File System Overview

Data Management: File Management

2009-08-14

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

Simultaneously published in the United States and Canada.

Contents

Tables

Introduction to the File System Overview

The file system is an important part of any operating system. After all, it's where users keep their stuff. In Mac OS X, the organization of the file system plays an important role in helping the user find files. The organization also makes it easier for applications and the system itself to find the resources they need to support the user.

The file system in Mac OS X has at its core a set of directories inherited from the Berkeley Software Distribution (BSD) operating system. While most of these directories are actually hidden by the Finder, many elements of the BSD world are still apparent. The file permissions model, symbolic links, and user home directories are all concepts inherited from BSD. Mac OS X also adds many of its own concepts to provide the user with a secure and elegant environment for managing files and folders.

The Mac OS X file system was designed to provide power and flexibility while maintaining the traditional ease-of-use users expect. To this end, the file system provides users with a consistent structure that makes it clear where resources are located. (This consistency also helps developers, whose applications need to know where important resources are located.) Other file system conventions, such as aliases, extension hiding, and display names also enhance the user experience.

Organization of This Document

The Mac OS X File System contains the following articles:

- "File-System Domains" (page 9) describes the high-level organization of the file system in Mac OS X.
- "The Library Directory" (page 15) describes the standard subdirectories that are used to configure the system and user environments.
- "The Developer Directory" (page 19) describes the developer-specific directories that are installed with the Xcode Tools.
- "Directories of the Classic Environment" (page 21) describes the legacy directories created for the Classic compatibility environment.
- "Where to Put Application Files" (page 23) provides guidelines on where applications should place non-essential configuration and support files.
- "Files and the Finder" (page 27) describes the role of the Finder in managing the file system. It also explains some of the techniques the Finder uses to associate files with applications.
- "Sorting Rules" (page 31) explains the rules for ordering file and directory names in Finder windows.
- "File System Guidelines" (page 33) offers tips and advice on how best to support Mac OS X file system features.
- "Filename Extensions" (page 35) describes the Mac OS X support for filename extensions and how to support them in your applications.
- "Display Names" (page 39) explains the difference between file names in the file system and the file names that users see. It also explains when your application should use display names.

- "BSD Permissions and Ownership" (page 43) describes the principles behind file permissions and their implications for file management in Mac OS X.
- "Access Control Lists" (page 49) provides an overview of access control lists and how they are used to supplement BSD permissions.
- "File System Comparisons" (page 51) offers a comparison of features between HFS+ and UFS volume formats.
- "Aliases and Symbolic Links" (page 53) describes the differences between aliases and symbolic links.

File-System Domains

On a multi-user system, controlling access to system resources is important for maintaining the stability of the system. Mac OS X defines several file-system domains, each of which provides storage for resources in an established set of directories. Access to resources in each domain is determined by the permissions for the current user.

There are four file-system domains:

- **User**. The user domain contains resources specific to the user who is logged in to the system. This domain is defined by the user's home directory, which can either be on the boot volume (`/Users`) or on a network volume. The user has complete control of what goes into this domain.
- **Local**. The local domain contains resources such as applications and documents that are shared among all users of a particular system but are not needed for the system to run. The local domain does not correspond to a single physical directory, but instead consists of several directories on the local boot (and root) volume. Users with system administrator privileges can add, remove, and modify items in this domain.
- **Network**. The network domain contains resources such as applications and documents that are shared among all users of a local area network. Items in this domain are typically located on network file servers and are under the control of a network administrator.
- **System**. The system domain contains the system software installed by Apple. The resources in the system domain are required by the system to run. Items in this domain are located on the local boot (and root) volume. Users cannot add, remove, or alter items in this domain.

The domain for a given resource determines its applicability or accessibility to the users of the system. For example, a font installed in the user's home directory is available only to that user. If an administrator installs the same font in the network domain, all network users have access to it.

Within each domain, Mac OS X provides a set of initial directories for organizing the contained resources. Mac OS X uses identical directory names across domains to store the same types of resources. This consistency simplifies the process of finding resources both for the user and for the system methods that use those resources. When the system needs to find a resource, it searches the domains sequentially until it finds the resource. Searches start in the user domain and proceed through the local, network, and system domains in that order.

Your code should never assume the path to a resource within a file-system domain, as those paths could change in the future. Apple provides public interfaces for accessing standard file-system paths. You should always use these interfaces to locate system resources. See "Searching Within the File-System Domains" (page 12) for more on searching for items within the domains.

The following sections describe the file-system domains in more detail, including some of the standard directories available in each domain.

The User Domain

The user domain contains resources that are specific to a single user. The user domain is represented by the home directory of the current (logged-in) user. Each user of a Mac OS X computer must have an account on that computer or on the local area network to which the computer is connected. Each user account comes with an assigned area of space in the file system, called the user's home directory. This directory is where the user's programs, resources, and documents reside. The name of each user's home directory is based on the user's short login name, which must be unique.

The user domain makes a customized working environment possible for each user. When a user logs in, the Finder restores the user's working environment and settings to their previous state using the preferences in the user domain. Similarly, programs and other system software use information in the user domain to restore application preferences, network settings, email settings, font sets, ColorSync profiles, and other settings.

The location of the user's home directory depends on the user account. If the user account is local to the computer, the user's home directory is in the `Users` directory on the boot volume. If the user account is a network account, the home directory is on a network server. Regardless of the physical location of the home directory, Mac OS X uses the UNIX convention of a ~ (tilde) character in some situations to indicate a user's home directory. The tilde character can be used in combination with other directory names or user names to specify specific user directories. Table 1 (page 10) illustrates this concept.

Table 1 Uses of tilde to indicate locations in home directories

`~`	Top level of current user's home directory
`~/Library/Fonts`	Where fonts are stored in current user's home directory
`~Steve`	Top level of user Steve's home directory

The home directory for each new user comes with some default directories and resources in place. If the user has a .Mac account, these directories are mirrored on the user's iDisk as well. (For more information on iDisk, go to http://www.mac.com.) Table 2 (page 10) lists some of the common directories you might find in a user's home directory.

Table 2 Home directory contents

User directory	Description
`Applications`	Contains applications available only to the current user.
`Desktop`	Contains the items the Finder displays on the desktop for the logged-in user.
`Documents`	Contains the user's personal documents.
`Library`	Contains application settings, preferences, and other system resources that are specific to the user. Should not contain user data. See "The Library Directory" (page 15).
`Movies`	Contains digital movies in QuickTime and other formats.
`Music`	Contains digital music files (`.aiff`, `.mp3`, `.m4p`, and other formats).
`Pictures`	Contains image files in a variety of formats.

User directory	Description
`Public`	Contains items the user wishes to share with other users. By default, this directory is accessible to other users.
`Sites`	Contains web pages for the user's personal website. Web Sharing must be enabled before these pages are accessible to other users.

When a user account is created, an `Applications` directory is not automatically added to the home directory. However, users can create an `Applications` directory and put their own applications in it. The system automatically searches for applications in this location.

The system protects the files and directories in the user's home directory from outside interference by a set of default permissions, which the user may change at any time. Any new folders created by the user inherit the privileges of the parent directory.

In addition to the individual home directories, the `Users` directory contains a `Shared` subdirectory. This directory is accessible to any user of the local computer system and is intended for use only by users; applications should not store application-specific content here, unless explicitly directed to do so by the user. Any user can write documents to, retrieve documents from, and read documents in this directory. Although this directory is not really associated with the user domain, it provides a convenient means for users to exchange documents and other files.

The Local Domain

The local domain contains resources that are available on the local computer but are not required by the system to run. Resources in the local domain typically include applications, utilities, custom fonts, custom startup items, and global application settings. The `Applications` and `Library` directories on the root volume contain the resources for the local domain. These resources are available to the current user of a computer system but are not available to users on other networked computers.

Administrators of a computer can install resources into the local domain if they want those resources to be shared by all users of the system. Apple ships its applications in the `/Applications` and `/Applications/Utilities` directories. Third-party applications and utilities should also be placed in these directories. Other system resources, such as fonts, ColorSync profiles, preferences, and plug-ins should be placed in the appropriate subdirectory of the `Library` directory. For more on the `Library` directory, see "The Library Directory" (page 15).

The Network Domain

The network domain contains the resources available to all users of a local area network. Network users can access applications, documents and other resources through this domain, including AppleShare and web servers. The exact composition of the network domain depends on institutional or corporate policy. Implementation of the network domain is the responsibility of the network administrator.

Table 3 (page 12) lists the standard directories available in the network domain, along with a description of the directory contents.

Table 3 Network directories

Location	Description
`/Network/Applications`	Contains applications that can be run by all users on the local area network.
`/Network/Library`	Contains resources—such as plug-ins, sound files, documentation, frameworks, colors, and fonts—available to all users of a local area network. For more on the `Library` directory, see "The Library Directory" (page 15).
`/Network/Servers`	Contains the mount points for the NFS file servers that make up the local area network.
`/Network/Users/`	Contains the home directories for all local-area network users. This is the default location for home directories. Home directories may also be stored on other servers.

The System Domain

The system domain contains the resources required by Mac OS X to run. All resources in the system domain are located in the `/System` directory on the root volume. These resources are provided by Apple and only the root user can modify the contents of this directory. Administrative users and applications cannot install resources in the system domain or modify its contents directly.

By default, the `/System` directory contains only a `Library` subdirectory. This subdirectory contains many of the same types of resources as other `Library` directories in the system. However, in the system domain, this directory also contains the core services, frameworks, and applications that make up Mac OS X. For more information on the `Library` directory, see "The Library Directory" (page 15).

Although the Classic compatibility environment contains system-related resources, it is not considered a part of the system domain. For more information about the Classic environment, see "Directories of the Classic Environment" (page 21).

Searching Within the File-System Domains

Mac OS X includes two public programmatic interfaces you can use to search for resources, plug-ins, and other items within specific directory locations of specific (or all) domains. One of these interfaces—the `FindFolder` function of the Folder Manager—is for Carbon or other C-based programs; for more information, see *Folder Manager Reference*. The other interface—the functions and constants defined in `NSPathUtilities.h` in the Foundation framework—is for Cocoa programs; for more information, see *Foundation Framework Reference*.

Both interfaces help you search through all file-system domains for a particular item. By convention, searches typically begin with the most specific domain and end with the most general. This domain order is as follows:

1. User
2. Local

3. Network
4. System

Most system software follows this order when it searches for items through all file-system domains. However, you may search in any domain order that is appropriate to your application's needs.

The Library Directory

The `Library` directory is a special directory used to store application-specific and system-specific resources. Each file-system domain has its own copy of the `Library` directory, with access levels to match the domain type. (See "File-System Domains" (page 9) for a discussion of domains.) Although an application can use this directory to store internal data or temporary files, it is not intended for storage of the application bundle itself or for user data files. Application bundles belong in an appropriate `/Applications` directory, while user data belongs in the user's home directory.

Important: You should not store user data files in the `Library` directory or any of its subdirectories. If your application stores the user's data automatically—that is, without prompting the user for a location—you should choose a more appropriate location (usually the `Documents` directory) inside the user's home directory. For the list of user directories, see "The User Domain" (page 10).

The `Library` directory contains many standard subdirectories. System routines expect many of the standard subdirectories to exist, so it is never a good idea to delete subdirectories of `Library`. However, applications can create new subdirectories as needed to store application-specific data.

Table 1 (page 15) lists some of the directories that can appear in a `Library` directory. You should use this table to determine where to put files needed to support your software. This list is not complete, but it lists some of the most relevant directories for developers. Directories that do not appear in all domains are noted appropriately.

Table 1 Subdirectories of the Library directory

Subdirectory	Directory contents
`Application Support`	Contains application-specific data and support files such as third-party plug-ins, helper applications, templates, and extra resources that are used by the application but not required for it to operate. This directory should never contain any kind of user data. By convention, all of these items should be put in a subdirectory named after the application. For example, third-party resources for the application MyApp would go in `Application Support/MyApp/`. Note that required resources should go inside the application bundle itself.
`Assistants`	Contains programs that assist users in configuration or other tasks.
`Audio`	Contains audio plug-ins and device drivers.
`Caches`	Contains cached data that can be regenerated as needed. Applications should never rely on the existence of cache files. Cache files should be placed in a directory whose name matches the bundle identifier of the application. Cache data should further be subdivided into user or session-specific subdirectories as needed. (See *Multiple User Environments* in Mac OS X Documentation for user-specific guidelines.)

Subdirectory	Directory contents
`ColorPickers`	Contains resources for picking colors according to a certain model, such as the HLS (Hue Angle, Saturation, Lightness) picker or RGB picker.
`ColorSync`	Contains ColorSync profiles and scripts.
`Components`	Contains system bundles and extensions.
`Contextual Menu Items`	Contains plug-ins for extending system-level contextual menus.
`Documentation`	Contains documentation files and Apple Help packages intended for the users and administrators of the computer. (Apple Help packages are located in the `Help` subdirectory.) In the local domain, this directory contains the help packages shipped by Apple (excluding developer documentation).
`Extensions`	Contains device drivers and other kernel extensions. (Available in the system domain only.)
`Favorites`	Contains aliases to frequently accessed folders, files, or websites. (Available in the user domain only.)
`Fonts`	Contains font files for both display and printing.
`Frameworks`	Contains frameworks and shared libraries. The `Frameworks` directory in the system domain is for Apple-provided frameworks only. Developers should install their custom frameworks in either the local or user domain.
`Internet Plug-ins`	Contains plug-ins, libraries, and filters for web-browser content.
`Keyboards`	Contains keyboard definitions.
`Logs`	Contains log files for the console and specific system services. Users can also view these logs using the Console application.
`Mail`	Contains the user's mailboxes. (Available in the user domain only.)
`PreferencePanes`	Contains plug-ins for the System Preferences application. Developers should install their custom preference panes in the local domain.
`Preferences`	Contains the user preferences. See *Runtime Configuration Guidelines* for information about user preferences.
`Printers`	In the system and local domains, this directory contains print drivers, PPD plug-ins, and libraries needed to configure printers. In the user domain, this directory contains the user's available printer configurations.
`QuickTime`	Contains QuickTime components and extensions.
`Screen Savers`	Contains screen saver definitions. See *Screen Saver Framework Reference* for a description of the interfaces used to create screen saver plug-ins.
`Scripting Additions`	Contains scripts and scripting resources that extend the capabilities of AppleScript.

Subdirectory	Directory contents
`Sounds`	Contains system alert sounds.
`StartupItems`	Contains system and third-party scripts and programs to be run at boot time. (See *System Startup Programming Topics* for more information about starting up processes at boot time.)
`Web Server`	Contains web server content. This directory contains the CGI scripts and webpages to be served. (Available in the local domain only.)

The Developer Directory

The Xcode Tools CD contains the applications, tools, documentation, and other resources for developing Mac OS X software. Developers install these tools separately from the Mac OS X installation. When you install the tools, the installer places all of the software components in the `/Developer` directory of the boot volume. Table 1 (page 19) lists the contents of this directory.

Table 1 Subdirectories of the developer directory

Directory	Contents
`ADC Reference Library`	Contains the locally installed documentation and links to additional resources available via the web. The pages in this directory offer easy navigation and consistent access to the complete ADC technical collection, including documentation, sample code, and other resources critical to Mac OS X development.
`Applications`	Contains the applications used to manage and build software projects. These tools include Xcode and Interface Builder for creating code and interface files. It also includes a set of performance tools, Java tools, graphics tools, and general utilities.
`Documentation`	Contains additional developer-related documentation.
`Examples`	Contains example projects organized by general type. These are working projects that you can build and use to increase your knowledge of Mac OS X.
`Extras`	Contains optional files that you can install as needed for your development.
`Headers`	Contains special header files, such as the stub “flat” Carbon headers and headers for debugging remote applications.
`Java`	Contains files needed for Java bridging in the Cocoa application environment.
`Makefiles`	Contains makefiles and jamfiles for building and converting legacy projects.
`Palettes`	Contains the Apple-supplied Interface Builder palettes.
`SDKs`	Contains the software development kits used to create software targeted specifically for previous versions of Mac OS X. Each SDK contains header files and stub libraries from a particular version of Mac OS X.
`Tools`	Contains command-line development tools and utilities, including those for creating and manipulating HFS resource forks.

Directories of the Classic Environment

The Classic compatibility environment contains several directories used to support Classic applications. The directories of the Classic environment are the directories of a Mac OS 9 installation. Mac OS X requires an installation of Mac OS 9.1 (or later) for the Classic environment. If a system has an earlier version of Mac OS 9 installed, the user must install a newer version to support Mac OS X.

A system may have multiple versions of Mac OS 9 installed on different partitions. If this is the case, the user can choose the preferred version of Mac OS 9 to use for the Classic environment from the Classic pane of System Preferences. The user can also switch Classic environments at any time or change the startup disk to boot directly into Mac OS 9 (on computers that support it).

When you install Mac OS 9.1 (or later) on a volume, the installer creates several directories to store the system files. Table 1 lists the directories created by the installer along with a description of the contents. If a user already has a version of Mac OS X or Mac OS 9.1 (or later) installed, the Mac OS 9 installer may not create all of these directories.

Table 1 Directories created by the Mac OS 9.1 (or later) installer

Directory	Description
`Applications (Mac OS 9)`	Contains the Classic applications and utilities.
`Documents`	Contains application-specific information. This directory should be used only by Classic applications. Mac OS X applications should store preferences and other application files in the appropriate `/Library` directory. Users should store their documents in their home directory.
`System Folder`	Contains the Classic environment system files.

The first time the user launches Classic, the system adds some required files to the System Folder of the selected Mac OS 9 volume. When a user installs Mac OS X on a system with Mac OS 9 already installed, the installer performs some additional tasks to support the Classic environment. In particular, the Mac OS X installer creates an alias to the Mac OS 9 desktop folder and puts it on the desktop of the admin user who ran the installer. This alias contains links to any files that were on the Mac OS 9 desktop prior to the Mac OS X installation.

Where to Put Application Files

Applications should be placed in the `/Applications` directory or the `~/Applications` directory of the current user. Applications placed in the `/Applications` directory are available to all users on the system. Applications placed in a user's home directory are available only to that user.

All of the resources and data files required for an application to run should reside inside the application bundle. However, applications often come with extra files, such as templates, plug-ins, and other application extensions over which the user has some degree of control, including whether or not they should be installed. Similarly, an application might generate cache and temporary files that should not reside in the application bundle.

The remaining sections include some of the appropriate and inappropriate locations for application files. For information about the purpose and intended content of specific `Library` subdirectories, see "The Library Directory" (page 15).

Support Files

A support file is any type of file that supports the application but is not required for the application to run. Document templates and sample files are simple examples of support files. However, you might store more application-bound information, such as custom configurations or preset data files for your application's workspace. In these instances, the information is intrinsically tied to a specific application (as opposed to the user's data) but is not essential for the application to run.

The preferred location for nearly all support files is in the `Application Support` directory of the appropriate domain. Which domain you choose to store your support files depends on the intended use of those resources. If the resources apply to all users on the system, such as document templates, place them in `/Library/Application Support`. If the resources are user-specific, such as workspace configuration files, place them in the current user's `~/Library/Application Support` directory.

Within the `Application Support` directory, you should always place support files in a custom subdirectory named for your application or company. Normally, you should use the application name, but you might want to use your company name if you have multiple products that share many of the same resources. How you organize the resources in this custom subdirectory is entirely up to you.

Even if a support file is user-specific, your application should not have any trouble accessing it from multiple user sessions. Because of fast user switching and remote logins, it's possible that the same user could be logged into the computer more than once. Support files should not contain any data that would adversely affect the behavior of multiple user sessions. All sessions should see the exact same behavior.

Plug-Ins

Plug-ins are code bundles that extend the behavior of an application. Mac OS X supports several types of specialized plug-ins to handle contextual menus and Internet content among other things. You can also define a plug-in interface for your application that lets third-party developers extend the behavior of your application.

Mac OS X plug-ins typically reside in a subdirectory of the `Library` directory in either the local or system domain. User-specific versions of these plug-ins may reside in the user domain as well. See "The Library Directory" (page 15) for a list of standard `Library` subdirectories.

The preferred location for application plug-ins is inside the application bundle itself. An application bundle can include a `PlugIns` directory for storing native and third-party plug-ins. Users can add plug-ins to this directory using the Get Info window of the Finder. Third-party plug-in developers can also install the plug-ins automatically using an installer package. If your application shares a set of plug-ins with another application, you may also install plug-ins in your custom subdirectory of `Library/Application Support`.

When deciding where to store plug-ins, be sure to consider licensing issues and the intended user experience. Plug-ins stored inside the application bundle always remain with the application. If the user copies the application to a different volume, the application retains any installed plug-ins. However, if you install plug-ins in a `Library/Application Support` subdirectory, those plug-ins may be limited to specific users booting from a specific disk partition.

Temporary Files

Many applications use temporary files to store transient data. The life span of a temporary file varies depending on its intended use. The file may be used to store scratch data or calculations, such as when rendering a 3D image, and deleted immediately upon completion of those calculations. It may store runtime data about the application and be deleted only when the application terminates. It may also linger until the next time the user launches the application. For example, an application typically uses a temporary file to store an autosave version a document, which acts like an insurance policy against application or system crashes.

Mac OS X provides an established set of directories for storing temporary files. The primary directory (`/tmp`) is where most local files go, but you should never hardcode this path into your application. Using hardcoded paths limits the portability and longevity of your code. Instead, Carbon applications should use the `FSFindFolder` function (in the Core Services framework) to obtain a reference to the temporary directory; Cocoa applications should use the `NSTemporaryDirectory` function in Foundation Kit.

When saving temporary files, make sure you use unique filenames. Running applications typically share the same temporary directory. With fast user switching enabled, several instances of the same application might also share this directory. Each instance of your application should easily be able to identify the files it created. You can incorporate the session ID and application name into your temporary file names directory or use that information to identify a subdirectory containing your temporary files. For more information about session IDs and operating safely with fast user switching, see *Multiple User Environments* in Mac OS X Documentation.

Cache Files

A cache file is a special type of file generally used to improve the performance of your application. You can use cache files in situations where retrieving or recreating the data might be an expensive operation. For example, web browsers cache previously visited web pages to improve the load time for that page later, assuming its content hasn't changed.

Cache files should be placed in a custom subdirectory of `~/Library/Caches` in nearly all cases. The name of the custom subdirectory should match the bundle identifier of your application. Because they are typically specific to a single user session, you should also tag your cache files, or their contents, with the session ID of the current user. For information on how to do this, see *Multiple User Environments* in Mac OS X Documentation.

If you have cache files that are relevant to all users of the application, you can store them in the local domain (`/Library/Caches`) instead of the user domain (`~/Library/Caches`).

Don't Pollute User Space

It is important to remember that the user domain (`/Users`) is intended for files created by the user. With the exception of the `~/Library` directory, your application should never install files into the user's home directory. In particular, you should never install files into a user's `Documents` directory or into the `/Users/Shared` directory. These directories should only be modified by the user.

Even if your application provides clip art or sample files that the user would normally manipulate, you should place those files in either the local or user's `Library/Application Support` directory by default. The user can move or copy files from this directory as desired. If you are concerned about the user finding these files, you should include a way for the user to browse or access them directly from your application's user interface.

Files and the Finder

The Finder is how most users interact with files in Mac OS X. It is in the best interest of developers to understand these interactions and make sure their software does not behave in an unexpected way.

Mac OS X supports several different types of file systems natively in the Finder. The most common formats are HFS+ (Mac OS Extended), HFS (Mac OS Standard), and UFS. Other file formats include the Universal Disk Format (UDF) for DVD disks and the ISO 9660 format used for CD-ROM volumes. Each of these file systems implements file storage in slightly different ways, but the Finder masks these differences to provide a seamless user experience.

Copy and Move Operations

Copy and move operations in which the source and destination use the same volume format occur much as you might expect, with the file information appearing at the destination. Things get interesting when files are copied or moved across volumes that support different volume formats.

What happens when a user copies a file from an HFS+ volume to a UFS volume? The file on the HFS+ volume includes additional information, such as the file type and creator codes. It may also include a resource fork, a concept that is not supported by UFS files. When such an operation occurs, the Finder splits out any information that is not located in the data fork of the file and writes it to a hidden file on the destination volume. The name of this file is the same as the original file except that it has a "dot-underscore" prefix. For example, if you copy an HFS+ file named `MyMug.jpg` to a UFS volume, there will be a file named `._MyMug.jpg` in addition to the `MyMug.jpg` file in the same location.

When copying a file from a UFS volume to an HFS or HFS+ volume, the Finder looks for a matching "dot-underscore" file. If one exists, the Finder creates an HFS+ (or HFS) file, using the information in the dot-underscore file to recreate the resource fork and Finder attributes. If the hidden file does not exist, these attributes are not recreated.

Note that the Finder accomplishes these operations through the Carbon API on which it is based.

Note: You can use the BSD `cp` or `mv` commands on an application package (or any other Cocoa) without any ill effects. However, if you use those commands on a single-file CFM application, the copied (or moved) application is rendered useless. For CFM applications, Apple includes the `CpMac` command-line utility in the `/Developer/Tools` directory.

Management of Aliases and Symbolic Links

HFS (Mac OS Standard) and HFS+ (Mac OS Extended) file systems include the file-system entity known as an alias. An alias bears some similarities to a symbolic link in a UFS file system, but the differences are significant. See "Aliases and Symbolic Links" (page 53) for a description of these differences.

How the Finder manages a file-system world in which both aliases and symbolic links coexist is simple. It recognizes symbolic links but creates only aliases (when given the appropriate menu command). Even when it encounters a symbolic link in the file system, it presents it as an alias—that is, there is no visual differentiation between the two. The only way to make a symbolic link in Mac OS X is to use the BSD command `ln -s` from a Terminal window or shell script.

File and Folder Presentation

When the Finder displays a file or folder, it takes great care in making sure that what it displays is what the user expects to see. The user may have several applications on the system capable of handling a given document type. The user may also manipulate file names individually or as a whole through language preferences or filename extension hiding. All of these options are handled automatically by the Finder.

Choosing an Icon

The Finder uses several pieces of information to determine an appropriate icon for a file or folder. The file's bundle bit, type code, creator code, and filename extension all help determine the icon. User settings also play a role. The following steps explain the process used to choose icons for files and directory:

1. The Finder checks to see if an item is a file or a directory. If it is a file, it asks Launch Services for an appropriate icon and displays the icon.
2. For directories, the Finder checks to see if it is a bundle.

   The bundle bit or file extension can indicate that the directory is a bundle and should be displayed as an opaque entity. Most bundles are displayed as opaque entities but some, including frameworks, are not.
3. If the bundled directory has the extension `.app` in its filename, the Finder hides that extension.
4. For a bundled directory, the Finder looks up the type code, creator code, and filename extension in the Launch Services database and uses that information to locate the appropriate custom icon.
5. If no custom icon is available for either a file or directory, the Finder displays the default icon appropriate for the given item type.

   The default icon can differ based on whether the item is a document, unbundled directory, application, plug-in, or generic bundle, among others.

Getting the Display Name

Two additional features that affect the way the Finder presents files to the user are filename extension hiding and filename localization. These features are cosmetic additions to the Aqua interface that alter the displayed name of a file without changing the actual name of the file in the file system. The Finder uses routines provided by Launch Services to obtain a **display name** for each file or folder. A display name takes into account user-specified options and returns a read-only name suitable for display from a user interface.

If your application displays file or folder names, you also need to be aware of display names and use them wherever you would otherwise display a file or folder name. Both Cocoa and Launch Services provide interfaces for obtaining display names. For more information, see "Display Names" (page 39).

Launch Services and the Finder

The Finder relies on Launch Services to launch applications and manage the bindings between applications and documents. If you need to perform operations that involve opening unknown document types, following URLs in a document, launching helper applications, or opening embedded document components, you should investigate the Launch Services API. Among the tasks handled by Launch Services are the following:

- Launch another application
- Open a document or URL in another application
- Identify the preferred application for opening a document or URL
- Register information about the kinds of files and URLS an application is capable of opening
- Obtain the icon, display name, or kind string of a file or URL
- Maintain and update the contents of the Recent Items menu.

For information about Launch Services, including how you can use it in your own applications, see *Launch Services Programming Guide*.

Sorting Rules

Mac OS X provides many ways for users to sort and organize documents using the Finder, including by name, by size, by modification date, and so on. Mac OS X sorting is based on the Unicode Collation Algorithm (Technical Standard UTS #10) defined by the Unicode Consortium. This standard provides a complete and unambiguous sort ordering for all Unicode characters and is available on the Unicode Consortium website (http://www.unicode.org).

The Finder in Mac OS X takes advantage of some sanctioned ways for altering the default sorting behavior defined by the Unicode standard. In particular, the Finder supports the following sorting rules:

- Punctuation and symbols are significant for sorting.
- Digit sub-strings are sorted by numeric value rather than as characters.
- Case is insignificant.

File System Guidelines

The following sections provide guidelines on ways to improve your application's interactions with the Mac OS X file system.

Guidelines for Saving Files

Whenever you save a document, you should make sure the file you create for that document contains the following information:

- A type code
- A creator code
- A filename extension

All files in the Mac OS X file system should have a filename extension. The Finder uses filename extensions to help identify the type of a file. More importantly, other platforms commonly use filename extensions exclusively to identify the type of a file. Some programs, such as network transfer programs, often use filename extensions to determine how to transfer files. Providing a filename extension for your application's data files makes it easier for end users to exchange your data files with users on other platforms. By the same token, including a type code and creator code make it easier to exchange data on legacy Mac OS systems.

For more information on filename extensions, including further guidelines on supporting them, see "Filename Extensions" (page 35).

Use Display Names

Display names improve the user experience in Mac OS X and should be supported by all applications. A display name is a name intended for display only to the user. You cannot use display names in your code to open files. Instead, display names make it possible to localize file and folder names dynamically and hide filename extensions without losing any identifying information about the file.

For more information on how to support display names, see "Display Names" (page 39).

Assume Case Sensitivity

Avoid the assumption that you can perform case-insensitive file comparisons or searches. Mac OS X supports many file systems that use case to differentiate between files. Even on file systems (such as HFS+) that support case insensitivity, there are still times when case may be used to compare filenames. For example, CFBundle and NSBundle consider case when searching bundle directories for named resources.

Try testing your software on volumes (such as UFS) that do distinguish files by case.

Get Only What You Need

Performance is an important consideration when writing code that deals with the file system. It can take millions of cycles of computing time from the time you request data to the time the disk is even ready to begin reading. Plus, different file systems store data differently, so although an HFS+ volume may cache the size of a directories contents, a UFS volume may calculate that value each time it's requested.

As you design your software, make sure that you actually use the data you get and that you actually need the data you use. Reducing file-system overhead can improve your application performance significantly.

For more information on measuring and improving the performance of your file-related code, see *File-System Performance Guidelines*.

Consider the Type of the Destination Volume

A user's files may reside on a network volume, a mobile volume, or on a local volume. If files are on a network volume, accessing them frequently may incur a greater performance penalty. If the network volume is part of a storage area network (such as an Xsan array), you may need to take additional steps to ensure the array performs at peak performance. In particular, you should do the following:

- If you need to optimize your file handling code for maximum performance on specific file systems, use `GetVolParms` or `statfs` to get information about a file system. Use special APIs if they are available.
- Lock only what you need before accessing files and be sure to release those locks as soon as possible.
- Other than the boot volume, don't assume that a volume will always be there. Portable drives may be removed by the user at any time. Sign up for I/O Kit notifications if you need to know if a specific volume is removed from the system.

Line Ending Guidelines

Mac OS X uses the `\n` character by itself to represent the end of a line. Most Mac OS X methods and functions that write out line ending characters write out only this character. Your own code should do the same when writing out content. When reading content, however, your code should be prepared to handle content that contains the `\n`, `\r`, or `\r\n` line endings. Being able to read these other line endings promotes interoperability with other platforms.

Filename Extensions

Some Macintosh software developers react to filename extensions with dismay. As a means for specifying document type and ownership, extensions seem primitive compared to the type and creator codes and the other rich metadata made possible by the multifork HFS and HFS+ volume formats. However, in the Internet age, documents frequently travel around a heterogeneous network. A document may move from a Macintosh to a Linux network server to a Windows computer. Each computer on this path may have a different notion of what constitutes a document type.

Many computer systems define document types solely by well-known filename extensions (such as `.jpg`, `.mp3`, and `.html`). These systems might not know what to do with a file that has no extension and may treat it as an unknown type. Other systems also have little or no knowledge of the HFS+ file system and the metadata it stores. When transferring files, they might strip out this metadata so that it is irretrievably lost.

Hiding Filename Extensions

In order to preserve the Macintosh user experience, Mac OS X provides a way to hide filename extensions on a per-file basis. Each file in the file system has a special flag identifying whether its extension is hidden or shown. Users can set this flag based on whether or not they want the filename extension shown. Users can adjust their Finder preferences to hide or show filename extensions for all files, regardless of the settings for individual files.

Note: The settings for hiding or showing filename extensions affects only the display name of the file. (For more information on display names, see "Display Names" (page 39).) These settings do not physically change the name of the file in the file system.

Supporting Filename Extensions

To ensure platform interoperability and retain the Mac OS X user experience, there are some basic guidelines you should follow to support filename extensions:

- Make sure all of your document types have an associated filename extension.
- Use the display name of a file whenever you display files in your user interface. (See "Getting Display Names in Your Code" (page 40) for details.)
- Save dialogs should allow users to control whether to hide filename extensions.
- Applications should preserve the existing show/hide setting and filename extension when opening or saving a document.
- Applications should add an appropriate extension when saving a new file or when saving an existing file using the Save As command.

- Applications should not append an extension or change the show/hide setting of a file that does not have an extension.
- Applications should display an alert when the user tries to save a file after typing a known, incorrect filename extension.

For Carbon applications, you use Navigation Services to create a Save dialog. If you specify the `kNavPreserveSaveFileExtension` creation option, the dialog preserves and initially hides the filename extension of the default filename. You can also extend your dialog to give the user the choice of showing or hiding filename extensions. If you do, you can use the `NavDialogSetSaveFileExtensionHidden` function to set the current extension visibility and use the `NavDialogGetSaveFileExtensionHidden` function to determine the user's choice. See "Save Dialog Scenarios" (page 36) for more information on potential scenarios when saving files.

If the user does not type a filename extension in your Save dialog, your dialog code should leave the Hide Extension checkbox enabled.

For Cocoa applications, the NSSavePanel class also provides options for hiding and showing filename extensions. If you want the user to have the option of showing or hiding filename extensions, call the `setCanSelectHiddenExtension:` method of NSSavePanel prior to displaying the dialog. This method controls the display of a checkbox that enables the user to toggle the extension visibility. You can use the `isExtensionHidden` method to determine the user's choice.

Carbon applications can also modify the visibility of filename extensions outside of Save dialogs using Launch Services. The `LSSetExtensionHiddenForRef` and `LSSetExtensionHiddenForURL` functions let you set the visibility of filename extensions using `FSRef` and `CFURLRef` types, respectively. Cocoa applications can similarly use the `changeFileAttributes:atPath:` method of NSFileManager.

Save Dialog Scenarios

When saving files, there are several scenarios to consider related to hiding and showing filename extensions:

- What happens when the user types a known, correct extension?
- What happens when the user types a known, incorrect extension?
- What happens when the user types no extension or an unknown extension?

The first scenario is the easiest to deal with. If the user types a known, correct extension, your dialog code should clear the hide extension flag and corresponding dialog controls. When saving the file, you should set the properties of the file to indicate that the extension should be shown.

For situations where the user types a known, incorrect extension, you should display an alert. In your alert, let the user know that an incorrect extension was entered and suggest a correct extension. The dialog box should then let the user cancel the operation, change the extension, or save the filename with both extensions, that is, with the correct extension appended to the user-entered text.

For situations where the user types no extension or an unknown extension, add an appropriate extension to the filename and mark it as hidden. For example, if the user saves a text document as `MyDocument.old`, you would create a file named "MyDocument.old.txt" and set the file attributes to indicate the extension is hidden. Thus, the user continues to see the name `MyDocument.old` in the Finder and in Open and Save dialogs.

Type Codes and Filename Extensions

In addition to filename extensions, applications should also set a file type and optionally a creator type for any files they create. Although these type codes are not strictly necessary, they do ensure interoperability with applications in the Classic environment. If a given file already has a file type or creator type, you should preserve that information.

Applications may set a creator type for documents they create. Doing so creates a tight binding between the document and the application that created it. Applications should not quietly change the creator type for documents that already have one. If you want to change the creator type of documents your application opens, your application should post a Save dialog when the user saves the file. This gives the user an opportunity to rename the file as needed. You might need to do this if editing a document in your application changes the type of the file. In this case, you should also assign an appropriate filename extension to the file in the Save dialog.

Applications that are not a primary editor for documents of a given type should not set a creator type for those documents. For example, an Internet browser may download and save files of many different types, but that does not mean it owns all of those files.

Display Names

Display names are an aspect of the Mac OS X user experience that all applications should support. A display name is a generated name for a file, directory, or application that is based on the user's current preferences. Display names let each user customize their view of the file system without modifying the file system or affecting the views of other users. Mac OS X currently supports the following display-name customizations:

- Application and directory name localization
- Filename extension hiding

The localization of applications and directories gives users a more complete localization experience than they might previously have had. Applications can have different names depending on the user's current language preferences. Application-defined directories can also be localized to make it possible for users to navigate the file system in their native language. Mac OS X automatically localizes the names of many well-known system directories.

Filename extension hiding provides comfort to Macintosh users who are used to the file-naming conventions of earlier versions of the operating system. Each user can decide whether to show or hide filename extensions. Users sharing the same system still see things their way. See "Filename Extensions" (page 35) for more information.

Display Name Usage Guidelines

Display names should not be confused with the actual names of files, directories, and applications in the file system. Display names are based on file-system names but are modified to reflect the current user's preferences. For example, the home directory of an English-speaking user has a `Pictures` directory. If the user changes the preferred language to German, the directory name is still `Pictures`, but now that directory appears in the Finder as the `Bilder` folder.

You cannot use display names to manipulate actual files and directories in the file system. You use display names only as read-only strings in your application's user interface. Display names should always be placed in non-editable controls or in controls whose data is treated as read-only. For example, you would use a display name in the title bar of a document window, in a read-only text field, or in a menu. You would typically not use display names in an editable text field, especially if the user could modify the text and save the changes.

You should always use display name strings immediately after retrieving them. Display names should not be considered persistent, that is, assume they can change from one call to the next. You should never write the display name of a file to your application preferences or store that name in your internal data structures. If you need to refer to a file, store a copy of the actual file name instead.

Mac OS X uses display names in the Finder and in its Open and Save dialogs. If you are writing a GUI-based application, you should support display names to avoid discrepancies between the files users pick and the names they see in your application. However, if you are writing a command-line application, you should not use display names. Mac OS X does not support display names in the Darwin and Classic environments. Applications in those environments must operate on the actual file-system names.

Getting Display Names in Your Code

Because display names are for display only, you should use them only in your application's user interface. For example, if you have a document window open, you would use the display name for the title bar of the window. You should also use display names in other places in your windows where you show filenames, and getting a display name should always be the last thing done before setting the name in a corresponding text field or label.

Carbon application developers can get the display name for a file from Launch Services. The `LSCopyDisplayNameForRef` and `LSCopyDisplayNameForURL` functions return the display name for `FSRef` and `CFURLRef` types, respectively. For information about these functions, see *Launch Services Reference*.

Cocoa application developers can get the display name of a file using the `displayNameAtPath:` method of `NSFileManager`.

Important: Users should not be able to modify display names in your interface. If you need to prompt the user for a filename, use the Open and Save dialogs available through Navigation Services and Cocoa.

Specifying a Localized Application Name

The display name of an application can be localized for the current user. To provide localized versions of your application's display name, you use the existing bundle localization mechanism.

The `Resources` directory of an application bundle can contain multiple `.lproj` subdirectories, each containing the localized resources for one language. One of the files you can put in these language subdirectories is a `InfoPlist.strings` file, which stores localized values for some information property list keys. To specify a localized name for your application, include the `CFBundleDisplayName` key in this file and set the value to the localized name of the application.

For display names, Mac OS X prefers user-customized names over any names contained in the application bundle. If the user changes the name of an application, that name is reflected in the file system. If the user-customized name doesn't match the value of the `CFBundleDisplayName` key in the application's information property list file (`Info.plist`), the system displays the user-customized name regardless of the current language settings. However, if the values do match, Mac OS X uses the localized names stored in the application.

Note: If you support localized display names in your application, you should also include the `LSHasLocalizedDisplayName` key in your information property list. Inclusion of this key improves performance associated with displaying localized application names.

For more information about application bundles and their configuration, see *Bundle Programming Guide*.

Specifying Localized Directory Names

If your application installs any custom support directories, you can provide localized versions of those directory names. A localized directory name shows up as the display name of the directory. Your code must still use the original directory name (not the display name) when accessing the directory's contents.

Providing localized names for directories is not required and should be done only for directories whose names you know in advance. It should not be done for any user-specified directories.

Note: Localized directory names appear only if the "Show all file extensions" option is not selected in the Finder preferences. Localized names do not appear until the next time the user logs in.

To provide a localized display name for a directory, do the following:

1. Add the extension `.localized` to the directory name.
2. From the Terminal application, create a subdirectory inside the directory called `.localized`.
3. Inside the `.localized` subdirectory, put one or more strings files corresponding to the localizations you support.

Each strings file is a Unicode text file. The name of the file is the appropriate two-letter language code followed by the `.strings` extension. For example, a localized Release Notes directory with English, Japanese, and German localizations would have the following directory structure:

```
Release Notes.localized/
    .localized/
        en.strings
        de.strings
        ja.strings
```

Inside each strings file, include a single string entry to map the nonlocalized directory name to the localized name. When specifying the original directory name, do not include the `.localized` extension you just added. For example, to map the name "Release Notes" to a localized directory name, each strings file would have an entry similar to the following:

```
"Release Notes" = "Localized name";
```

For information on creating a strings file, see "Extracting Localizable Strings From Your Code" in *Internationalization Programming Topics*.

Note: System-defined directories, such as `/System`, `/Library`, and the default directories in each user's home directory, use a localization scheme different from the one described here. For these directories, the presence of an empty file with the name `.localized` causes the system to display the directory with a localized name. Do not delete the `.localized` file from any these directories.

BSD Permissions and Ownership

At a fundamental level, Mac OS X is a BSD system. A part of this underpinning is the way BSD implements ownership of, and permissions for, files and folders in the file system. This model, in turn, controls which users can read, write, rename, and execute files, and which users can copy and move files to and from folders. Although the file ownership model is conventionally associated with UFS or similar file systems, Mac OS X extends it to all supported file systems, including Mac OS Standard (HFS) and Mac OS Extended (HFS+).

The following sections describe the basic file ownership model, pointing out areas where Mac OS X differs. These sections also contain a discussion of how root and admin user accounts affect file management.

Overview of BSD Permissions

For each folder and file in the file system, BSD has three categories of users: owner, group, and other. For each of these types of user, three specific permissions affect access to the file or folder: read, write, and execute. If a user does not have read permissions in any of the categories for a file, the user cannot read the file. Similarly, if a user does not have execute permissions for an application, the user cannot run the application.

- The **owner** of a folder or file is generally the user who created it. Owners typically have full privileges (read, write, and execute) for that file or folder. The owner of a file can set the permissions for other classes of users. The root user is the only user that can transfer ownership of files.
- Every user on the system also belongs to one or more groups. A **group** is a named collection of users that have something in common. Every file has a group owner associated with it. The group owner can give additional permissions to a particular set of users. For example, if you had a group containing the engineers for a specific project, you would likely give that entire group write permissions for the source files in the project's code repository.

  The system administrator is responsible for setting up groups.
- Users in the **other** category are just that—everyone who is neither the owner of a file or a member of the group associated with a file. Permissions for this type of user are generally the most restrictive.

Viewing File Permissions

In a Terminal window, if you enter the command `ls -l` for some location in the file system (say, `~steve/Documents`), you get results similar to these:

```
total 3704
drwxrwxrwx  5 steve  staff  264 Oct 24 20:56 General
drwxrwxr-x  5 steve  admin  264 Oct 21 21:47 ProjectDocs
drwxr-xr-x  6 steve  staff  160 Oct 25 12:00 Planning
drwx--x--x  6 steve  staff  160 Oct 21 15:22 Private
```

```
-rwxrwxrwx  1 steve  staff    0 Oct 23 09:55 picture clipping
[sponge:~/Documents] steve%
```

The results show, among other things, the owner and primary group for a file or folder and the permissions for each type of user. The owner of a file is shown in the third column and the primary group is shown in the fourth. Thus, in the preceding listing, the `General` folder is owned by user `steve` and the group is `staff`.

The first column in a detailed file listing is a set of ten characters. The first character indicates the type of the item. A hyphen indicates an ordinary file, a `d` indicates a folder (directory), and an `l` indicates a symbolic link. The remaining nine characters fall into three implicit groups representing the permissions for the owner, group, other user types. The `r`, `w`, and `x` characters, if present, indicate that read, write, and execute permissions are turned on for the type of user the set of bits applies to.

To give an example of how to read the permissions of a file, the following listing shows the permissions for the `Planning` folder in `~steve/Documents`:

```
drwxr-xr-x  6 steve  staff  160 Oct 25 12:00 Planning
```

In this example, the item is a folder. The owner permissions are `rwx`, so the owner view the contents of the directory, can copy files and folders to this directory, can make it his or her current working directory (through the `cd` command), and can execute any program in it. The permissions for both the `staff` group and other users are both `r-x`, meaning that those users can read files in the directory and make it their current working directory; however, they cannot write files to the directory or modify or delete files in it.

Changing File Permissions

If you have the appropriate permissions, you can change owner, group, and individual permissions from a Terminal shell using, respectively, the `chown`, `chgrp`, and `chmod` commands. (See the associated man pages for details.) You can also see the same ownership and permissions information for a selected file or folder in the Privileges pane in an Info window in the Finder. If you are the owner of the file or folder, you can also change permissions directly from the Info window.

To change the individual permissions for a file, you must be able to specify the desired permissions as three-digit integer. In this scenario, each read, write, execute triplet is represented as a decimal number between 0 and 7. Table 1 lists the relationships between each number and triplet.

Table 1 File permission mappings

Decimal number	Permission	English translation
0	---	No permissions
1	--x	Execute only
2	-w-	Write only
3	-wx	Write and execute
4	r--	Read only
5	r-x	Read and execute

Decimal number	Permission	English translation
6	rw-	Read and write
7	rwx	Read, write, and execute

For example, suppose you have a directory whose contents are visible only to the owner. In this case, the directory listing might look something like this:

```
drwx--x--x  6 steve  staff  160 Oct 21 15:22 Private
```

To get the permissions specified here, the owner would have had to execute a command similar to this one:

```
chmod 711 ./Private
```

To break this command down, the `chmod` command changes the individual permissions for the specified item. The number `711` represents the overall permissions for the directory. The first digit (`7`) represents the triplet `rwx`. The second and third digits (both `1`) represent the triplet `--x`.

Permissions for Applications and Documents

When you build an application with Xcode, the build subsystem automatically sets the permissions of the executable file to `-rwxr-xr-x`; this setting enables the owner—that is, the person who installs the application—to execute and write to the application, whereas all others can only execute it. Other IDEs set similar permissions on built executables.

The `-rwxr-xr-x` setting should suffice except in the rare situations where an application requires privileged (root) access. An example would be an application such as a disk repairer that required low-level hardware access through the kernel. In such cases, you could use the `setuid` command to acquire root access for the application. You could then use the features of the NetInfo Kit and System frameworks that allow you to authenticate administrators. For more information on `setuid`, consult the `setuid (2)` and `chmod (1)` man pages.

Although the permission set of the application determines who can launch an application, once it is launched, the application process is owned by the user who launched it. This means the application has the same access rights as the logged-in user and inherits the permissions of that user account. These permissions affect many aspects of the application, such as where the application can save user documents. The application can save files only to locations for which the user has appropriate permissions.

When a Carbon, Cocoa, or Java application saves a document, the respective application environment automatically sets the permissions of the document to (`-rw-r--r--`) by default, giving the owner read and write access but limiting other users to only read-only access. If you want different permissions for the documents created by your application, you must set those permissions using an appropriate file-management interface. For Carbon, use the interfaces of the File Manager. For Cocoa, use the NSFileManager class.

Administrative and Root Accounts

In Mac OS X, administrative and root accounts give you extended access to modifying the file system.

Root and Superuser Access

On BSD systems there is a root user account, which has unlimited access to the folders and files on the system. The root user is often known as the superuser because of the superior access available to that account. For example, a root user can perform the following tasks:

- Read, write, and execute any file
- Copy, move, and rename any file or folder
- Transfer ownership and reset permissions for any user

On BSD systems, root-level access is also available to members of the `wheel` group. Membership in this group confers on users the ability to become the superuser temporarily. To gain this ability, the user enters `su` at the command line and then enters his or her password when prompted for it. Superuser access grants the user temporary root access without requiring the user to logout and log back in as `root`.

In Mac OS X, the root user account is disabled by default to improve system security. The disabling of the root account prevents users from gaining superuser privileges, including through use of the `su` command.

Administrative Accounts

Although the root account is disabled, Mac OS X establishes an admin user account when the system is first installed. The admin user can perform most of the operations normally associated with the root user. The only thing the admin user is prevented from doing is directly adding, modifying, or deleting files in the system domain. However, an administrator can use the Installer or Software Update applications for this purpose.

Any user on the system may have administrative privileges, that is, there is no special need for an account with the name `admin`. Admin users gain their privileges by being added to the `admin` group; non-administrative users belong to the `staff` group. An admin user can grant administrative rights to other users of the system using the Accounts pane of System Preferences.

Enabling the Root User

Although the root user is disabled by default, an administrative user can reenable it and acquire superuser status. To reenable the root user, do the following:

1. Launch the NetInfo Manager application in `/Applications/Utilities`.
2. Choose Security > Authenticate (as needed) to authenticate yourself as an administrative user. (A domain window must be open in order to authenticate yourself.)
3. Choose Security > Enable Root User. (This menu item is enabled only if you are an authenticated member of the local `admin` group.)

The root user password is blank by default. When you enable the account, you are prompted for a root password automatically. You should always provide a root password for security reasons.

Warning: Do not enable the root user account unless circumstances absolutely require it. Misuse of the root account may lead to data loss or damage to your Mac OS X installation.

After you've completed the task requiring root access, you should relinquish superuser privileges immediately by choosing Security > Disable Root User in the NetInfo Manager application.

Access Control Lists

Mac OS X v10.4 introduced support for access control lists (ACLs)—a more fine-grained approach for implementing file and directory permissions. ACLs supplement the existing BSD permissions model in cases where more control is needed over who has access to a file or directory and what actions that person may perform. For example, using ACLs you can assign access rights for a file to multiple users and groups, each with its own distinct permission sets.

In addition to enhancing the ownership model for files and directories, ACLs also offer more fine-grained access to those entities. The basic BSD permissions for a file allow a user or group to read, write or execute that file. With ACLs, you could let a user write data to the file but not delete the file or change its file-system attributes. Similarly, you could let a user read the file attributes but not search the file or read its data.

ACLs are most often used in file server implementations, where fine-grained access to files and directories is crucial. Mac OS X Server v10.4 supports ACLs and the ability to configure them for share points and directories. The client version of Mac OS X v10.4 respects the presence of ACLs but does not currently use them to specify file and directory permissions.

Important: Not all file systems in Mac OS X support ACLs and some may require the system administrator to enable support for ACLs explicitly before they can be used. Currently only the HFS+ file system supports ACLs locally. On the network, ACLs are supported by both AFP and SMB/CIFS.

If you are writing an application that interacts with the file system directly, you should check for the existence of ACLs and respect their presence. For example, developers of file backup software must remember to save ACL data along with the corresponding files and directories and be able to restore that data later.

For detailed information about ACL support in Mac OS X, see *Security Overview* . For more information on how to read and write ACL information in your own code, see the `acl` man page.

File System Comparisons

There are many significant differences between the two major file systems on Mac OS X: HFS+ and UFS. In many cases, these differences have some bearing on programs developed for Mac OS X. The following list summarizes the major differences between these file systems (many of these statements apply to HFS as well as HFS+):

- **Case sensitivity**. UFS is sensitive to case; although HFS+ is case-insensitive, it is case-preserving.
- **Multiple forks**. HFS+ supports multiple forks (and additional metadata) whereas UFS supports only a single fork. (Carbon simulates multiple forks on file systems that do not support them, such as UFS.)
- **Path separators**. HFS+ uses colons as path separators whereas UFS follows the convention of forward slashes. The system translates between these separators.
- **Modification dates**. HFS+ supports both creation and modification dates as file metadata; UFS supports modification dates but not creation dates. If you copy a file with a command that understands modification dates but not creation dates, the command might reset the modification date as it creates a new file for the copy. Because of this behavior, it is possible to have a file with a creation date later than its modification date.
- **Sparse files and zero filling**. UFS supports sparse files, which are a way for the file system to store the data in files without storing unused space allocated for those files. HFS+ does not support sparse files and, in fact, zero-fills all bytes allocated for a file until end-of-file.
- **Lightweight references to file-system items**. See "Aliases and Symbolic Links" (page 53).

In addition, the interfaces historically associated with each file system sometimes have different behaviors. For example, a program using BSD (or BSD-derived) interfaces can delete a file that is open; on the other hand, a Carbon program can delete only a file that is closed.

Table 1 provides a comparative summary of features in the UFS and HFS+ file systems.

Table 1 Feature comparison

Feature	HFS+	UFS
Case sensitive	No	Yes
Supports multiple file forks	Yes	No
Path separator character	":"	"/"
Supports modification dates	Yes	Yes
Supports creation dates	Yes	No
Supports sparse files	No	Yes
Supports zero-filling of files	Yes	No

Feature	HFS+	UFS
Supports aliases	Yes	No
Supports symbolic links	Yes	Yes
Supports ACLs	Yes	No

Aliases and Symbolic Links

Aliases and symbolic links are lightweight references to files and folders. Aliases are associated with Mac OS Standard (HFS) and Mac OS Extended (HFS+) volume formats. Symbolic links are a feature of HFS+ and UFS file systems. Both aliases and symbolic links allow multiple references to files and folders without requiring multiple copies of these items. Prior to Mac OS X v10.2, aliases and symbolic links behaved very differently when a referenced file or folder moved or changed.

Alias Semantics

On HFS and HFS+ file systems, each file and folder has a unique, persistent identity. Aliases use this identity along with pathname information to find files and folders on the same volume.

In versions of Mac OS X before 10.2, aliases located a file or folder using its unique identity first and its pathname second. Beginning with Mac OS X 10.2, aliases reversed this search order by using the pathname first and unique identity second. This means that if you move a file and replace it with an identically named file, aliases to the original file now point to the new file. Similarly, if you move a file on the same volume (without replacing it), aliases use the unique identity information to locate the file.

When a file or folder moves, the alias may update either its path information or unique identity information to account for the change. If a file moves somewhere on the same volume, the alias updates its internal record with the new path information for the file. Similarly, if the original file is replaced by a file with the same name, but a different unique identity, the alias updates its internal record with the unique identity of the new file.

Symbolic Link Semantics

Because aliases use a file system path to resolve a file's location initially, they now offer a similar behavior to symbolic links. Symbolic links rely exclusively on path information to locate a file. If you move a file somewhere on the same volume without replacing it, symbolic links to the file break while aliases do not. The only way to fix a symbolic link is to delete it and create a new one.

Avoiding Broken Aliases

The Finder and other system applications now use aliases with this pathname-first behavior. However, applications can still resolve aliases by unique identity first using the methods of the Alias Manager.

If your application supports versions of Mac OS X prior to Mac OS X v10.2, you should follow certain guidelines when modifying files. First, when editing a file, modify the existing file. Second, if you need to replace a file transparently with a new version, use `FSExchangeObjects` to swap the new file for the old one. The NSDocument class already uses similar techniques to update the document file, thus maintaining aliases whenever possible.

Document Revision History

This table describes the changes to *File System Overview*.

Date	Notes
2009-08-14	Added links to Cocoa Core Comptencies.
2008-07-11	Revised information about initial file permissions.
2006-06-28	Updated line ending and where to put application files guidelines. Updated guidance for what goes in Library directories.
	Removed Sherlock plug-ins directory from list of /Library directories.
2006-04-04	Clarified display name information and added guidelines related to line-ending characters.
2005-07-07	Updated the developer directory information. Incorporated minor fixes reported by developers.
	Added an overview of access control lists.
	Changed title from "The Mac OS X File System".
2004-08-31	Revised display name guidelines to reflect the fact that localization of all bundles is not supported.
	Added guidelines on where to put additional application files.
	Added guidelines on case sensitivity.
	Added initial guidelines relating to files on network and mobile volumes.
2004-03-26	Removed information about the Finder and document bindings. That information is now covered in *Launch Services Programming Guide*.
	Added information about sort ordering rules for Finder windows.
	Added information about the rules used by the Finder to determine the name and icon for files and directories.
	Fixed minor bugs.
2004-02-12	Fixed a minor bug related to the comparison of HFS+ and UFS features.
2003-12-09	First version of *The Mac OS X File System*. The information in this document replaces information about the file system that previously appeared in *System Overview*.