Chapter 7

APPLICATIONS AND FILE TYPES

CHAPTER 7 APPLICATIONS AND FILE TYPES

Goal

To explore the various ways that applications “own” specific file types and reflect this in their design and their UI.

Prerequisites

Familiarity with NSApplication, NSWindow, multi-document applications and a typical hierarchical file system.

Objectives

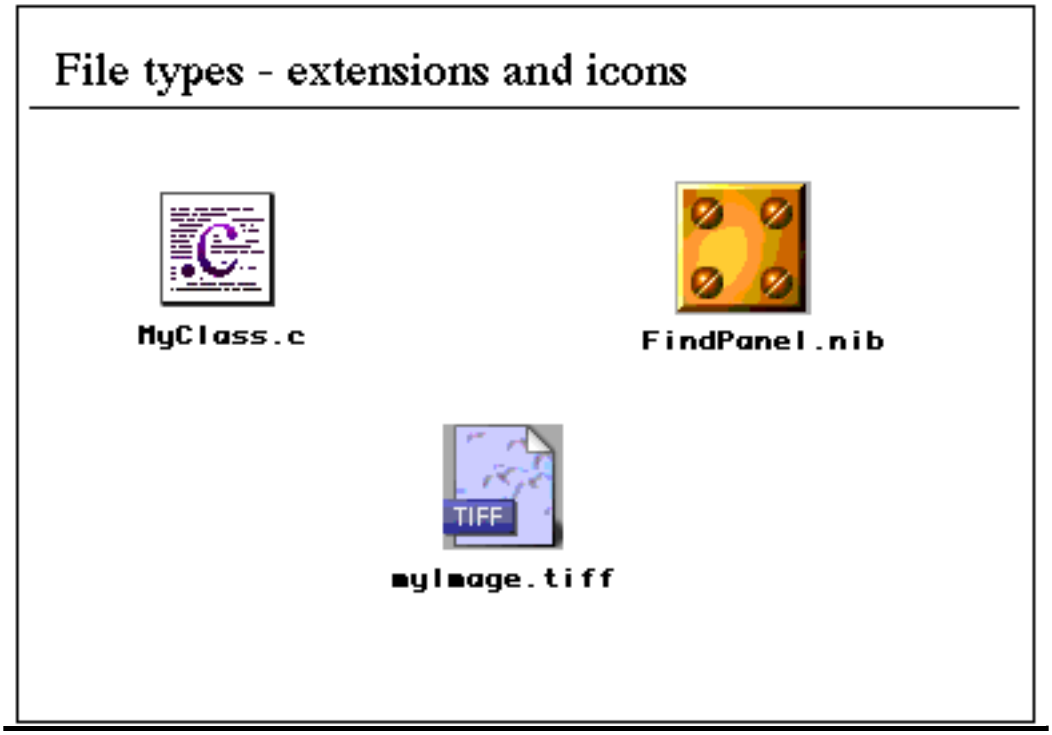
At the end of this section, you will be able to:

- Configure your application for automatic launching via the file icon of an owned file type
- Reflect your application’s file type in your handling of Open and Save panels, window titles and miniwindow icons.
- Effectively manipulate file system objects and pathnames using NSFileManage, NSWorkspace and various NSUtilities.

Reading

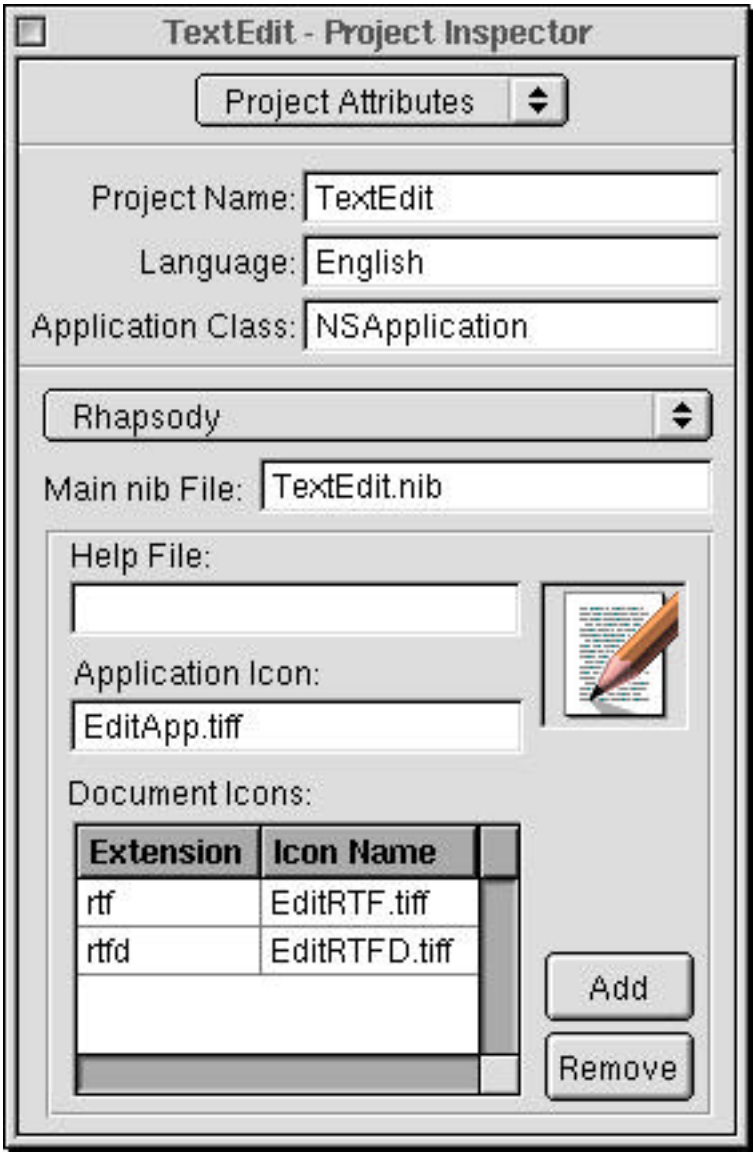
NSWorkspace class reference in the Foundation

NSFileManager class reference in the Foundation



File types—extensions and icons

Files usually have associated types indicated by their filename extension. A file's type is graphically represented by a corresponding icon. Functionally, the file type describes its contents and determines which applications are capable of processing it. A primary application is usually bound to the file type so that when the file is opened from a file browser or desktop icon, the application is automatically launched to display it.



Applications can “claim” file types

Applications can claim any file types that they work with. Project Builder provides a registry table that includes the file type extension and its corresponding icon.

Yours may be one of many applications that can claim to work with a given file type. This does not imply that your application will automatically launch when a file of this type is activated. It still needs to be registered with the underlying platform and further depends on the user’s preferences.

Associating the application with its file type

Different platforms provide different ways of associating the file type with your application for automatic startup from a file browser or desktop.

For YellowBox on Windows 95:

» Using the Associate command from the File menu of the File Manager, create a new type and supply the required information

For YellowBox on Windows NT:

» Using the View command from the Options menu of the Explorer, create a new type from the File Types tab and supply the required information

For YellowBox on Mach:

» Build the install target from Project Builder with an install path that is one of the standard folders searched for applications:

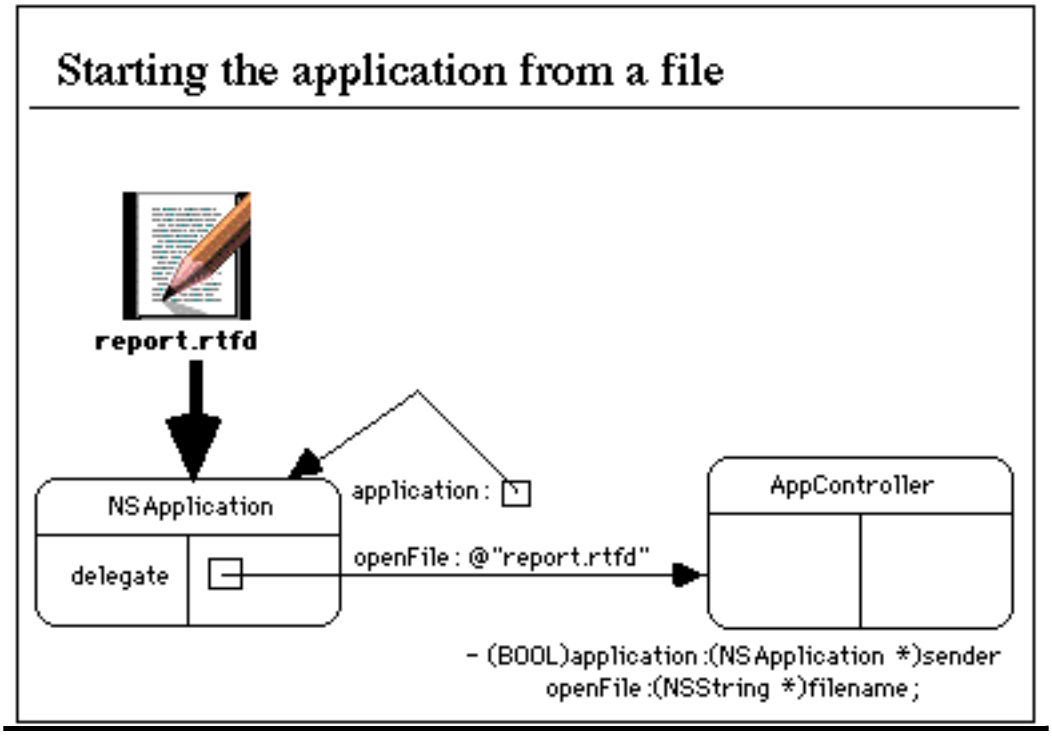
/NextApps

/NextDeveloper/Apps

/LocalApps

$HOME/Apps

» Make the workspace update its viewers

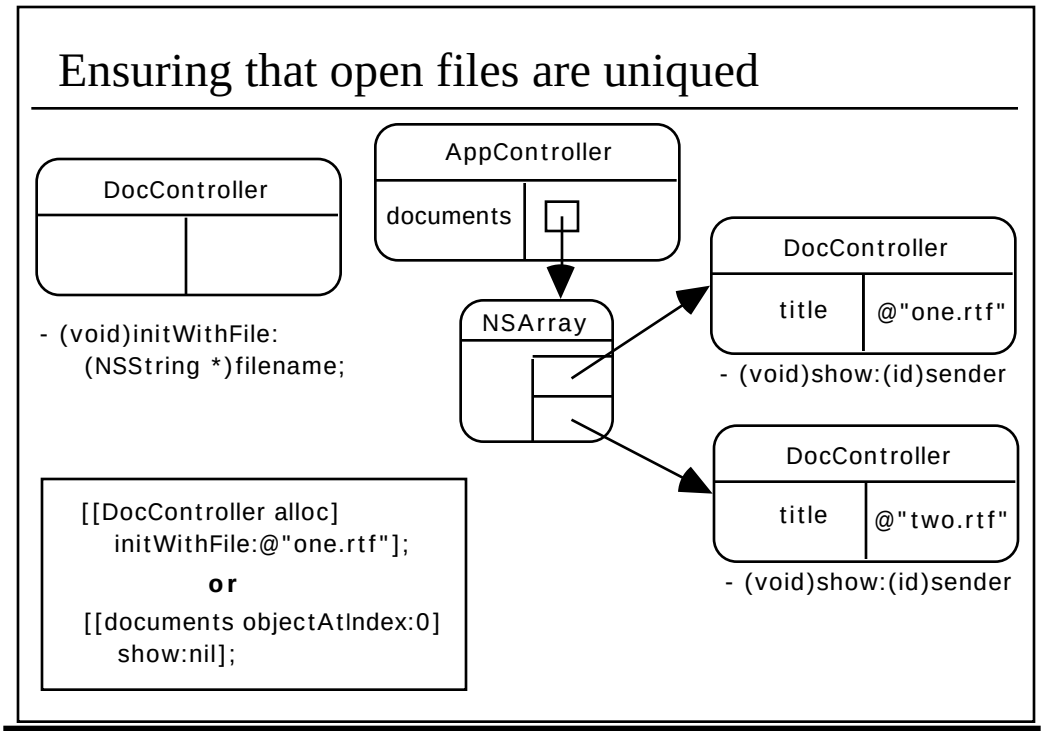


Starting the application from a file

When automatically started on behalf of the activated file icon, NSApplication sends a message to its delegate, your application controller, passing the corresponding full pathname.

Like an open operation triggered from the main menu, your application should allocate a new document controller with the filename and display the window. The application may have been launched for the occasion or it may already be running. This might be the first or the tenth document the application has opened.

If for any reason your application cannot open and display the file, your delegate should return NO.



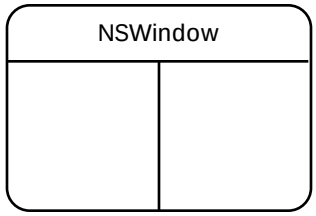
Ensuring that open files are uniqued

What if that file was already opened and is currently displayed in a window managed by an existing document controller instance? Your application should not normally open the file twice, each with a different window. The files will get out of sync. A user might become confused. Your application controller should first walk through it list of any open documents searching for a match. If found, simply message the document controller to make its window key. If not, open the file for the first time with a new document controller instance.

This process of uniquing applies to an open request from the menu and the open panel. A user could easily try to open the same file twice. Both types of open should use the same logic and ideally the same code.

```
- (BOOL)application:(NSApplication *)sender openFile:(NSString
*)file
{
   NSEnumerator *enumerator = [[NSApp windows] objectEnumerator];
   NSWindow *window;
   while (window = [enumerator nextObject]) {
      if ([[window representedFilename] isEqualToString: file]) {
         // already open
         [window makeKeyAndOrderFront: nil]; return YES;
      }
   }
   // new file. Alloc/init DocController and show: . . .
}
```

Window titles and miniaturized window icons




```
- (void)setRepresentedFileName:(NSString *)astring;
- (void)setTitleWithRepresentedFileName:(NSString *)filename;
- (void)setMiniWindowImage:(NSImage *)image;
- (void)setMiniWindowTitle:(NSString *)title;
```

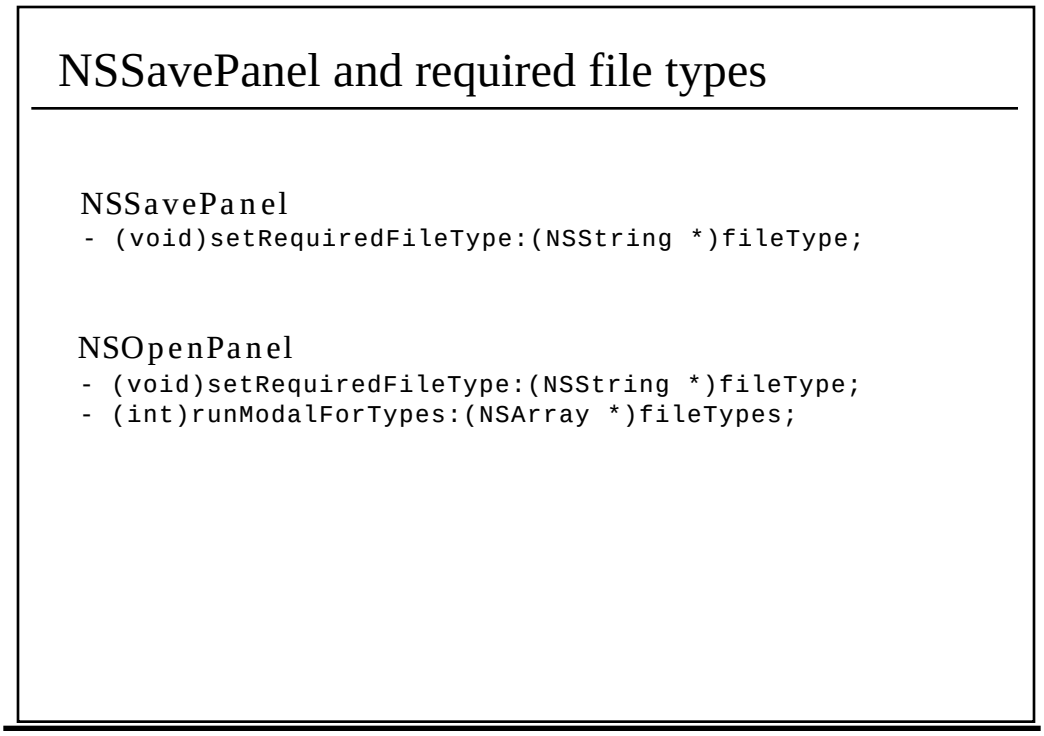
Window titles and miniaturized window icons

Your document window reflects its file type in a couple possible ways:

- RepresentedFileName
- Title

When your window is miniaturized on some platforms, it takes on an alternate appearance that also provides for visual representation of the file. The object is called a mini window and NSWindow has the following related attributes that you can configure:

- MiniWindowImage
- MiniWindowTitle



NSSavePanel and required file types

As mentioned before, NSOpenPanel and NSSavePanel have options for restricting the file types that they will allow.

With NSSavePanel, a required file type's extension will automatically be appended to a filename the user types if it is not already included.

In both NSSavePanel and NSOpenPanel, files with types not matching the required type will be filtered out of view. When opening, it is possible to have more than one possible required type.

Handling file pathnames with NSPathUtilities

NSString Extensions
- path from components, components from path
- appending and extracting path components, file extensions
- extracting and deleting last path components

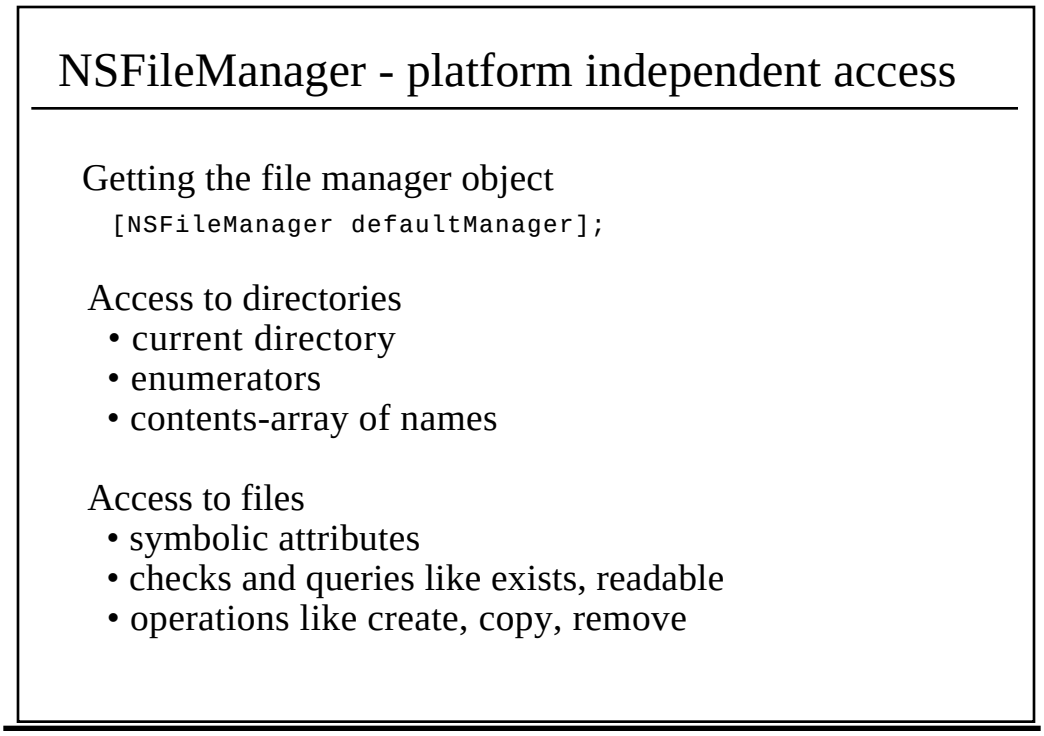
Useful Functions
- NSUserName, NSFullUserName
- NSHomeDirectory, NSUserHomeDirectory
- NSCurrentDirectory
- NSOpenStepRootDirectory
- NSTemporaryDirectory

Handling file pathnames with NSPathUtilities

Because your application is file based, it needs platform independent API for handling file and path names and, in a related fashion, information about the current user, home directory, temporary directory, root directory etc.

There are a number of convenient path utilities. Note that the informational API is a set of functions whereas path manipulation API is a set of method extensions to NSString, the natural place for storing and manipulating file and path names.

When doing any kind of manual handling of pathnames for a multi-platform application, remember that different platforms use different path separators. This API encapsulates this but your methods may need to take account of this fact. Be aware that NSBrowser uses a **pathSeparator** which, if used to browse file hierarchies, may not be what you actually want in your pathname strings.



NSFileManager—platform independent access

More interesting than just pathname parsing is a platform independent way to manage files and directories. NSFileManager is an object for doing just that. Use it to check the validity and attributes of files and directories. It can perform basic file system operations like create, copy, and remove. It provides a platform independent way for enumerating over a directory so you can obtain a list of its constituents without knowing the filesystem type or directory format.

NSWorkSpace - platform independent operations

Getting the workspace object

```
[NSWorkSpace sharedInstance];
```

Services

- opening files - via implied or explicit applications
- selecting and locating files in the File Viewer
- displaying system-specific file information
- file operations like compress, encrypt
- access to a file's icon
- launching applications
- workspace related notifications

NSWorkspace—platform independent operations

NSWorkspace is an object-oriented interface to the file system itself. In addition to file operations, you can programmatically handle the file along with its associated application. You can even ask that a given file be selected in the file viewer.

You can obtain the shared NSWorkspace instance with the **sharedInstance** factory message.

Important ideas from this section

» Files are typed and associated with a:

  Filename extension

  File icon

  Application that owns that file type

» Each OS platform has a unique system for binding an application to the file type(s) that it claims.

» NSOpenPanel and NSSavePanel can automatically support and enforce an application's file type(s).

» Multi-document applications should "unique" each open file ensuring that the same file is not open multiple times in different windows.

» Platform independent file and path handling is available via NSFileManager and NSWorkspace objects and the functions and NSString extensions available from NSPathUtilities.

Classes featured in this section

» NSFileManager

» NSWorkspace

» NSPathUtilities.h (functions)

» NSApplication

» NSWindow

» NSOpenPanel, NSSavePanel