# About the Copland File System

**Preliminary**

Developer Press

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010



Simultaneously published in the United States and Canada.

# About the Copland File System

## Contents

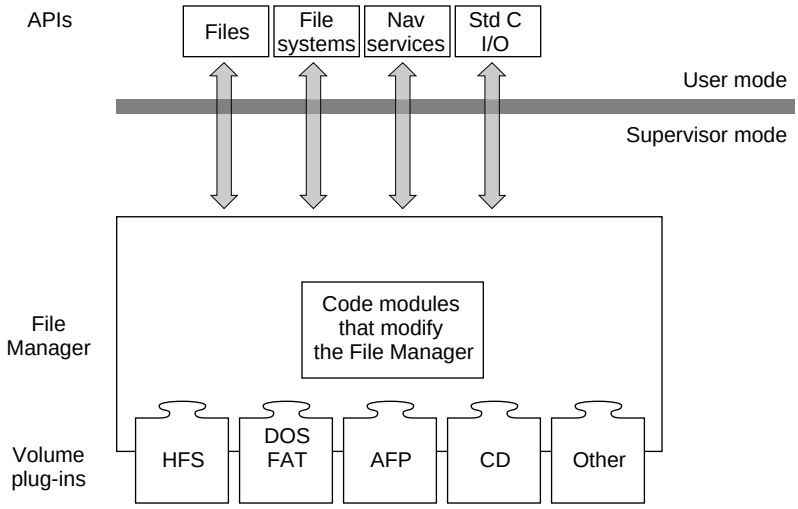
This document introduces the features of the new file system available with Copland. The Copland file system includes several components:

- File Manager
- Navigation Services API
- files API
- file systems API
- standard C I/O API
- volume-format plug-ins
- code modules that modify or add features to the File Manager

Figure 1-1 shows the components of the Copland file system.

The file system manages the organization, reading, and writing of data located on persistent media. You should read this chapter if your product manages or manipulates files. If your software product creates, opens, closes, saves, or renames files, it can take advantage of Copland's new file systems API features for improved performance. If your product provides access to volume formats other than HFS (for example, a standard format such as DOS FAT or a custom format optimized for your customers' particular needs), the Copland file system plug-ins simplify your product development.

**Figure 1-1** The components of the Copland file system




This document describes the architecture of the Copland file system, briefly describes the Navigation Services and file systems APIs, and describes the services provided by the Copland File Manager for plug-ins.

# Compatibility—Backward and Forward

The File Manager in Copland provides complete support for all System 7 File Manager functions used strictly in the manner documented in *Inside Macintosh: Files*.

If your application uses System 7 File Manager routines (and their associated data structures) to manage files and volumes and does not directly manipulate the data structures or low-memory global variables used by the System 7 File Manager, it is compatible with the Copland File Manager. (Specific instances of when your application might not be compatible with the Copland File Manager—as well as specific recommendations about what you can do now to prepare your application to support the Copland File Manager—are described at the end of this chapter.)

In Copland, the File Manager APIs are implemented as dynamically linked libraries (DLLs), so that only the code you actually use is loaded into memory. Whereas the files API consists of a fairly large block of code that translates the old File Manager calls into messages understood by the Copland File Manager, the new file systems API functions map much more directly into file system messages and require much less code in memory. The file systems API functions are simpler, easier to use, and more powerful than the old File Manager functions. The Navigation Services API provides an improved user experience, and supports all the new human interface features being introduced with Copland. Whereas applications using the Standard File Package will continue to work under Copland, they will look dated and will not offer the full Copland user experience.

So that Apple Computer can respond more quickly and easily to your future needs, it has designed flexibility and extensibility into the Copland File Manager. The Copland file systems API will continue to be supported in Gershwin and beyond. Apple Computer is not committed to supporting the old File Manager APIs past Copland.

# The Components of the Copland File System

The components of the Copland file system are shown in Figure 1-1 on page 1-4. This section briefly describes each of these components.

## The Copland File Manager

The Copland File Manager is designed to

- run faster and more efficiently than the old File Manager
- be fully reentrant
- be able to run multiple requests simultaneously (referred to as being *concurrent*)
- support larger volumes and files
- support a variety of volume formats in addition to an improved version of the HFS volume format

- support international filenames by using text objects instead of text strings
- work closely with the virtual memory system and microkernel
- make extensive use of the Copland event notification system

## Increased Speed

The Copland File Manager provides significant performance improvements over previous versions of the File Manager.

- **Increased efficiency.** Better algorithms improve the performance of the Copland File Manager. For example, the Copland File Manager can perform several operations concurrently. The File Manager dispatches and schedules many threads of execution to complete its work as quickly as possible.
- **High-performance paging and memory cache.** The Copland File Manager is integrated with Copland's virtual memory system to provide high-performance paging and a new cache architecture, both of which take advantage of the Copland I/O system. This integration results in dramatic improvements in memory use.
- **Multitasking.** By taking advantage of Copland's multitasking capabilities, your application can create secondary tasks to efficiently perform file I/O processing in the background.

## Reentrancy

The Copland file system is fully reentrant; you are not restricted to calling the File Manager from primary tasks, as you are with the Toolbox.

## Improved HFS Volume Format

To take advantage of contemporary storage technology, the Copland file system supports volumes and forks up to $2^{63}$ bytes. Up to this limit, the sizes of the volumes and forks that the File Manager can support are limited only by the capabilities of a specific volume format.

A newly implemented HFS volume format, for example, supports $2^{48}$ byte volumes and $2^{31}$ byte forks. Table 1-1 compares the limits of the HFS volume formats for System 7 and Copland.

**Table 1-1** HFS volume and fork sizes

| System version | Maximum volume size | Maximum fork size | Corresponding minimum allocation size |
|---|---|---|---|
| 7.1 | 2 GB | 2 GB | 32 KB |
| 7.5 | 4 GB | 2 GB | 64 KB |
| Copland | 256 TB | 256 TB | 4 TB |

(Because HFS allocates storage in units called allocation blocks and accounts for those with 16-bit values, the smallest unit of allocation in a volume is 1/65,536 the size of the volume. The size of an allocation block can be stored in a 32-bit value, and the allocation block size field can go up to a volume size of 2 terabytes.)

## Support for Third-Party Volume Formats

The Copland File Manager does not include any code that is specific to a particular volume format. Instead, it passes messages to volume plug-ins. Each volume plug-in handles I/O for a specific volume format. To facilitate support for a wide variety of volume types, the Copland File Manager provides a variety of services for third-party volume plug-ins, including

- cache management
- range locking
- B*Tree support
- notification services
- control-block services

## International String Support

The Copland File Manager manipulates volume, folder, and filenames internally as text objects rather than as Pascal or C text strings. Text objects include information about the language system and the text encoding system used, in addition to the character codes themselves. They can use any system of character codes, including Unicode. The use of text objects means that the File Manager can properly handle file, folder, and volume names in any language.

### Virtual Memory and Microkernel Integration

The File Manager and virtual memory system in Copland are closely integrated to provide highly efficient virtual-memory paging.

Similarly, the microkernel works directly with the File Manager to clean up after the unexpected termination of a process or other exceptional condition.

### Use of Notification Services

Copland provides an event notification system that allows code modules to publish the occurrence of particular events. Other code modules can subscribe to be notified when specific events occur. Notification of events may be exchanged among the following file-system-related code modules:

- File Manager
- volume-format plug-ins
- File Manager modification code modules
- clients of the File Manager (applications, servers, and so forth)
- the block storage I/O family
- the SCSI I/O family
- other I/O families

## Copland File Manager APIs

This section describes the features of the Copland application programming interfaces to the File Manager and the benefits of the new APIs to users and developers.

### Navigation Services API

Although calls to the Standard File Package continue to work in Copland, the Copland Navigation Services API provides an improved user experience that is consistent with other features of the Copland human interface. Compared to the Standard File Package, Navigation Services provides a better and more

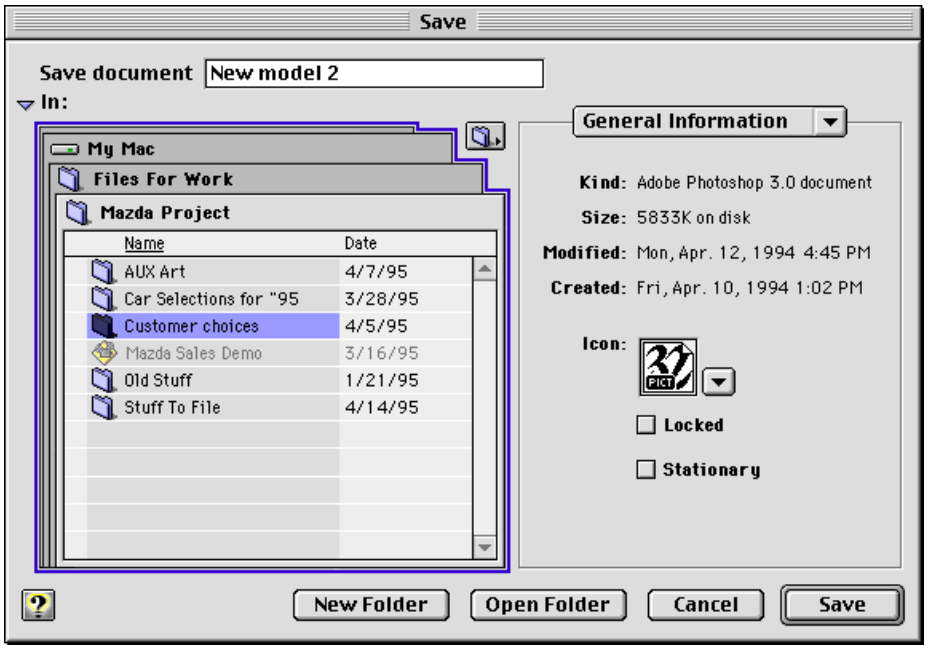
easily customizable browsing interface and better access to document-specific information from within an application.

Here are some of the improvements in the Copland human interface that make document management easier for the user:

- In Copland, when a user selects the New command in the File menu, the application should save the new document to disk when it first creates the document. Users can set the default location for new documents using the Default Location control panel. The first time they choose the Save command for a new untitled document, users are given an opportunity to change its name and location.
- In Copland, your application should replace the Save As command in the File menu with the Save A Copy command. The behavior of the Save A Copy command is more readily understood by users.
- The Find Document command gives users access to the Copland Finder's improved Find facility from within an application.
- The Open, Save, and Save A Copy commands use an intuitive navigation browser that you can customize for your application. The Navigation Browser makes it easy for users to access favorite items quickly, create new folders, and perform other common tasks that are not supported by the Standard File Package.
- The Document Info command in the Edit menu invokes a **document information panel,** similar to the current Get Info window in the Finder, that allows the user to manipulate document-specific information without switching contexts. The same document information panel also appears in the Open and Save dialog boxes.

Figure 1-2 shows the Navigation Browser and one of several document information panels as they appear in the Save dialog box. You can easily specify filters for the files displayed and let the user choose among available filters with a pop-up menu at the bottom of the Navigation Browser. You can also add specialized document information panels for your application's documents that permit editing and display of document information such as file type, author, keywords, colors, and dimensions.

**Figure 1-2** Standard Navigation Browser and General Information panel in the Save dialog box




The Navigation Services API also makes it much easier for applications to display an Open or Save dialog box with just one call, specifying options by using parameters rather than separate calls.

## Files API

In Copland, all of the functions described in *Inside Macintosh: Files* have been implemented by the File Manager to provide backward compatibility with existing software. However, the new file systems API is easier to use and will execute faster than the files API functions; you should not use the files API for any new development.

## File Systems API

The file systems API contains about half as many functions as the old File Manager, to provide a simpler but more powerful interface. The functions you can perform with the file systems API include

- creating a folder
- creating a file
- moving and renaming files and folders
- copying files and directories
- performing stream-access I/O operations on data in files
- opening a file
- establishing a position in an opened file
- allocating storage
- deallocating storage
- closing a file
- performing backing-store I/O operations on files
- locating objects in the file system
- searching file system objects for data and information about the objects
- obtaining information about specific file system objects
- iterating over file system objects to get information about the objects or perform operations on the objects

## Standard C Library File I/O API

The file system in Copland provides a dynamically linked library that includes all the standard C library file I/O functions, optimized to use the Copland File Manager.

## Volume Plug-ins

The Copland file system specifies a message protocol for volume format plug-ins. Apple Computer provides several plug-ins for volume formats, and you can provide your own.

The Copland file system will initially support

- HFS
- AppleTalk Filing Protocol (AFP), which allows access to AppleShare and Personal FileShare volumes
- DOS FAT
- These common CD-ROM formats:
  - High Sierra
  - ISO 9960
  - Photo CD
  - Audio CD

## File Manager Modification Code Modules

Traditionally, third parties have modified the behavior of the File Manager in order to provide such system-enhancing features as data compression utilities, data encryption services, and utilities that control data access. The Copland File Manager provides an interface that allows developers to add such features to the file system.

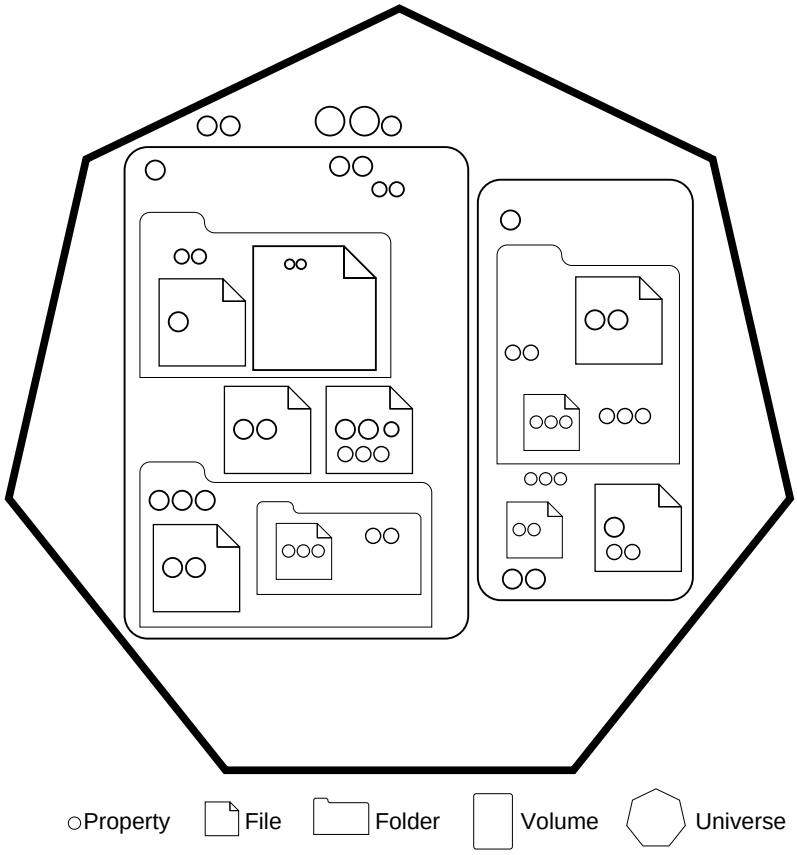
# Copland File Manager Architectural Overview

This section discusses two aspects of File Manager architecture in Copland: the way in which the File Manager organizes information and the prototypical flow of a file operation through the File Manager.

## Data Organization

Afile system must store persistent information in a manner that is efficient to access, where information includes everything from a user's data to the code modules of the File Manager itself. To do this, the File Manager in Copland uses a hierarchical information-storage model in which every file-system object is a container for information. Figure 1-3 shows how data is organized in this file system.

**Figure 1-3** The organization of data by the File Manager



In the past, file systems—such as Macintosh HFS—have defined a specific set of properties that are associated with each file. In HFS, for example, a call to the `PBGetCatInfo` function returns a predetermined set of parameters that describe a file. The File Manager in Copland, on the other hand, defines a file simply as a collection of data items of any size or content. It is up to each specific file system running under the File Manager to determine the meaning and content of each data item. This more generalized definition of a file allows the File Manager to support such diverse file systems as HFS, UNIX, and DOS FAT.

As you can see from Figure 1-3, every type of file system object in Copland contains file system properties. A **property** is a data item or a set of data that is stored by the file system. Properties can be simple data items, such as dates, file types, and icon definitions; or they can be expandable sets of data such as the data fork and resource fork of a file. Each property has a value, and the value of the property has an actual size and a certain amount of allocated space. A file system property cannot exist by itself; it must be contained in a file system object. The properties contained by an object define the object and make it identifiable to the File Manager. In addition to containing properties, certain types of file system objects can contain other file system objects.

File system objects include

- files, which can contain fork properties and simple properties
- folders, which can contain files, other folders, and simple properties
- volumes, which can contain files, folders, and simple properties
- the universe, which contains all of the other objects in the file system plus simple properties

There is only one universe, which represents the user's computer system including all mounted volumes. Because the universe is transient, existing only while the system is running and changing every time the user mounts or unmounts a volume or reconfigures the system in some other way, the properties contained in the universe are transient and are created by the File Manager each time the computer starts up.

The file systems API includes functions that allow you to iterate into and out of objects and over the objects contained in a given object. You can perform the same operation on all of the objects over which you iterate, or you can obtain the properties of the objects. You can direct the File Manager to iterate over one object at a time, or to perform the same operation on all of the objects in one container. The latter case, known as *bulk iteration*, can significantly reduce

network traffic when you want to iterate over objects on a volume mounted over a network. The concept of the universe makes it possible to iterate across volumes with a single function call.

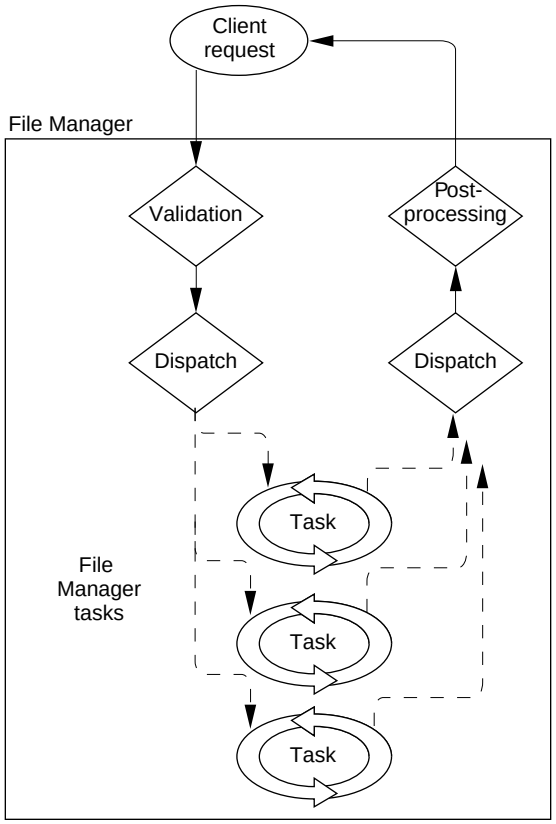
## File Operations

You can submit a request to the File Manager to be processed either synchronously or asynchronously. If you submit a synchronous request, your task is blocked until the File Manager completes the operation. If you submit an asynchronous request, your task can continue processing as soon as the File Manager's dispatcher has queued the request. In either case, the File Manager's internal tasks follow the same procedure to complete the request.

### Synchronous Requests

Figure 1-4 illustrates the processing flow of a synchronous request to the File Manager. As soon as the File Manager receives the request, your task is blocked and the File Manager divides the request into as many concurrently-executing tasks as possible. When all of the tasks have completed execution, the dispatcher passes execution to the postprocessor. When the postprocessor completes the operation, it returns control to your task, which can then continue execution.

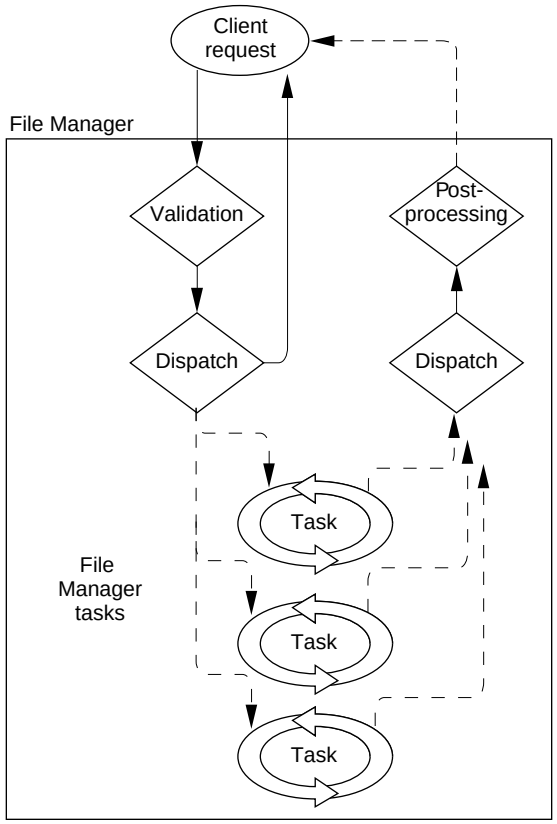
**Figure 1-4** Processing a synchronous File Manager request




## Asynchronous Requests

Figure 1-5 illustrates the processing flow of an asynchronous request to the File Manager. As soon as the File Manager queues the request, it returns control to your task, which can continue execution. The File Manager divides the request into as many concurrently executing tasks as possible. When they've all completed execution, the dispatcher passes execution to the postprocessor. When the postprocessor completes the operation, it sends a message to your task, informing it that the operation is complete.

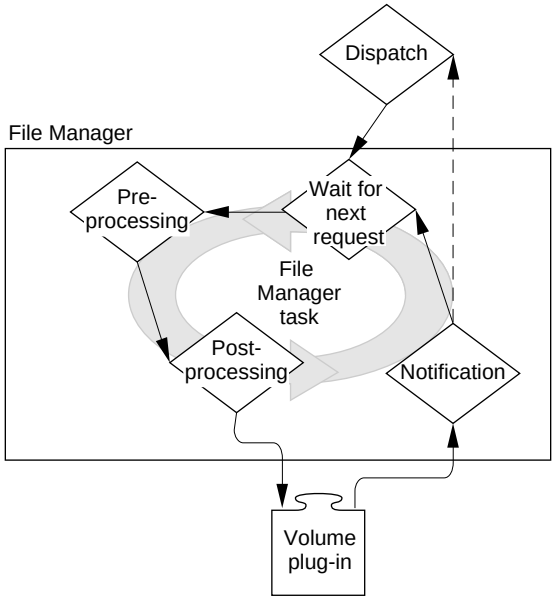
**Figure 1-5** Processing an asynchronous File Manager request




## File Manager Task Processing

Figure 1-6 shows the processing sequence of a typical File Manager task. When a task receives a request from the dispatcher, it performs some preprocessing, and then passes the request on to the volume plug-in for the requested volume. The volume plug-in carries out the request on that volume, and then passes control to the task's postprocessor. When the postprocessor is finished, it passes control to the notification code, which sends a message to the dispatcher telling it the task is complete. The task then returns to waiting for requests.

**Figure 1-6**




# Preparing Your Product for the Copland File Manager

The following recommendations are offered to assist you in preparing your software product now to take advantage of the File Manager in Copland.

- Make your application native for PowerPC-based computers.
- Don't assume 31-character filenames. Many volume formats have shorter or longer names.
- Do not use newline mode, because in the System 7.5 File Manager, newline mode is a mixed-mode switch. Instead, you should write your own function to put data into a buffer, and then write another to get each line of data out.
- Depending on the amount of memory you have and the speed of the device you are using, consider reading and writing file data in multiples of 16 KB.

You have less overhead if you read an entire file into a buffer with one call instead of using multiple calls.

- Use the `noCacheBit` flag in the `ioPosMode` field of the parameter block. Don't cause other blocks of the cache to be flushed out to make room for data if you're not going to reread it.
- Isolate data storage methods.
- Isolate information calls. That is, write your own functions to get the data you need. For example, put wrappers around calls to `PBGetCatInfo` to define the information you require (for example, to get a file's type or creator).
- Keep reads and writes block aligned.
- Set the following when compiling your code:

```
OLDROUTINENAME=0
OLDROUTINELOCATION=0
```

- Preallocate forks before writing file data.
- Use 64-bit math for operations on all file system size data.
- Use the `FSSpec` structure to specify files and directories, because use of partial pathnames is limited on the File Manager in Copland.

To improve I/O performance, the Copland File Manager provides concurrent processing of file processing requests. This has two important implications that you must consider when preparing your software product for Copland.

- Your software cannot read or write any data after passing it to the File Manager because your application cannot depend on the state of that data.
- If your code extends the File Manager, it must also be reentrant and able to handle concurrent requests.

As mentioned at the beginning of this chapter, your application will be compatible with the File Manager if it uses System 7 File Manager routines (and their associated data structures) to manage files and volumes, and if it does not directly manipulate the data structures or low-memory global variables the File Manager uses in System 7. Even if you do not take advantage of the new Copland File Manager APIs, you should make sure that your System 7 application runs on Copland. You will most likely have compatibility problems with the File Manager in Copland if your System 7 software product does any of the following:

- creates or modifies data stored in file control blocks (FCBs) or modifies the file control block list
- relies on fields in an FCB other than `fcbFlNum`, `fcbFType`, `fcbCrPs`, `fcbDirID`, `fcbVPtr`, and `fcbCName`
- creates or modifies data stored in volume control blocks (VCBs) or modifies the volume control block queue
- relies on fields in a VCB other than `vcbFlags`, `vcbSigWord`, `vcbCrDate`, `vcbLsMod`, `vcbAtrb`, `vcbNmFls`, `vcbNmAlBlks`, `vcbAlBlkSiz`, `vcbFreeBks`, `vcbVN`, `vcbDrvNum`, `vcbDRefNum`, `vcbFSID`, `vcbVRefNum`, `vcbVolBkUp`, `vcbFilCnt`, `vcbDirCnt`, and `vcbFndrInfo`
- relies on or modifies any low-memory global variables maintained by the System 7 File Manager, such as `FSQHdr`, `FSBusy`, `FSQueueHook`, `ExtFSHook`, `ToExtFS`, `CkdDB`, `CurDB`, `NxtDB`, `FSCallAsync`, `MaxDB`, `FlushOnly`, `RegRsrc`, `FLckUnlck`, `FrcSync`, `NewMount`, `NoEject`, `DrMstrBlk`, `HFSStkTop`, `RgSvArea`, `hfsVars`, `HFSStkPtr`, `XRgSvArea1`, `HFSFlags`, `CacheFlag`, `SysCRefCnt`, `XRgSvArea2`, `SysBMCPtr`, `SysVolCPtr`, `SysCtlCPtr`, `XRgSvArea3`, `PMSPPtr`, `HFSTagData`, `XRgSvArea5`, `HFSDSErr`, `CacheVars`, `HFSVarEnd`, `cacheCom`, `XRgSvArea6`, `HFSDefaults`, `FmtDefaults`, `ErCode`, `Params`, `FSTemp8`, `FSTemp4`, `FSIOErr`, and `FSVarsPtr`
- relies on data returned by `GetFCBInfo` other than that returned in the `FCBPBRec` fields `ioNamePtr`, `ioVRefNum`, `ioRefNum`, `ioFCBIndx`, `ioFCBFlags` (only bits 9 and 14), `ioFCBEOF`, `ioFCBPLen`, `ioFCBCrPs`, `ioFCBVRefNum`, and `ioFCBParID`

A number of System 7 File Manager APIs are superfluous in Copland. Therefore, these calls perform no action and instead return the `noErr` result code: `PBDTCloseDown`, `PBDTDeleteAsync`, `PBDTDeleteSync`, `PBDTRestAsync`, `PBDTResetSync`, `DILoad`, `DIUnload`, and `FInitQueue`. The functions `PBDTGetPath` and `PBDTOpenInform` return reference numbers usable only by other PBDT calls.

The File Manager in Copland provides only limited support for the System 7 patching mechanism. To give your application greater control over the APIs that it absolutely needs to modify, Copland provides new mechanisms for patching the File Manager.