Mac OS 8 Compatible Application

This paper describes a Mac OS 8 Compatible Application.

Note: This is an evolving document which will be revised based on developer feedback.

1.0 Definition

A Mac OS 8 Compatible Application is an application that can run well on both System 7.x and Mac OS 8. It only uses features documented in Inside Macintosh. It is possible to write a Mac OS 8 Compatible Application on System 7.x today — and many System 7.x Applications are Mac OS 8 Compatible today!

1.1 Summary of Guidelines

In order to be classified as a Maxwell Compatible Application, the following guidelines apply:

Unsupported in Mac OS 8:

1. Don't use ASLM
2. Don't access the trap table directly
3. Don't use jGNEFilter to intercepts events globally.
4. Don't call PPostEvent
5. Don't call GetEvQHdr
6. Don't access private traps or private LowMem
7. Don't use PrivateInterfaceLib
8. Don't rely on FileSystem hooks or patches
9. Don't use Compressed Resources

10. Don’t call InitResources, RsrcZoneInit, RsrcMapEntry
11. Don’t use resource file refNums with File Mgr calls
12. Don’t call SetApplLimit
13. Don’t call Virtual Memory calls
14. Don’t assume relationships between the application stack and heap
15. Don’t allocate all of Temporary Memory
16. Don’t allocate all hard drive space
17. Don’t share data structures between applications
18. Don’t hard-code Font usage
19. Don’t write to your applications data-fork
20. Don’t access hardware directly
21. Don’t assume the system state in Notification Mgr routines
22. Don’t change the Window List directly
23. Don’t set the Global-Share bit in a CFM Library that contains code
24. Don’t rely on the structure of system memory
25. Don’t require AOCE interfaces.
26. Don’t require an INIT, Control Panel or Desk Accessory
27. Don’t use the Dictionary Manager
28. Don't depend on Script Manager internals
29. Don’t poll for Open Transport asynchronous events
30. Don't pass open file reference numbers between processes
31. Don't assume all FCB and VCB fields are valid
32. Don't modify FCB or VCB data structures.
33. Don't call GetDrvQHdr or AddDrive

Discouraged in Mac OS 8:

1. Don’t draw directly to the screen
2. Don’t patch within an application
3. Don’t access memory in other applications
4. Don’t access low memory directly
5. Don’t use a custom MBDF
6. Don’t patch toolbox definition procedures(WDEF, MDEF, etc.)
7. Don’t rely on pairing of suspend & resume events
8. Don’t assume that current process and front process are the same when launching

9. Don’t use TopMapHndl and SysMapHndl lowmems
10. Don’t use vCheckLoad
11. Don’t use ROMBase to test your environment
12. Don't use obsolete names for Text Utilities
13. Don't use negative selectors for GetScriptVariable()
14. Don't assume the system script is the same in all processes
15. Don't assume resource and file reference numbers are interchangeable
16. Don't use working directories
17. Avoid registering Gestalt selectors based on selector functions
18. Make no assumptions about resource ordering using index calls
19. Make no assumptions about editing behavior of resource file
20. Don’t rely on Memory Manager implementation

Well-behaved in Mac OS 8:

1. PowerPC native implementations
2. Use GX printing
3. Use OpenTransport networking
4. Use Lowmem accessors
5. Use the latest universal interfaces
6. Support only System 7 and later.
7. Minimize patching
8. Factor your application
9. Use standard definition procedures
10. Be VM friendly
11. Locate special folders using Folder Mgr
12. Specify stack size in code fragment resource
13. Be WorldScript aware
14. Be Inline-Input aware

1.2 Unsupported in Mac OS 8

This section details features that are not supported in Mac OS 8 and identifies alternative solutions that are supported.

1.2.1 Don’t use ASLM

The Apple Shared Library Manager is not available under Mac OS 8. Applications that rely on ASLM as a shared library mechanism or to maintain plug-ins will need to be redesigned. Mac OS 8 supports two shared library mechanisms: SOM for object-oriented libraries and the Code Fragment Manager for procedural libraries.

1.2.2 Don’t access the trap table directly

Don’t make assumptions about the implementation of the trap table. The location, size and nature of the trap table are all private. Execution of the entries in the trap table is only supported through A-Traps (68k system calls) or through `NGetTrapAddress`. Applications which circumvent the APIs will break under Mac OS 8. Modification of trap table entries is only supported via `NSetTrapAddress`.

1.2.3 Don’t use jGNEFilter to intercept events globally.

Mac OS 8 supports the jGNEFilter mechanism in a limited fashion. The jGNEFilter proc will be called for all events intended for the installing application only; events targeted at other applications will not be available to a jGNEFilter proc.

1.2.4 Don’t call PPostEvent

`PPostEvent` is used in System 7.x to post fake mouse and keyboard events. In Mac OS 8, `PPostEvent` does not allow events to be posted from one application to another. If you need to post application-defined events, use Apple Events to send a mouse or keyboard event.

1.2.5 Don’t call GetEvQHdr

`GetEvQHdr` points to a list which is always empty on Mac OS 8. Again, events are maintained on a per-application basis so most uses of this call are invalid on Mac OS 8.

1.2.6 Don’t access private traps or private LowMem

All private traps have been removed from the Mac OS 8 trap table. System software uses CFM binding to get to system services. Any applications that call private traps directly or indirectly (via `GetTrapAddress` & `CallUniversalProc`) will break under Mac OS 8. Specifically, native applications which have their own glue code to get to system calls unavailable on System 7.x will break.

1.2.7 Don’t use PrivateInterfaceLib

Under Mac OS 8, PrivateInterfaceLib no longer exists. Therefore, applications that link against or dynamically load PrivateInterfaceLib will not work.

1.2.8 Don’t rely on FileSystem hooks or patches

The System 7.x hooks: `ExtFSHook` and `FSQueueHook` are not supported under Mac OS 8. In addition, patching the file system will have the same limitations as `JGNEFilter` (see 1.2.3, above) -- the effect of the patching is per-application, not system-global.

1.2.9 Don’t use compressed resources

Compressed resources are not supported in Mac OS 8. Compressed resources are a system-private mechanism of System 7. Your application should not contain any compressed resources or be built with development tools which produce compressed resources.

1.2.10 Don’t call InitResources, RsrcZoneInit, RsrcMapEntry

Both InitResources and RsrcZoneInit are now obsolete. In the past, these functions where called by system software to setup the application resource environment on behalf of the application. They are no longer called by system software and are no longer supported.

In the past RsrcMapEntry was used to read resource manager data directly from the resource map. In order to use this call properly, the caller uses explicit knowledge of the layout of resource map entries within the resource map. In Mac OS 8, the in-memory resource map is not present and this call is not supported.

1.2.11 Don’t use resource file refNums with File Mgr calls

In the past, the resource manager used the file manager to open, read, write and close the resource fork. The refNum obtained from the file manager was used to represent the resource manager refNum for an open resource fork. This enabled the callers of the resource manager to use the refNum obtained from the resource manager to call the File Manager. In Mac OS 8, the resource manager no longer uses the file manager to open, read, write and close resource forks and thus, the refNum returned by the resource manager is in no way related to File Manager refNums. Callers of the Resource Manager can no longer call the File Manager using refNums obtained from the Resource Manager.

1.2.12 Don’t call SetApplLimit

In the past, a relationship existed between an application stack and the application heap. This relationship allowed for the shrinking or growing of the application stack by calling `SetApplLimit`. In Mac OS 8, the relationship between stack and application heap no longer exists. Using `SetApplLimit` to adjust an application's stack is no longer supported in Mac0S 8. Use the code fragment resource instead (see section 1.4.12).

1.2.13 Don’t call Virtual Memory calls

With the new MicroKernel architecture introduced with Mac OS 8, the System 7.X virtual memory calls are no longer supported.

1.2.14 Don’t assume any relationship between application stack and heap

See "Don’t call `SetApplLimit`", section 1.2.12.

1.2.15 Don’t allocate all of Temporary Memory

Mac OS 8 has a virtual memory subsystem which can allocate address space on demand. The amount of this address space is limited only by the amount of disk space available on the system. Allocate only the amount of memory that you can use, do not allocate all that is available.

1.2.16 Don’t allocate all hard drive space.

Since the virtual memory system uses the disks to "back" memory, allocating all hard disk space on the system will inevitably lead to running out of memory.

1.2.17 Don’t share data structures between applications

Although applications still run in the same address space, system services (like the Window Manager and Event Mgr) maintain structures on a per-application basis. For this reason, creating a data structure in one application (i.e. a menu) and attaching it to a data structure in another application (i.e. a menubar in another app) will not work under Mac OS 8.

1.2.18 Don’t hard-code Font usage

Mac OS 8 allows for customization of the appearance of the system. One of the settings that users are allowed to customize is the default system and application fonts. Applications should use the calls `GetSysFont` and `GetAppFont` instead of hard-coding (e.g. Chicago 12).

1.2.19 Don't write to your applications data-fork

Writing to your applications data-fork while the application is running will result in a file system permission error. Under Mac OS 8, all application data-forks will be file-mapped read-only with exclusive access. Applications should store personalized information (i.e. name, serial number, etc.) into a preference file.

1.2.20 Don't access hardware directly

Under Mac OS 8, hardware will not be directly accessible (i.e. it will not be mapped into the address space in which an application runs). This limitation greatly improves system stability. Similar functionality will be available by interactions with drivers and I/O calls.

1.2.21 Don't assume the system state in Notification Mgr routines

Under Mac OS 8, a notification manager completion routine will be called at different times than on System 7.x. Making assumptions around how and when the routine will get called (i.e. the frontmost app) will generally not be valid.

1.2.22 Don't change the Window List directly

Under Mac OS 8, the Window Manager maintains the system window list separately from the `nextWindow` field. Changing this field directly will not have the desired effect. Use the routines `BringToFront` and `SendBehind` instead.

1.2.23 Don't use Global-Share bit in a CFM Library containing code

Linking against per-context libraries in a globally-shared library that contains code will not work in a per-context fashion. If you need to maintain system-wide global data, you should use a separate library that only contains data and doesn't link against other libraries.

Note: The global share bit is accessible in Metrowerks CodeWarrior from the "share data section" checkbox in the PPC Pef preferences pane, and from the -s option in the MakePef MPW tool.

1.2.24 Don’t rely on the structure of system memory

In Mac OS 8, the structure or layout of system memory is not the same as in System 7.x. In general, addressing arithmetic based on System 7.x memory structures will not work as desired. The location (address) and relationship of application heaps, code, static data and stacks have changed. For example, do not assume that your native code is in your application heap.

1.2.25 Don’t require AOCE interfaces.

Not all currently available AOCE interfaces may be available on Mac OS 8. If your application uses AOCE, use Gestalt to check for the availability of AOCE features AND weak-link against the AOCE library.

1.2.26 Don’t require an INIT, Control Panel or Desk Accessory

INITs, Control Panels and Desk Accessories are not supported under Mac OS 8. If your application is packaged up as one of these, you must rewrite your application. If you rely on one of these for functionality, be prepared to fail gracefully when they are not present.

1.2.27 Don’t use the Dictionary Manager

The Dictionary Manager, described in *Inside Macintosh: Text chapter 8,* is not supported in Mac OS 8.

1.2.28 Don't depend on Script Manager internals

Some applications take advantage of knowledge of the script manager's internal data structures. These will be completely different in Mac OS 8. The private low-memory locations that provided access to these structures are unsupported in Mac OS 8.

1.2.29 Don’t poll for Open Transport asynchronous events

If using Open Transport and asynchronous events, endpoint providers are discouraged from polling for these events. Polling will have adverse effects under Mac OS 8. Use a notifier function in responding to asynchronous events instead.

1.2.30 Don't pass open file reference numbers between processes

Mac OS 8 maintains the System 7 compatibility FCB list on a per-process basis. This means that open file reference numbers have meaning only within the process which opened the file. Use file specifications or aliases instead.

1.2.31 Don't assume all FCB and VCB fields are valid

Only a subset of the FCB and VCB fields are valid in Mac OS 8. The following FCB fields are valid: `fcbFlNum, fcbFlags, fcbEOF, fcbPLen, fcbCrPs, fcbVPtr, fcbClmpSize,fcbFType, fcbDirID`, and `fcbCName`. The following VCB fields are valid: `vcbSigWord, vcbVRefNum, vcbCrDate, vcbLsMod, vcbVolBkUp, vcbAtrb, vcbClpSiz, vcbNmAlBlks, vcbAlBlkSiz, vcbFreeBks, vcbVN, vcbDrvNum, vcbDRefNum, vcbNmFls, vcbNmRtDirs, vcbFilCnt, vcbDirCnt`, and `vcbFndrInfo`.

1.2.32 Don't modify FCB or VCB data structures

The FCB and VCB data structures are maintained for compatibility reasons only. They should be considered as read-only entities. Note that directly accessing FCB and VCB data structures is discouraged in Mac OS 8.

1.2.33 Don't call GetDrvQHdr or AddDrive

`GetDrvQHdr` points to a list which is always empty on Mac OS 8. Consequently, AddDrive is not supported. Use explicit File Manager calls to determine information about the volume list.

1.3 Discouraged in Mac OS 8

This section details features that are discouraged in Mac OS 8 and identifies alternative solutions that are supported. These features are very likely to be unsupported in future versions of the Mac OS.

1.3.1 Don’t draw directly to the screen

Applications that write directly to the base address of the screen will continue to work under Mac OS 8. However, bypassing the system graphics systems prevents this type of application from being able to take advantage of video acceleration.

Note: As before, ShieldCursor and ShowCursor should be used to ensure that the cursor isn’t overwritten in the frame buffer.

1.3.2 Don’t patch within an application

Although global patching isn’t supported, patching within an application is still allowed. `SetTrapAddress`is supported for compatibility. However, you should try to minimize your use of patching, since it lowers your applications performance, lowers overall system reliability and introduces compatibility risks.

1.3.3 Don’t access memory in other applications

Some applications may pass pointers to data to each other through Apple events, Gestalt routines, or other means. This mechanism will break in future system software releases when each application runs in its own protected address space. However, since under Mac OS 8 all applications (not servers or drivers) run in the same address space, sharing data across applications is discouraged but allowed.

Furthermore, Mac OS 8 supports several alternatives for sharing which will continue to work when we transition to protected address spaces for applications (i.e. AppleEvents, kernel messaging, shared memory, etc.).

1.3.4 Don’t access low memory directly

Under Mac OS 8 the LMSet/LMGet calls still change low-memory locations, so writing and reading directly from low memory is still technically supported. However, direct access will stop working in future releases of Mac OS that run applications in separate address spaces.

1.3.5 Don’t use a Custom MBDF

Under Mac OS 8, custom menu bar definition procedures (MBDF) are supported only to a limited extent. The current user selectable theme maintains control of the menubar and its appearance. Custom MBDFs only get called to process requests not related to drawing operations.

1.3.6 Don’t patch toolbox definition procedures

Definition procedures will continue to follow the same definition procedure message protocol that they did in System 7.x so that handlers installed to customize particular behaviors (e.g. `wDraw` for WDEFs) will continue to function. However, this practice is discouraged because any drawing operations will not be visually compatible with the currently selected theme.

1.3.7 Don’t rely on pairing of suspend & resume events

Some applications assume that suspend and resume events are always paired so that they don’t have to examine the event to determine which of the two was sent (they just toggle a boolean to differentiate the state). Generally, this will continue to work in Mac OS 8, but is somewhat problematic when another application unexpectedly terminates. When this happens, the "toggle" assumption is invalid.

1.3.8 Don’t assume that current process and front process are the same when launching

Some applications assume that they are always launched into the foreground. This is an invalid assumption because the current foreground application must process its suspend event before this is true.

1.3.9 Don’t use TopMapHndl and SysMapHndl lowmems

Both TopMapHndl and SysMapHndl require the caller to have knowledge of the in-memory Resource Map data structure. Using any knowledge of Resource Manager internals is discouraged in Mac OS 8. Future releases of the Mac OS may not support TopMapHndl and SysMapHndl.

1.3.10 Don’t use vCheckLoad

The `vCheckLoad` vector is a private Resource Manager hook that has been reverse-engineered by the developer community. With past versions of the OS, it has been maintained to avoid breaking application compatibility. With Mac OS 8, it is still being maintained at a high cost to runtime performance.   Future releases of the Mac OS will not support vCheckLoad.

1.3.11 Don’t use ROMBase to test your environment

Some applications test ROMBase (and information found by dereferencing ROMBase) to verify that they are running on a system release which supports their minimum features. Use Gestalt instead.

1.3.12 Don't use obsolete names for Text Utilities

The obsolete names and their replacements are documented in *Inside Macintosh: Text, Appendix D.*

1.3.13 Don't use negative selectors for GetScriptVariable().

These negative verbs provide access to the so-called "developer routines." While they will continue to be supported in Mac OS 8, we encourage application developers to use Quickdraw GX instead, which removes the need to use these routines.

1.3.14 Don't assume the system script is the same in all processes.

Mac OS 8 makes it possible to correctly run applications with a different localization than the workspace or other applications. This means that many Script Manager variables will vary from process to process. Be sure you always check these values for the current process before using them. It also means that changes made to a Script Manager variable may only affect the process that the change was made in.

1.3.15 Don't assume resource and file reference numbers are interchangeable

Use the Resource Manager calls wherever possible. Do not assume that an open **resource** file reference number is, in fact, an open **file** reference number. They are no longer interchangeable.

1.3.16 Don't use working directories

The Mac OS 8 `FSSpec` to `FSObjectRef` translation facilities cannot use working directories. Use `FSMakeFSSpec` to create file specifications.

1.3.17 Avoid registering Gestalt selectors based on selector

functions

Mac OS 8's multiple address spaces make exposed addresses that might be referenced from other processes dangerous. Instead of registering selector function (procptr) based Gestalt selectors, use value-based selectors whenever possible. Use `NewGestaltValue`, `SetGestaltValue`, and `ReplaceGestaltValue` when they are available instead of `NewGestalt` and `ReplaceGestalt`.

1.3.18 Make no assumptions about resource ordering using index calls

The ordering of the resources as returned from the index-based Resource Mgr calls is not guaranteed to persist. The order will remain fixed so long as the resource file is opened and the file is not edited. Once the file is closed, the ordering of the resource returned upon the following open is not guaranteed to be the same. The System 7.X ordering of resources is currently being maintained by Mac OS 8 for application compatibility at high run time cost. This support will be removed in future releases of the Mac OS.

1.3.19 Make no assumptions about editing behavior of resource file

Due to resource file format limitations, the Resource Manager is limited in both the number and size of resources in a resource file. Some applications, having hit these limits, used knowledge of the editing behavior of the Resource Manager to increase the number and/or the size of resources in a single resource file. In Mac OS 8, changes to the algorithms used when editing a resource file will break applications that rely on some of the behaviors of the past.

1.3.20 Don’t rely on Memory Manager implementation

The reliance on the Memory Manager implementation comes in several forms. Don't rely on the layout of block headers including the layout of the headers individual fields. Don't use knowledge of the space overhead of Memory Manager private data structures including heap headers and block headers. Don't use knowledge of Memory Manager allocation behavior to predict memory use and/or memory layout of future allocations.

In general, the block headers, zone headers and size overhead of zones are private to the implementation of the Memory Manager. Applications which depend upon these data structures prevent Apple from improving the performance and stability of the Memory Manager.

1.4 Well-Behaved in Mac OS 8

This section details features that are well-behaved in Mac OS 8. These features are likely to work compatibly in future versions of the Mac OS.

1.4.1 Be PowerPC native

Native or FAT applications are first class citizens on Mac OS 8 since the operating system is almost entirely native. During the rewrite of the system, performance choices were made to favor the PowerPC based application.

If you have accelerated applications but portions of your application are still 68k code, consider porting those portions to PowerPC as well. Portions of your applications which were not performance sensitive on System 7.x may be more noticeable on Mac OS 8.

In addition, new API introduced in Mac OS 8 will not have A-traps associated with them, and so, will not be accessible to 68k code.

1.4.2 Use GX printing

While traditional printing calls are supported for backward compatibility, the printing implementation for Mac OS 8 is PowerPC native QuickDraw GX. This means that your application will print faster and more reliably if you support the QuickDraw GX printing API. Support for both GX and traditional printing can be maintained by runtime Gestalt checks.

1.4.3 Use Open Transport

The PowerPC networking implementation for Mac OS 8 is PowerPC native Open Transport. The traditional AppleTalk and MacTCP APIs are still supported for backward compatibility. However, optimal networking performance is only available by adopting Open Transport.

1.4.4 Use LowMem accessors

With the advent of the Power Macintosh, new calls were added to provide access to supported low-memory locations. By migrating to the LMGet/LMSet accessor functions today, you can be assured that you aren’t relying on any undocumented low-memory globals. The existing LMGet/LMSet calls will eventually be migrated to the individual owner components, where they’ll be made into full-fledged API calls, and their connection with low memory will be broken. Note that for Mac OS 8, the LMGet/LMSet calls still do access the global low-memory area.

1.4.5 Use the latest Universal Interfaces

The universal interfaces (available with some development tools, MPW Pro and develop CDs) will have Mac OS 8 features conditionally added. These interfaces will give an indication of how certain features will (or will not) be supported.

1.4.6 Support only System 7 and later.

In our compatibility labs, we have encountered obsolete code buried in PowerPC native applications. For example, code that checks for Color QuickDraw or does not use the FSSpec based File Manager calls is no longer necessary.

1.4.7 Minimize patching

A well-written PowerPC-native application should not have to patch any traps. Relying on undocumented side effects may cause your application to break unexpectedly with any new system software release.

1.4.8 Factor your application

Separating your application into distinct parts has several advantages under Mac OS 8. Even in System 7.x, separating your application into a user-interaction portion and a “back-end” is a necessary step to making your application scriptable. Furthermore, in Mac OS 8, you will be able to take advantage of tasking (See the *Mac OS 8 Transitional Application* document or *Mac OS 8 Only Application* document).

1.4.9 Use standard definition procedures

In Mac OS 8, any number of user-selectable themes can be chosen. The appearance of windows, menus, and controls can vary dramatically from theme to theme. Custom definition procedures will look the same no matter which theme the user selects. If you must use them, give the user the option of turning them off to revert to the selected appearance.

1.4.10 Be VM friendly

There is no way to turn virtual memory off on Mac OS 8. If you have your own memory management system and/or have been telling users of your application to "Turn Virtual Memory off", you will probably have to do some performance tuning to achieve optimal performance on Mac OS 8. You can start to do this work by getting your application to a) run acceptably on System 7.x with virtual memory turned on; and b) not use System 7.x Virtual Memory calls. The virtual memory (paging) performance of your application will improve on Mac OS 8.

1.4.11 Locate special folders using Folder Manager

Use FindFolder, which has been available since System 7, to locate the System Folder, Preferences folder, Extensions folder, and other system-created folders. In addition, all user-specific preference information should be stored in the Preferences folder. By using FindFolder, and correctly storing your preferences, you'll be compatible with the workspaces mechanism in Mac OS 8, which allows different system users to have their own sets of application settings.

1.4.12 Specify stack size in code fragment resource

If your application needs additional stack space above and beyond the default stack size, it should use the application stack size field provided in the code fragment ('cfrg') resource. Calls to GetApplLimit and SetApplLimit have no effect on PowerPC-native applications in Mac OS 8.

1.4.13 Be WorldScript savvy

Don't assume text is in a single script. Don't assume a one-to-one relationship between characters and displayed glyphs: always use text measuring routines on entire runs of text instead of character by character. Similarly, use `PixelToChar`() and `CharToPixel`() for hit testing and highlighting. Use `GetFormatOrder`() to properly reorder mixed-direction text.

1.4.14 Be Inline input aware

Inside Macintosh: Text chapter 7 details what an application needs to do to become inline input aware. This enables input methods to work directly in the document content without putting up a secondary window. In Mac OS 8, inline input aware applications will have access "for free" to a wide range of services besides input methods which use the same protocol (e.g. spelling checker, hyphenators).

1.5 How To Build a Mac OS 8 Compatible Application

While you can start to build a Mac OS 8 Compatible application on System 7.x with the latest Universal Interfaces, you probably want to use the interfaces and libraries on the **Mac OS 8 Developers Release**: ***Compatibility Edition*** CD. This release will provide helpful feedback while you are compiling, linking and running your application.

Interfaces

The Mac OS 8 version of the interfaces, like all Apple interfaces, are universal to all Apple software. These are the interfaces that Apple engineers use to write their software. The **Mac OS 8 Developers Release**: ***Compatibility Edition*** CD will include the latest version of our interfaces.

Libraries

In addition to the interfaces on the CD, we will include stub libraries on the CD to link your application against. These libraries correspond to the different types of products you might build. They allow you to link against one library without having to know what specific library the service (and symbol) in question came from.

Compiling Your Application

To link a Mac OS 8 Compatible application, use the **`BUILDING_FOR_SYSTEM7_AND_SYSTEM8`** compiler flag to indicate to the system that you are building an application which runs on both System 7.x and Mac OS 8.

Linking Your Application

To link a Mac OS 8 Compatible application, use the **`AppSystem7orMacOS8.stubs`** library in your development environment. Linking against this library ensures that you will not import facilities which are not available on Mac OS 8.

Running Your Application

When running your application against the debug version of the system release (on the **Mac OS 8 Developers Release: *Compatibility Edition***), you may encounter debugger breaks which detect unsupported or discouraged usage patterns. This will help you determine how well your application will run.