documents. The Windows OS is normally described as having two document models, but there are actually three. These include:

- The Single Document Interface (SDI) is for simple applications that only support one open document at a time, such as the WordPad or Paint accessories shipped with Windows 95.
- The Multiple Document Interface (MDI) that provides one parent frame which encloses smaller child windows for each document that is opened. Examples include Office applications like MS Word and PowerPoint.
- An application which can be opened multiple times, with each instance of the application acting like an SDI application. Examples of this include Microsoft Internet Explorer, which treats each open HTML file as a separate launch of the application. Applications like this get most of their functionality from DLL shared libraries, so there is no significant memory hit from opening multiple versions of the application.

The Macintosh OS does not allow an application to be launched more than once at the same time, so the third option is not available. The Mac OS does allow an application to simultaneously have more than one document open, but there are no standard APIs for MDI-type child windows. Instead the Mac OS keeps all windows belonging to a particular application in a single layer. When an application is made active, its layer is brought to the front, so the user can interact with its set of windows.

In all but the simplest applications, the preferred behavior is to allow an application to simultaneously have multiple documents open. It will make your code a bit more complex, but the framework can handle most of the extra work.

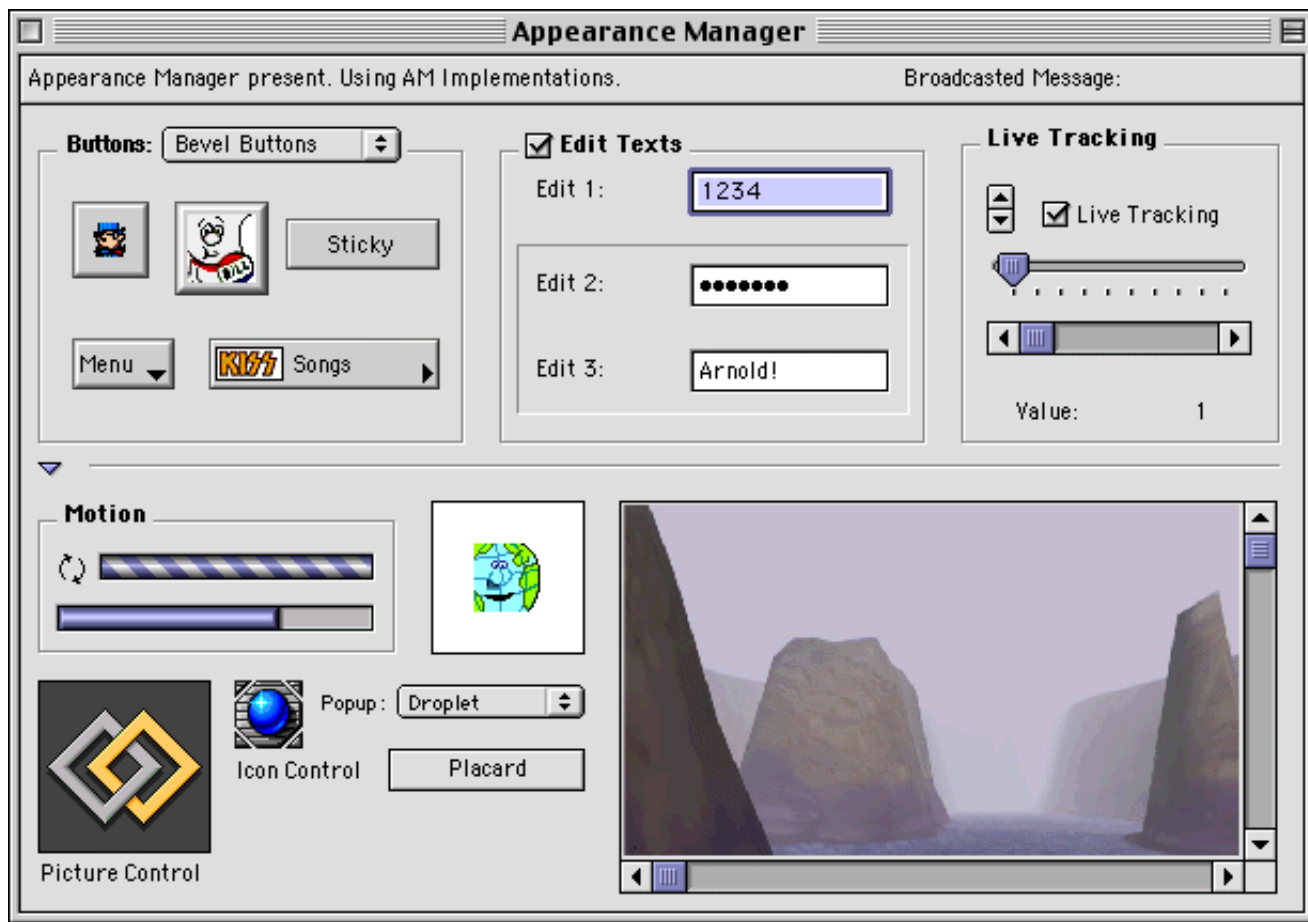
## 9.5 - Controls

Most controls behave pretty much the same on Windows and Macintosh. In fact, almost all common Windows controls are supported in Mac OS 8.0 using the [Appearance Manager](#), although some of these controls were not supported in earlier Mac OS releases. (The Windows “spin control” is not supported on the Mac.) You should recognize a lot of your favorite controls in Figure 9.2 (screen shot of a PowerPlant sample called Appearance Demo). The full Mac OS 8 list includes the following:

- Push button
- Checkbox
- Radio button
- Scroll bar
- Bevel button
- Slider

- Disclosure triangle
- Progress indicator
- Little arrows
- Asynchronous arrows
- Tab control
- Separator line
- Primary & secondary group box
- Image well
- Pop-up arrow
- Placard
- Clock
- User pane
- Editable text field
- Static text field
- Picture
- Icon
- Window header
- List box
- Pop-up menu
- Radio group

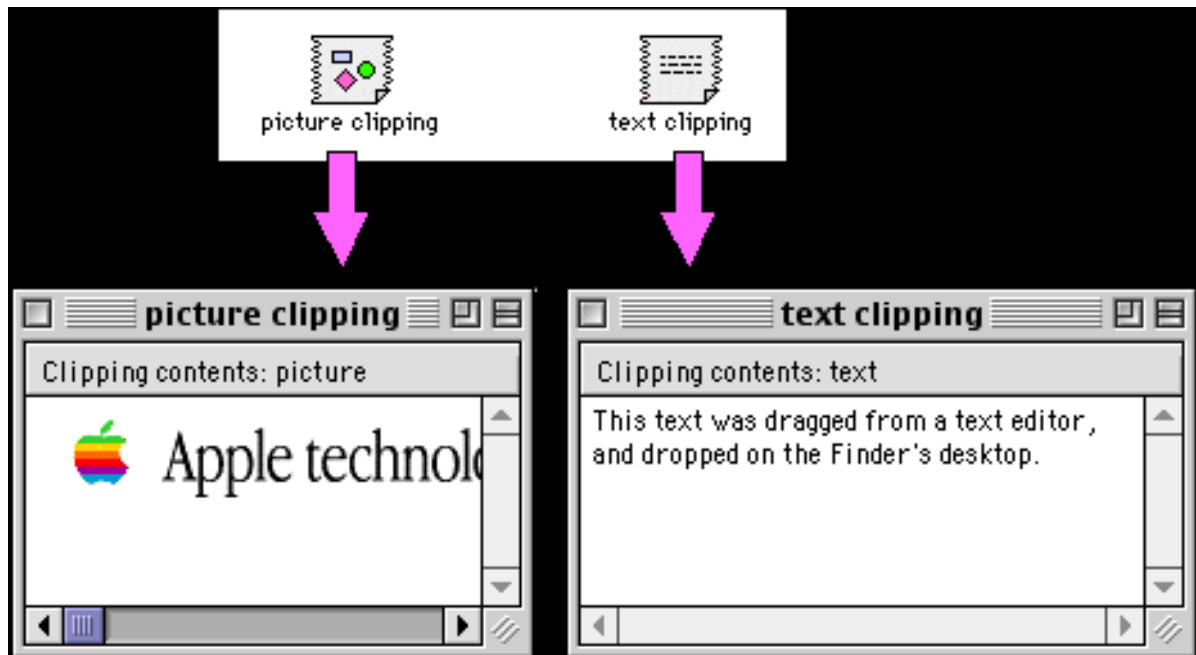
**Figure 9.2 - Some Mac OS 8 Controls**



## 9.6 - Drag & Drop

The Macintosh Drag Manager APIs support drag and drop (DnD) as an elegant, user-friendly way to transfer data inside a single application or between applications. The Finder's support for dragging is quite elegant. If you were to create data inside a productivity application that fully supports DnD, you would find that you could select data in a document window and drag it on the Finder's desktop. The Finder will then create a either a picture or text clipping file on the desktop. A clipping file can be opened to display the text or picture that was placed on the system Clipboard by the DnD operation, as shown in Figure 9.3. If a Finder clipping file is dragged and dropped into an application window, it is treated like a Paste operation from the system clipboard.

**Figure 9.3 - Finder clipping files created by drag and drop**



Drag and drop can also be used to transfer information between applications, and it is much more convenient to the user than using Copy and Paste operations from the Edit menu. Implementing drag and drop is extra work, but there is good support for it in PowerPlant, and there are sample programs that show exactly how to add this feature.

## 9.7 - Keyboard

This may be an oversimplification, but Windows tends to be keyboard-centric, while the Macintosh is more mouse-centric. By this I mean the following:

- Windows keyboards traditionally had more keys. Original Macs keyboards did not even have function keys or Page Up and Page Down keys, even though all the modern ones do. Many Mac users do not use function keys, even if they are on the keyboard.
- Windows users are more likely to navigate through text using arrow keys, while Mac users tend to use the mouse for navigation.
- You can control almost all Windows operations from the keyboard, while many Macintosh programs require the use of a mouse.
- Windows users expect the space bar to be equivalent to clicking the mouse on the current target control. Mac users do not expect the space bar to do anything except add a space character to text (except for use in games).

What does this mean to you as an applications developer? It means that you should not require the use of function keys, arrow keys, or the space bar to operate your program. It is fine to have keyboard equivalents for menu items, but these should generally be Command keys.

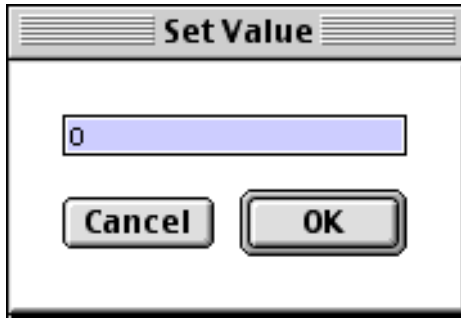
Here are a few other differences:

- The Mac's return key is equivalent to the Window's Enter key. The Mac has a separate enter key that may trigger different functionality. For example, in Spreadsheet 2000, the enter key triggers a recalculation, while the return key triggers a recalculation and also moves the selection down one row.

- The Mac uses Command-period to cancel or abort an operation, analogous to the Windows Esc key.

Another slight difference can be seen in Figure 9.4, showing a modal dialog's typical OK and Cancel buttons. The extra round-rect border on the default OK button shows that it can be triggered from the keyboard with either the return or enter keys. The Cancel button should be triggered by either the esc key or by holding down the Command key while pressing period.

**Figure 9.4 - Default and Cancel buttons in a dialog**



## 9.8 - Mouse Buttons

The Macintosh has traditionally used a one-button mouse, while Windows machines come with a two-button mouse. The Mac mouse is equivalent to the left button on a Windows mouse.

Apple added support for something called contextual menus in Mac OS 8, which are context-sensitive menus that are analogous to the menus displayed by a right-click in Windows. The major difference is that a Mac user with a one-button mouse must hold down the Control key before clicking the mouse to see the contextual menu. Mac users can buy two-button mice with software extensions that let the right button trigger contextual menus, but few Macs are equipped with these.

You should expect to map your Windows left-mouse items to the Mac's normal mouse clicks, and map the right-mouse items to the Mac's contextual menus. A few games have used Apple's InputSprocket library to allow right-click commands to be used with Macintosh two-button mice.

## 9.9 - Palettes

Floating tool palettes (sometimes called windoids) are fairly common in Macintosh applications, but most Mac programs manage them differently than would an equivalent Windows program. For example, is it not common for Mac programs to support "docking" tool palettes into a fixed tool bar. Mac tool palettes tend to be left floating until the user is finished with them—then they are closed.

Let us suggest that massive amounts of tool palettes are not always a good idea. It has become common for Windows applications to feature tool bars with 10 or 20 or 50 very small, unlabeled icons. This is very intimidating to new users, and even power users have trouble remembering what the icons represent. It would be better to offer an option of

having a permanent text label appear under each icon—especially for user who do not have eidetic memories. Another problem with palettes and toll bars is that they can cover much of the screen real estate, so users cannot see as much of their data.

## 9.10 - Printing

If your application supports, printing, then you should have the following two items on the File menu:

- Page Setup
- Print

See [Technote 1092](#): *A Print Loop That Cares...* for complete information on adding a print loop to your application.

This is similar to what you might see in a Windows application, except that MFC provides support for the Print Preview feature. Unfortunately, print preview is not common in Macintosh applications. Should you provide this feature in your Mac program? I hope you do, since it is very valuable, and users will love you for it—even if they had not expected it to be there.

[Back to top](#)

---

[Previous Chapter](#)

[Table of Contents](#)

[Next Chapter](#)

[Further References](#)

---

The Apple Store	Hot News	Products	Design & Publishing	Developer	
	About Apple	Support	Education	Where to Buy	

[Search Tips](#) | [Site Map](#) | [Extended](#) | [Index](#)

[The Apple Store](#) | [Hot News](#) | [About Apple](#) | [Products](#) | [Support](#)  
[Design & Publishing](#) | [Education](#) | [Developer](#) | [Where to Buy](#) | [Home](#)

[Contact Us](#) - [Developer Site Map](#)

Copyright © 2000 Apple Computer, Inc. [All rights reserved.](#)

# An Introduction to Macintosh Programming for Windows Programmers

---

[Previous Chapter](#)
[Table of Contents](#)
[Next Chapter](#)

---

## Chapter 10—Resources

---

A Macintosh resource is a block of data in some defined format that is stored in the resource fork of a file. A resource is identified by its four-character type (e.g., 'WIND', 'MENU', 'STR#', 'A56x') and a 16-bit integer ID. In a given file, resources of the same type must have unique IDs.

The Mac OS APIs define a large number of standard resources, including both their binary data format and their four-character type. For example, the format of a compiled 'DLOG' resource used to specify a modal or modeless dialog window is shown in Figure 10.1. If your application were to need a dialog window to appear for some reason, you would need to create a 'DLOG' resource that describes the window size and location, window type, title, and so forth. At runtime, you would use the functions defined in the Dialog Manager part of the Toolbox to create a dialog based on the data in the resource.

**Figure 10.1 - Compiled structure of a 'DLOG' resource**



'DLOG' resource type	Bytes
Rectangle	9
Windowdefinition ID	2
Visibility	1
Reserved	1
Close box specification	1
Reserved	1
Reference constant	4
Item list ID	2
Window title	1 to 255
Alignment byte	0 or 1
Dialog box position	2

## 10.1 - Creating Resources

There are two basic ways to create resources:

- You can create a text file description of the resource, and then compile that with CodeWarrior's Rez resource compiler. The resulting compiled resource would be stored in a resource file, customarily created with a ".rsrc" suffix. The source text file is stored with a ".r" suffix. These file suffixes do not mean anything to the operating system. They are merely used to remind the programmer that they are either text or binary resource files.
- You can use a visual editor to design the resource. In this case, you would use a combination of mouse drawing actions plus a "property editor" to specify the design of each resource. Visual resource editors are discussed in [Chapter 3.4](#).

In addition to the standard resource types defined in the Mac toolbox, you can also define your own custom resource types. This will not be necessary in most cases, since all of the standard user interface items already have resource types defined for them.

## 10.2 - Visual Resources

The PowerPlant framework uses special resources of type 'PPob' ("PowerPlant object") to define the visual layout of windows and their contents. This could be done using standard Mac OS resource types, and the PowerPlant framework provides additional functionality for users of their 'PPob' resources. We strongly recommend you design your window contents using 'PPob' resources when using PowerPlant.

[Back to top](#)

The Apple Store	Hot News	Products	Design & Publishing	Developer	
	About Apple	Support	Education	Where to Buy	

[Search Tips](#) | [Site Map](#) [Extended](#) [Index](#)

[The Apple Store](#) | [Hot News](#) | [About Apple](#) | [Products](#) | [Support](#)  
[Design & Publishing](#) | [Education](#) | [Developer](#) | [Where to Buy](#) | [Home](#)

[Contact Us](#) - [Developer Site Map](#)

Copyright © 2000 Apple Computer, Inc. [All rights reserved.](#)

# An Introduction to Macintosh Programming for Windows Programmers

---

[Previous Chapter](#)[Table of Contents](#)[Next Chapter](#)

---

## Chapter 11—Summary

---

This document has introduced some of the issues involved with porting a Windows program to Macintosh. We have not attempted to delve into the details of the necessary C++ code changes, but have tried to describe the big picture, so that you can see the overall scope of the effort. Here are some of the areas that you will have to address when developing your application for the Macintosh.

### 11.1 - Things To Do

- Join the [Apple Developer Connection \(ADC\)](#). Joining ADC will allow you to get the technical support and information you need to efficiently finish your project.
- Get a Power Macintosh for development. You probably need at least 64 MB RAM, 6 GB hard disk, 10/100 Ethernet, 56K modem, etc. Any of Apple's tower models have sufficient horsepower for serious program development. You may want to get a model with DVD support, since programmer tools may come on DVD-ROM in the future.
- Buy an extra graphics card, and get two monitors. (These Macs can drive standard, multi-sync PC monitors, so you may not have to buy new ones.) One can be as small as 15". You can use that for running the application. The other one should be at least 17", so you can see enough source code and debugging information to be

productive. Tell your boss you need at least one 21” monitor to be really productive. (Two monitors will make you more productive. They also provide a good way to test your application behavior in a multiple-monitor setup.)

- Install the free assembly-language MacsBug debugger on your development machine. This allows you to trap serious errors, display processor stack crawls, exit from crashed programs, inspect blocks of memory on the heap, and other good stuff.
- Get CodeWarrior Pro from Metrowerks for your IDE.
- Get other tools as necessary, such as Resorcerer for resource editing, Object Master for browsing, and QC and SpotLight for memory management.
- Build a CodeWarrior project for each application, and import all your existing text files into it.
- Port the problem-domain modeling code—few changes should be necessary.
- Redesign the user interface to follow the Mac Human Interface Guidelines.
- Convert user interface resources to Macintosh format.
- Move GUI code from MFC to Metrowerks PowerPlant.
- Modify the code for handling memory management, threads, etc. as necessary.
- Convert 2D graphics calls to use QuickDraw.
- Convert 3D graphics calls to QuickDraw 3D or OpenGL for Macintosh.
- Convert Windows Help files to HTML Help files.
- Convert tool tips to Balloon Help.
- Design new application and document icons.
- Test the program with hard-core Mac users—only they will know if it feels right.

## 11.2 - Things to Remember

The Macintosh platform has some unique strengths. Leveraging these strengths can help you make the Macintosh version of your product better than it was before. These strengths include:

- The Apple Event Object Model, and the ability to make an application scriptable and recordable using AppleScript. AppleScript makes it easy for users to integrate multiple applications from many vendors, thereby automating complex processes.
- The ability to seamlessly display information across multiple monitors.
- Very powerful graphics and multimedia support using the QuickTime 3 technologies.
- A relatively seamless user experience, where plug-and-play really works and most users do not ever have to know about operating systems or device drivers.
- No worries about Y2K.
- The most loyal user base in the known universe.

The Apple Store	Hot News	Products	Design & Publishing	Developer	
	About Apple	Support	Education	Where to Buy	

[Search Tips](#) | [Site Map](#) [Extended](#) [Index](#)

[The Apple Store](#) | [Hot News](#) | [About Apple](#) | [Products](#) | [Support](#)  
[Design & Publishing](#) | [Education](#) | [Developer](#) | [Where to Buy](#) | [Home](#)

[Contact Us](#) - [Developer Site Map](#)

Copyright © 2000 Apple Computer, Inc. [All rights reserved.](#)



Apple Developer Connection

Technical: Mac OS

[Membership](#)

[Technical](#)

[Business](#)

[Site Map](#)

[Log In](#)

# An Introduction to Macintosh Programming for Windows Programmers

## Chapter 12—Some Sample Programs

---

### 12.1 - Overview

This Chapter describes a series of six C++ sample programs used to illustrate how to write a simple Macintosh application. The samples use the Metrowerks *PowerPlant* applications framework, which ships as part of the CodeWarrior Integrated Development environment. (*PowerPlant* ships with a wide range of useful sample programs. We'll describe many of those in more detail below.) These samples illustrate many common applications features, including:

1. Multiple document windows.
2. Disk files for persistent data.
3. Menu items for modifying the data.
4. Buttons for modifying the data.
5. Modal dialogs.
6. Printing.
7. Custom application and document icons.
8. Customized "About this program" dialog.

Each sample is based on a simple problem-domain class called **CCounter**, which keeps a 32-bit integer value for the count. The simplest version (#1) programmatically modifies the count, and displays the results in a console window. More elaborate samples use push

buttons, menu items, and modal dialogs to allow the user to modify the count, and use document windows to display the current count. The Print version (#5) is shown in the screen shot in Figure 12.1. This sample is described in more detail in Chapter 12.2, Anatomy of a Program.

The basic features supported in each sample program are summarized in Table 12.1.

Figure 12.1 - The Counter Sample, Version 5

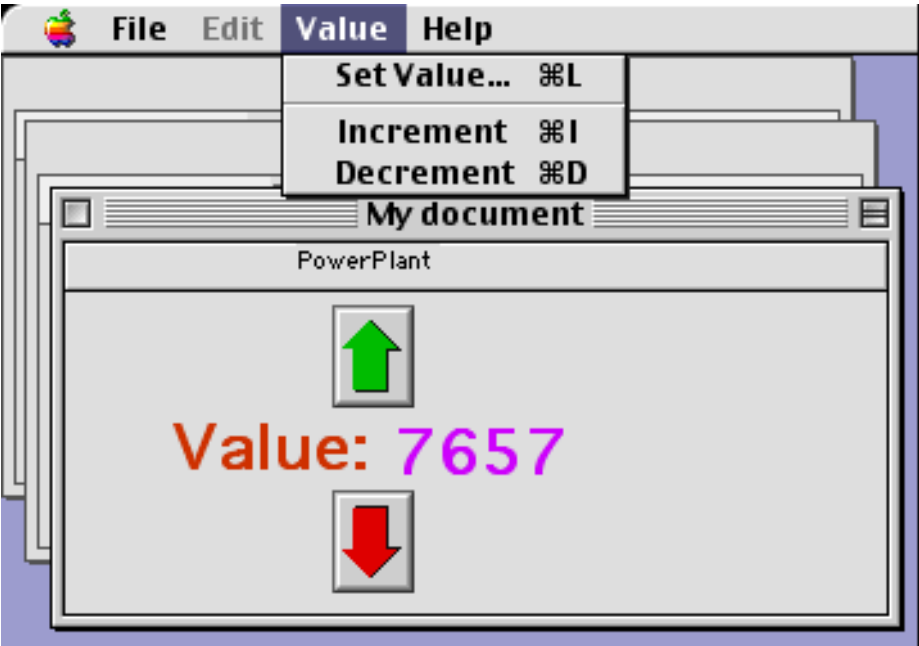


Table 12.1. Features of Various Samples

Feature	1. Console	2. Window	3. Dialog	4. Docs	5. Print	6. About
Counter model	XX	XX	XX	XX	XX	XX
Console window	XX	XX	XX	XX	XX	XX
Normal window		XX	XX	XX	XX	XX
Push buttons		XX	XX	XX	XX	XX
Menus			XX	XX	XX	XX
Modal dialog			XX	XX	XX	XX
Documents & Files				XX	XX	XX
Printing					XX	XX
Finder icons					XX	XX
Custom About& box						XX

You can use these samples in two ways: (1) as a kind of mini-tutorial, and (2) as a place from which to steal code.

To learn about writing a program using *PowerPlant*, you can look at the code in the simplest version (number 1) to see how to write a very basic program that has only one



simple C++ class. You can look at each subsequent version to see how the program evolves as features are added. As a source for techniques and code fragments, you can use Table 12.1 to see which samples support a feature of interest, and then go to that sample to see how the resources and C++ code are structured to support that feature.

*PowerPlant* ships with a wide range of useful sample programs. We'll describe many of those in more detail in an upcoming document.

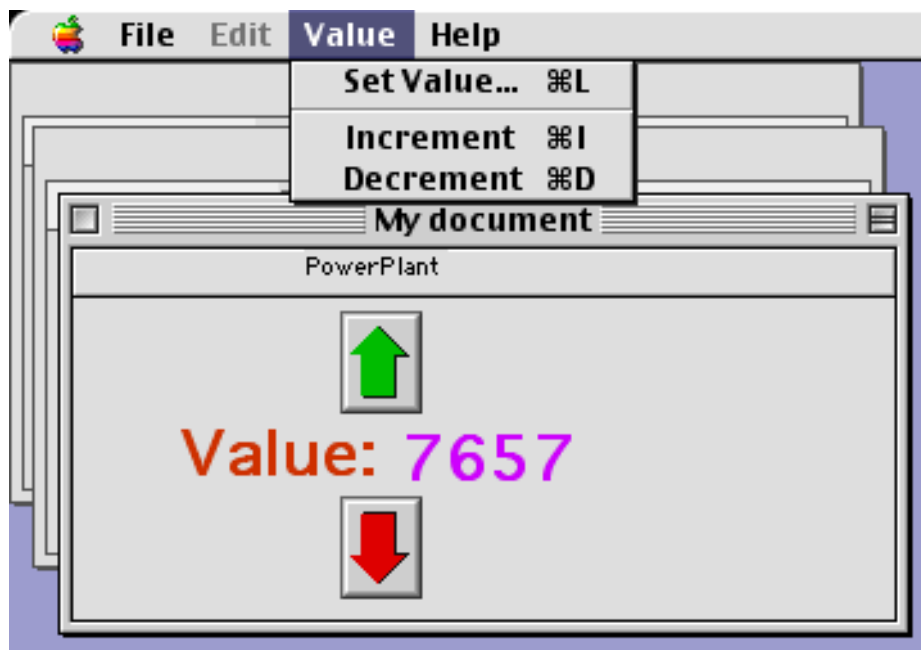
## 12.2 - Anatomy of a Macintosh Application

### Introduction

This Chapter gives an overview of the structure of a small but functional Macintosh sample application. Before reading this, you should read the separate Chapters entitled “[An Introduction to Macintosh Programming for Windows Programmers](#)”. The first Chapter introduces some of the issues that you must deal with when porting a Windows application to Macintosh, while the second provides an overview of a whole family of sample programs.

The Counter Sample Version 5 is a multiple-document application with the user interface of a typical (but very simple) productivity application. A screen shot is shown in Figure 12.2.

**Figure 12.2 - Counter screen shot**



Although this program consists of less than 250 lines of C++ code, its features include:

- Multiple document windows.
- Disk files for persistent data.
- Menu items for modifying the data.
- Buttons for modifying the data.
- Modal dialogs.

- Printing.
- Bundle resource for custom application and document icons.

We'll describe this sample from a number of perspectives, including:

1. What the user sees - that is, the Graphical User Interface (GUI).
2. Some of the resources used to build the GUI.
3. The C++ classes used to build the application.


## User View (GUI)

### Finder Icons

When this application is installed, the user sees a custom application icon. (This application consists of a single, self-contained, file less than 1 MB in size. You would not need a fancy installer for this app. Just drag the application from a floppy disk to the hard disk and it is ready to go. See [Altura Software](#) for more information.) Each document saved by the user is seen as a custom document icon. These desktop icons are shown in Figure 12.3. These icons were designed by the programmer (me), so they are kind of ugly. They are included in the sample, however, so a programmer can see how to manage Finder icons.

**Figure 12.3 - Finder icons for application and documents**

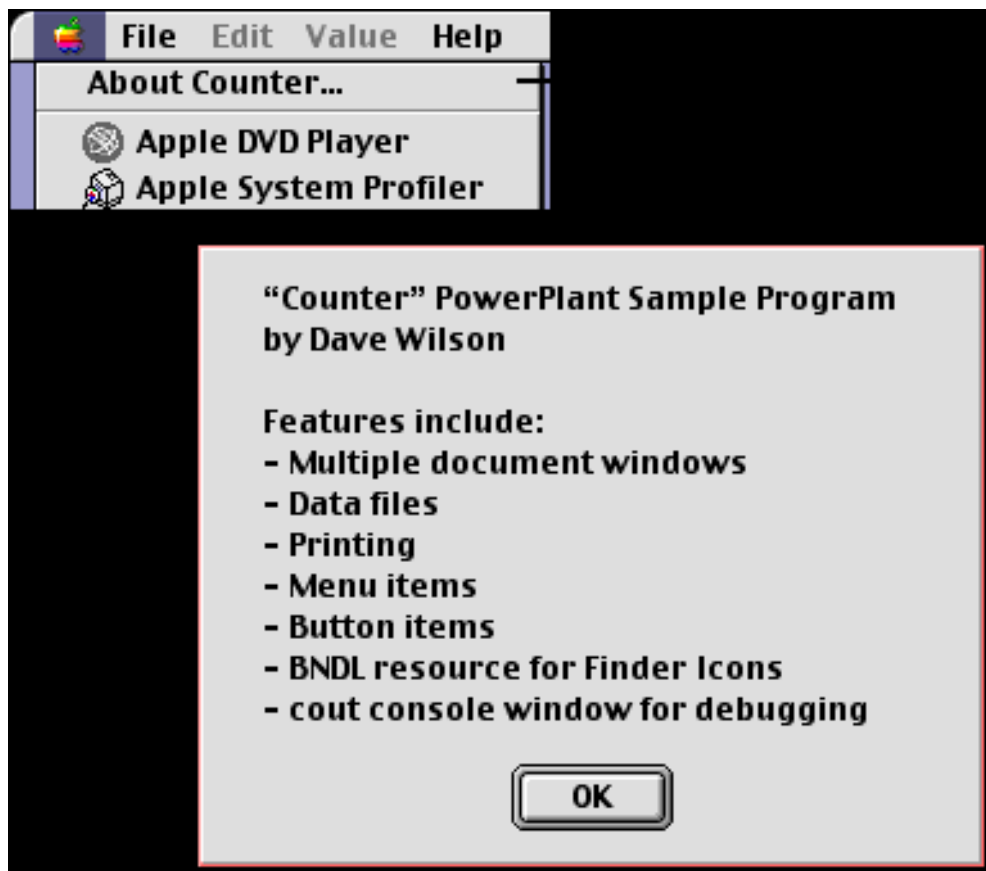


The program has the standard **Apple** , **File**, and **Edit** menus, plus a **Value** menu unique to this application, as described in the next Chapters.

### Apple Menu and About Box

The Apple menu, as usual, has the first item bring up an Alert describing the program, as shown in Figure 12.4. An Alert is a simple kind of modal dialog that usually has text, an optional icon, and OK and Cancel buttons. Note that any menu item that leads to a window requiring user action should end with the ellipsis character (&), typed using Option-semicolon.

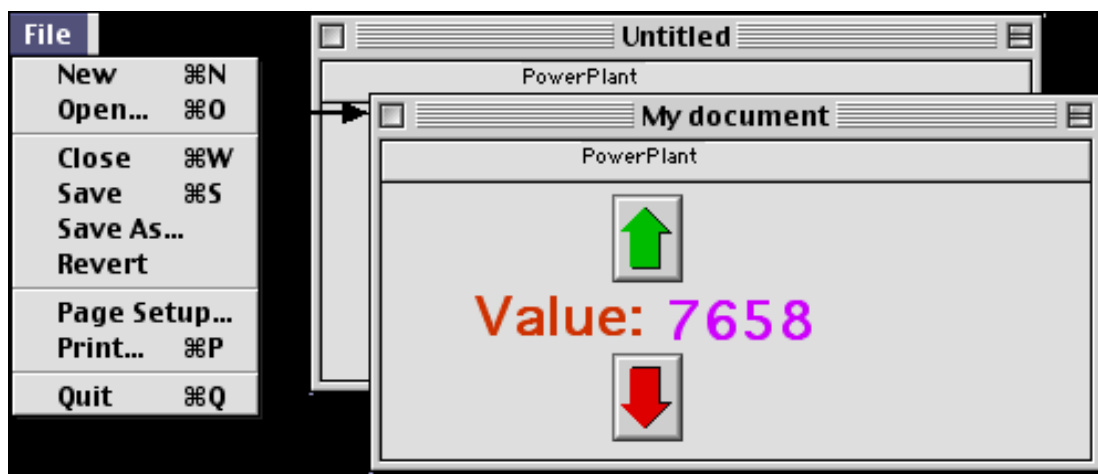
**Figure 12.4 - The Apple Menu and About& Dialog**



## The File Menu and Document Windows

The File menu is quite standard, as shown in Figure 12.5. New opens a new "Untitled" document, while Open& opens an existing document, and so forth. Notice the standard command-key equivalents. These should always be used with these particular menu items.

**Figure 12.5 - The File Menu and Document Windows**



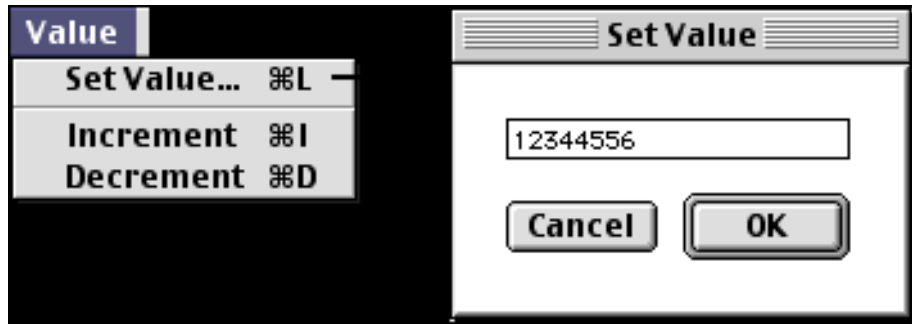
## The Edit Menu

This program does not support Undo or the Clipboard, so the Edit menu items are always disabled.

## The Value Menu and Modal Dialog

The Value menu is specific to this Counter program, as shown in Figure 12.6. These menu items are managed by the active document object. If no documents are open, then the menu items will be disabled. Note that the first menu item leads to a movable modal dialog, as indicated by the dialog's special window style. This type of dialog is modal inside the application, so the user must put it away before working with the rest of the application. The user can drag this dialog around on the screen, or even switch to any other application while this dialog is being displayed. Movable modal dialogs are therefore more user friendly than the older style strictly modal dialogs.

**Figure 12.6 - Program-Specific Menu and Set Value Modal Dialog**



## Resources

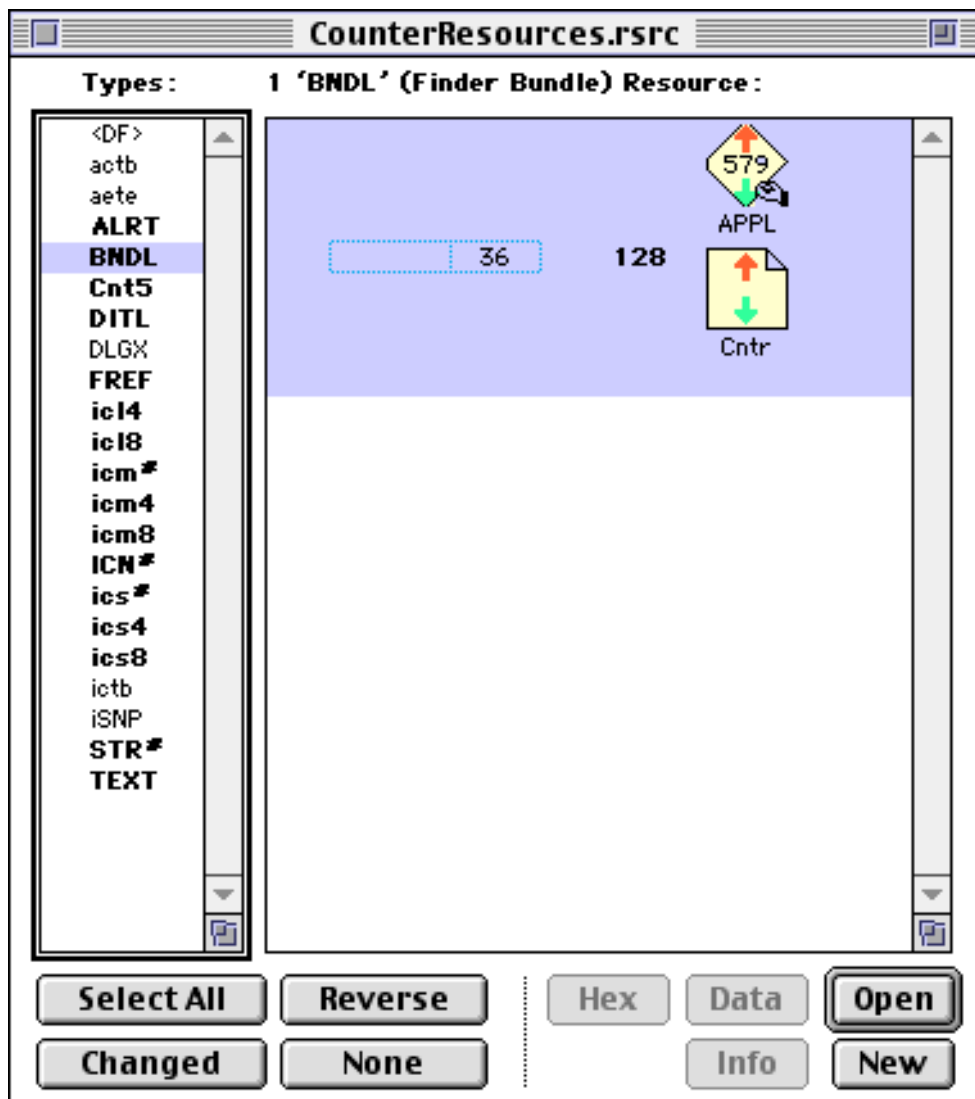
The Counter sample was based on two different sets of resources, created with two different resource editors. The Resorcerer editor was used to create various non-visual resources, plus a few specialized visual resources. Metrowerks Constructor was used to create most of the visual resources that appear on the screen, including menus, windows, dialogs, and so forth. We'll describe the major ones below.

### Resorcerer Non-Visual Resources (“CounterResources.rsrc”)

The most interesting resources are the ones you'll create using Constructor to define the windows and their contents. There are other, mostly non-visual, resources that you must create that are less exciting but still necessary. You can use Apple's free ResEdit tool to create these, but most Mac programmers use the non-free Resorcerer editor, as shown in Figure 12.7.

Resources often needed include the ones listed in Table 12.2. This list may look quite intimidating, but most of them are provided for you by template projects included with the *PowerPlant* framework. In this Counter example, the only resources in the list that I had to create were a 'TEXT' resource used in the About box, and the 'BNDL' resource and associated resources used for the Finder icons.

**Figure 12.7 - Resorcerer mostly non-visual resources**



**Table 12.2 - Resources created in Resorcerer**

Type	Name	Comments
actb	Alert Color Table	For non-standard alert colors.
aete	Apple Event	Part of required Apple Event support
ALRT	Alert window	Window description for an alert.
BNDL	Bundle	Collect resources needed for Finder icons
Cnt5	Creator (Signature)	Unique resource type for each application
DITL	Dialog Item List	Items that go in an alert or dialog window.
DLGX		Dialog extensions
FREF	File reference	For application and document types
icl4		Finder icons: large 4-bit/pixel
icl8		Finder icons: large 8-bit/pixel
icm#	Icon list	Finder icons: mini B#38;W
icm8		Finder icons: mini 8-bit/pixel
ICN#	Icon List	Finder icons: large B&W icon and mask

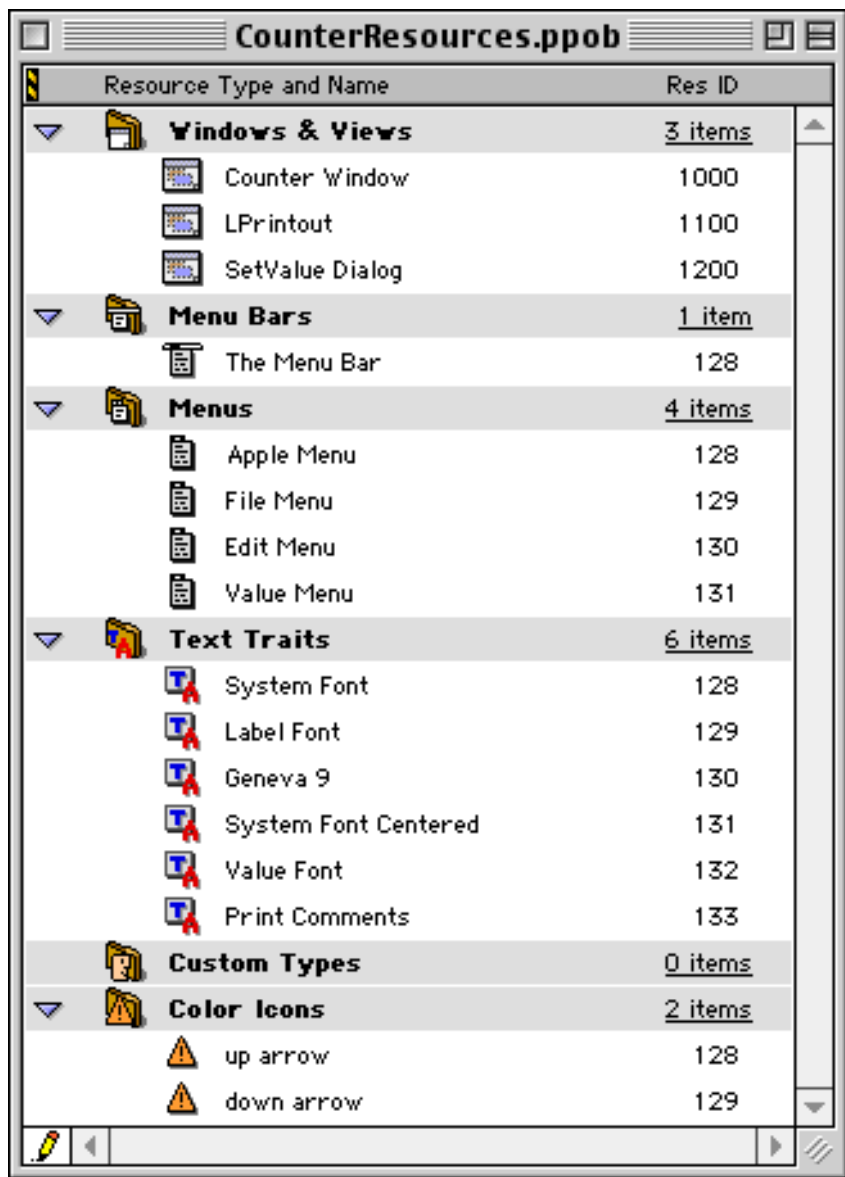
ics#		Finder icons: small B&W icon and mask
ics4		Finder icons: small 4-bit/pixel
ics8		Finder icons: small 8-bit/pixel
iSNP	Snapshot	Finder icon snapshot
STR#	String list	List of single-line strings
TEXT	Text	Paragraph of text.

## Constructor Visual Resources (“CounterResources.ppob”)

Figure 12.8 summarizes the visual resources created using Constructor. The most interesting are the Windows and Views shown at the top of the figure. These include:

- **Counter Window PPob #1000**, used to create each document window, as shown in Figure 12.9.
- **LPrintout PPob #1100**, used to define the printing layout, as shown in Figure 12.10.
- **SetValue PPob #1200**, used to create the modal dialog, as shown in Figure 12.11.

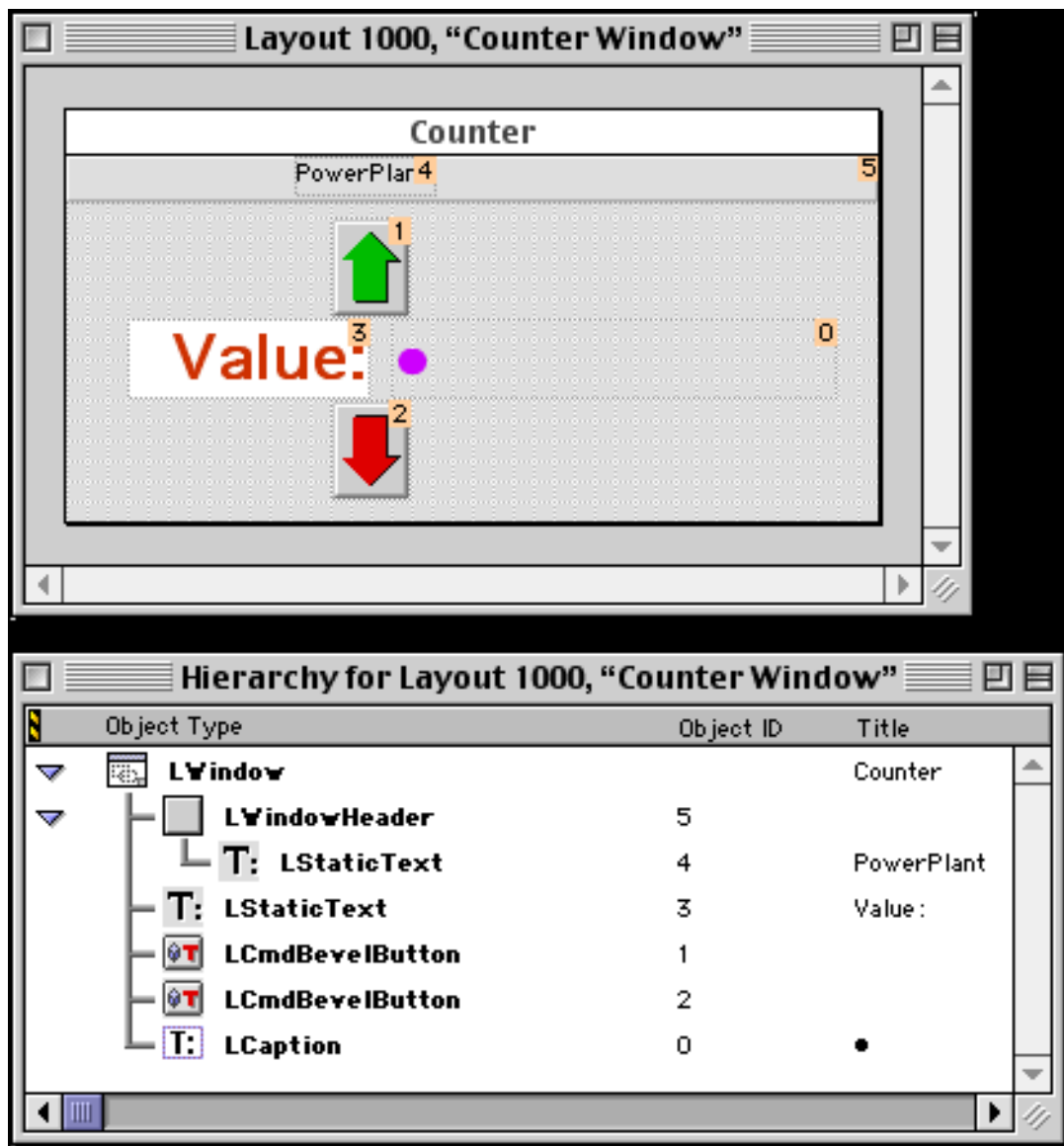
### Figure 12.8 - Constructor's '.ppob' resources



As shown in Figure 12.9, each document window is described by a window resource containing:

- a window header, which does nothing useful in this program, but could be used as a tool bar full of small icons.
- static text, used to display the “Value” label.
- a caption, which is another type of static text used to display the current value of the counter.
- two command buttons, used to increment and decrement the counter.

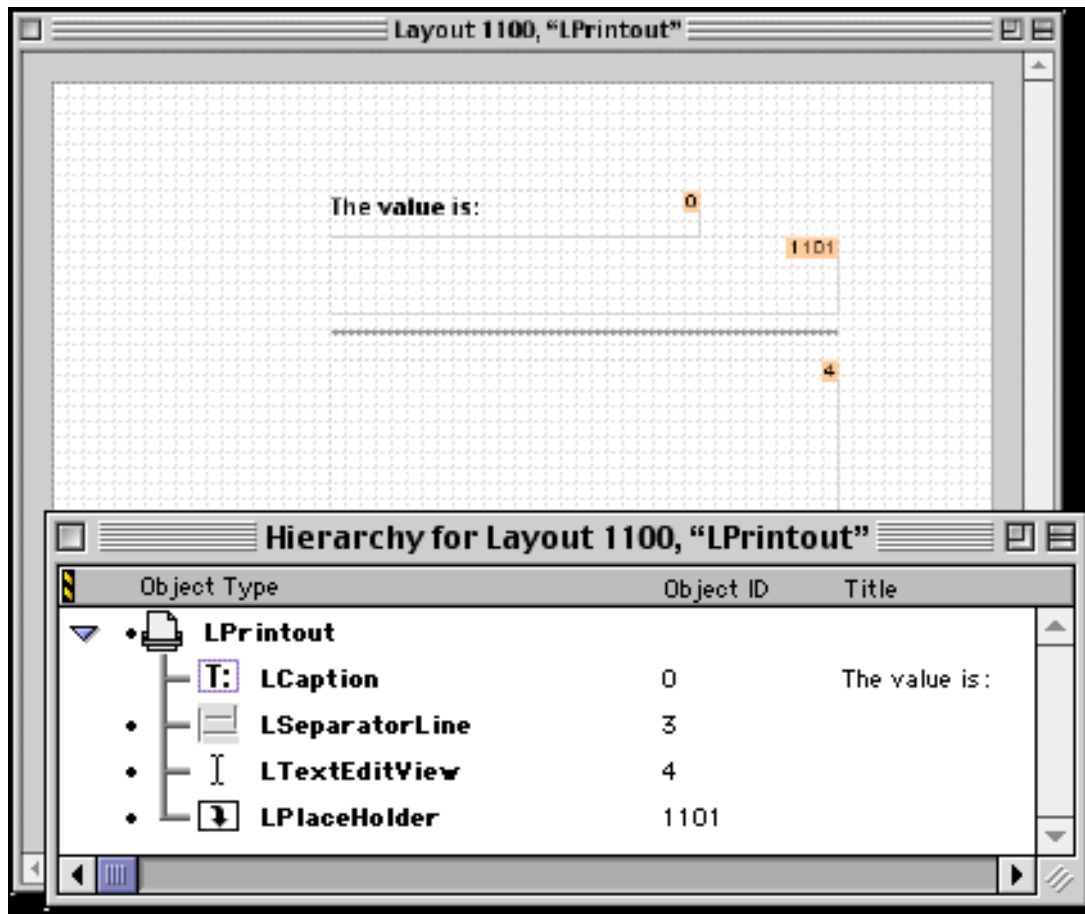
**Figure 12.9 - View hierarchy for Counter Window resource**



The print layout shown in Figure 12.10 is used only when printing. This separation between the screen layout and the printing layout is especially valuable in this program, since you would not want to print the command buttons as they appear on the screen. The lower `LTextView` was set in Constructor to display the text specified in a 'TEXT' resource of ID 1000. This separate 'TEXT' resource was created in the "CounterResources.rsrc" file described above.

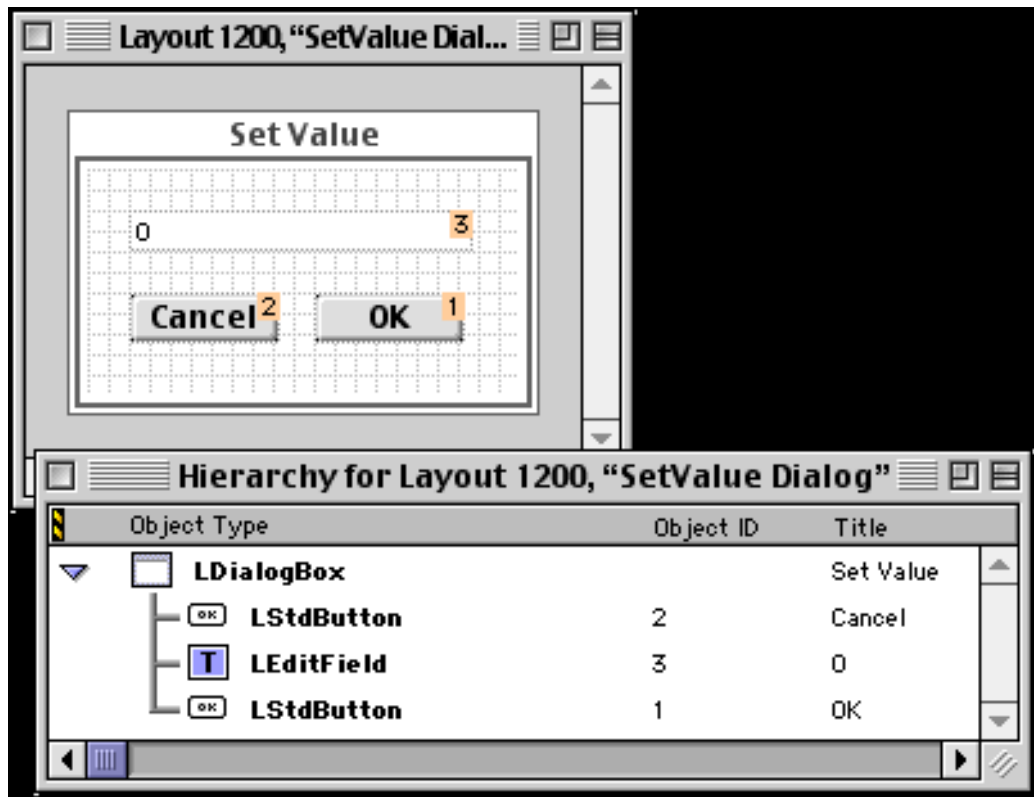
**Figure 12.10 - Layout used for printing**





The Set Value dialog shown in Figure 12.11 is based on defining a modal dialog window, and then placing text and buttons inside. Each button is assigned an integer “value message.” When a button is clicked, this message value is processed by the program code. In the case of a modal dialog, pressing either button closes the window, but your code will modify the counter's value only if the OK button has been clicked.

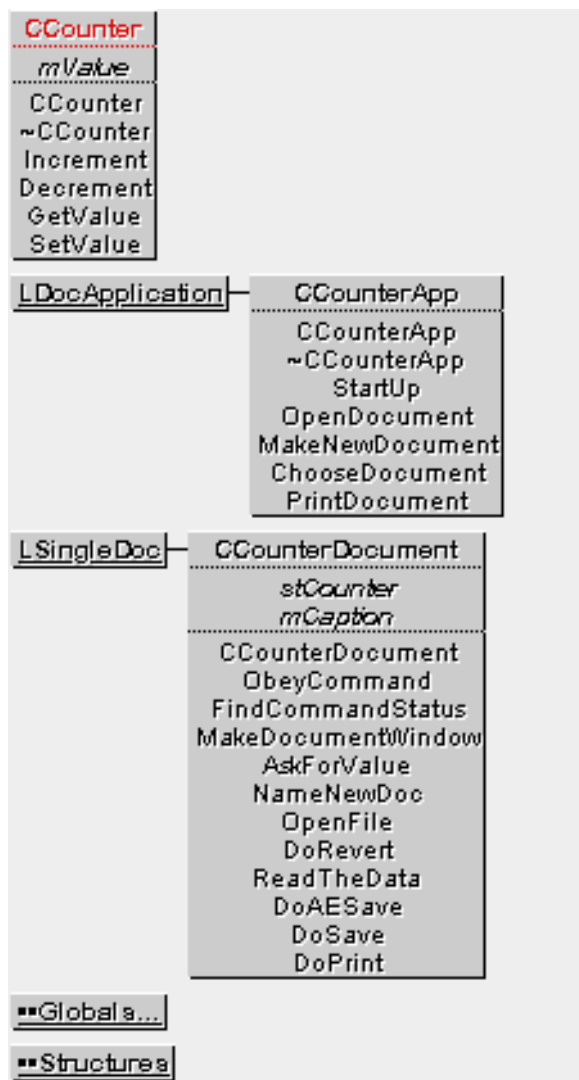
**Figure 12.11 - Resource visual hierarchy for Set Value dialog**



## C++ Classes

Although Counter is a complete Macintosh application, the code is quite small. The program consists of only three new C++ classes, plus the hundreds of classes in the Metrowerks *PowerPlant* framework (PPFW) on which this sample was based. The class diagram is shown in Figure 12.12. This diagram was produced automatically by the cross-platform *Object Master* editor/browser, sold by [Altura Software](http://www.altura.com)). This very handy tool can parse C++ or Java source code into these compact diagrams.

**Figure 12.12 - C++ Class Diagram for the Counter Sample**



## Model Class

The CCounter class represents a problem-domain specific “model” class. In this example, each counter object holds onto a 32-bit integer data member (mValue) as its current value. This class is not derived from any other class. There is no common base class in the *PowerPlant* framework, so you can start a new class tree anywhere. The code for the CCounter class is shown in Listing 12.1 and Listing 12.2.

### Listing 12.1 - CCounter Header File

```

1.      #pragma once
2.
3.      #include          // Int32
4.
5.      // =====
6.      class CCounter {
7.      public:
8.
9.          virtual      CCounter();
10.         virtual      ~CCounter();
11.         virtual void   Increment();
12.         virtual void   Decrement();
13.         virtual Int32  GetValue();
14.         virtual void   SetValue(Int32 value);
15.     private:
16.         Int32          mValue;
17.     };

```

## Listing 12.2 - CCounter Implementation

```

1.      #include "CCounter.h"
2.
3.      //=====
4.
5.      CCounter::CCounter()
6.      {
7.          mValue = 0;
8.      }
9.
10.     //-----
11.     CCounter::~~CCounter()
12.     {
13.     }
14.
15.     //-----
16.     void
17.     CCounter::Increment()
18.     {
19.         mValue++;
20.     }
21.
22.     //-----
23.     void
24.     CCounter::Decrement()

```

```

25.     {
26.         mValue--;
27.     }
28.
29.     //-----
30.     Int32
31.     CCounter::GetValue()
32.     {
33.         return mValue;
34.     }
35.
36.     //-----
37.     void
38.     CCounter::SetValue(Int32 value)
39.     {
40.         mValue = value;
41.     }

```

## PowerPlant Style Guidelines

Note the *PowerPlant* style convention that your class names begin with uppercase “C”, followed by another upper case letter, while data member names begin with lowercase “m”, followed by an uppercase letter. Class names in the *PowerPlant* framework usually begin with upper case “L”. Classes intended to be stack-based rather than heap-based have class names that begin with “St”.

## Application Class

The CCounterApp class is derived from *PowerPlant*'s LDocApplication class. There is a single instance of the application class made in CCounterApp::main() when the application is launched by the user. The application object manages menu commands that apply to the application as a whole, such as New, Open, and Quit. These items will be enabled even when there are no open document windows.

## Document Class

The CCounterDocument class is derived from *PowerPlant*'s LSingleDoc class. LSingleDoc is a misleading name, since you can use it in multiple-document applications such as this one. The word “Single” refers to the fact that there is a single window associated with each document. Each instance of the document holds an embedded instance of a CCounter object, stCounter, to manage the value. Each document object manages its associated window, manages reading and writing data to disk files, and is responsible for printing the window contents.

## Member Functions

Interesting member functions for the three classes are described in Table 12.3.

**Table 12.3 - C++ Classes and Member Functions**

<b>Class</b>	<b>Functions</b>	<b>Comments</b>
CCounter	Increment()	Increase 32-bit integer by 1.
	Decrement()	Decrease value by 1.
	GetValue()	Return current value.
	SetValue()	Set new value.
CCounterApp	main()	Called when application is launched.
	StartUp()	Called once by PPFW when program is launched.
	OpenDocument()	Called by PPFW after user specifies file to open. Creates new document object.
	MakeNewDcoument()	Called by PPFW when user chooses New menu item. Creates document object.
	ChooseDocument()	Called by PPFW when user chooses Open& menu item. Displays Get File dialog, so user can choose file to open.
	PrintDocument()	Called by PPFW when user chooses Print& menu item.
CCounterDocument	FindCommandStatus()	Repeatedly called by PPFW. Enables menus or buttons as indicated by current document state.
	ObeyCommand()	Called by PPFW when user chooses a menu item, or when a command button is pressed. Should carry out requested action.
	MakeDocumentWindow()	Factory method to create a new document window from the specified PPob resource.
	AskForValue()	Called from ObeyCommand(). Displays modal dialog so user can specify new counter value.
	NameNewDoc()	Called by PPFW. Sets new document name, such as "Untitled-3".
	OpenFile()	Called by PPFW. Open disk file and reads in data.
	DoRevert()	Called by PPFW when user chooses Revert&. Resets document data to previously saved version.
	ReadTheData()	Called by OpenFile() and DoRevert().

DoAESave()	Called by PPFW when user chooses Save menu item.
DoSave()	Called by DoAESave(). Writes data to a disk file.
DoPrint()	Called by CCounterApp::PrintDocument(). Renders print layout view to printer.

## Summary

A popular process for writing a Macintosh application follows the organization of this document. First, you design the user interface. Then, you use resource editors to create resources for the visual and non-visual parts of the program. Finally, you write the code to make everything work.

When using the *PowerPlant* framework, your C++ coding is usually split into two parts. First, you create problem-domain specific model classes such as CCounter. Then you subclass standard *PowerPlant* classes like LDocApplication and LSingleDoc, overriding the standard methods described above. Because *PowerPlant* does most of the work, you can often add major pieces of end-user functionality by writing a few extra lines of code. In fact, the C++ code base for this Counter application comes to slightly less than 250 lines of code, not counting blank lines or comments. *PowerPlant* does all the rest.

What could be easier?

[Back to top](#)

---

[Previous  
Chapter](#)

[Table of  
Contents](#)

[Download Win2MacCounter  
Sample Code](#)

[Further References](#)

---

The Apple Store	Hot News	Products	Design & Publishing	Developer	
	About Apple	Support	Education	Where to Buy	

[Search Tips](#) | [Site Map](#) [Extended](#) [Index](#)

[The Apple Store](#) | [Hot News](#) | [About Apple](#) | [Products](#) | [Support](#)  
[Design & Publishing](#) | [Education](#) | [Developer](#) | [Where to Buy](#) | [Home](#)

[Contact Us](#) - [Developer Site Map](#)

Copyright © 2000 Apple Computer, Inc. [All rights reserved.](#)



# An Introduction to Macintosh Programming for Windows Programmers

---

[Table of Contents](#)

---

## Further References

---

### Apple resources

- [Apple and the Year 2000](#)
- [Apple Developer Connection Programs](#)
- [Apple Developer Documentation](#)
- [Apple Human Interface Guidelines](#)

### Third-party resources

- [Altura Software, Inc.](#)
- [Bare Bones Software, Inc.](#)
- [CWASTEEdit](#)
- [Electric Fish, Inc.](#)
- [Mathemaesthetics Home Page](#)
- [Memory Madness \(Stairways Software\)](#)

- [Metrowerks Corporation](#)
- [Onyx Technology, Inc.](#)
- [Perforce Software](#)
- [Synergex Home Page](#)
- [Uni Software Plus](#)

---

## [Table of Contents](#)

---

The Apple Store	Hot News	Products	Design & Publishing	Developer	
	About Apple	Support	Education	Where to Buy	

[Search Tips](#) | [Site Map](#)   [Extended](#)   [Index](#)

[The Apple Store](#) | [Hot News](#) | [About Apple](#) | [Products](#) | [Support](#)  
[Design & Publishing](#) | [Education](#) | [Developer](#) | [Where to Buy](#) | [Home](#)

[Contact Us](#) - [Developer Site Map](#)

Copyright © 2000 Apple Computer, Inc. [All rights reserved.](#)