Bundles

Author: Mark Szymczyk
Last Update: May 9, 2007

This article provides an introduction to Mac OS X application bundles, showing you how to create an application bundle, add files and folders to the bundle, and retrieve files from the bundle.

What Are Bundles and Packages?

A *bundle* is a directory. That's all it is: a directory. A *package* wraps a bundle so the bundle looks like a single file instead of a directory. The most common bundle is an application bundle, which contains an executable file and the external files the application uses. A game's external files include graphics files, sound files, and data files like levels.

Most Mac OS X GUI applications are application bundles. Application bundles are packages, which provides a major benefit for software developers. Packages appear to the user as a single file so he or she can't accidentally delete or rename the external files your application needs to run.

Application Bundle Organization

Because most Mac OS X applications are application bundles, there are many examples for you to look at. To examine an application bundle, select an application from the Finder and control-click. A contextual menu will open. Choose Show Package Contents.

When you examine an application bundle, the top folder is named Contents, which holds the contents of the bundle. What lies underneath the Contents folder depends on the application, but there will be a minimum of four items.

- A MacOS folder that contains the executable file.
- A Resources folder that holds the external files. Your application will store its files here.
- A `PkgInfo` file, which tells you the type of bundle it is and the bundle's creator code. The creator code links your application to its document files so that your application will open when the user double-clicks a document file in the Finder.
- An `info.plist` file that provides various pieces of information about the bundle, such as its version number and the name of the executable file.

If you look inside an application's Resources folder, you can see what spoken languages the application supports. There will be a folder with the extension `.lproj` for each language the application supports.

More complicated applications pack more folders into their bundles. Xcode has a Plugins folder that contains Xcode's plugins. iTunes has a Frameworks folder that contains the code frameworks iTunes uses.

Creating Bundles

The first question that comes to mind when talking about application bundles is how do you create the bundle. The good news is you don't have to do any work to create an application bundle if you use Xcode. Xcode's Cocoa and Carbon application projects are set up so all you have to do is build the project to create the bundle.

Adding Files to a Bundle

Now that you know how to create an application bundle, a second question comes to mind. How do you get your files into the application bundle?

The Groups and Files list runs along the left side of Xcode's project window. You should see a Resources folder in the Groups and Files list. Files in the Resources folder will get copied to the Resources folder in the application bundle when you build the project. To add files to the application bundle, you must add them to the Resources folder in your project and build the project.

1) Select the Resources folder from the Groups and Files list.
2) Control-click to open a contextual menu.
3) Choose Add > Existing Files to add the files to your project.

If you want to make sure the files you added will get copied to the application bundle, check to see if they are in the Copy Bundle Resources build phase for your project's target.

1) Click the Targets disclosure triangle in the Groups and Files list. You will see a list of your project's targets.
2) Click the disclosure triangle next to a target. You will see a list of the target's build phases.
3) You should see a Copy Bundle Resources build phase. Click the disclosure triangle next to it.
4) Make sure the files you added are in the Copy Bundle Resources build phase. If they are not, drag them from the Groups and Files list to the Copy Bundle Resources build phase.

Adding a Folder to a Bundle

If your application uses lots of files, organizing them into folders makes sense. Suppose you have a game with 50 levels, 75 graphics files, and 75 sound files. Throwing all 200 files into the application bundle's Resources folder would be messy. Having separate folders for the levels, graphics files, and sound files makes finding specific files in the bundle easier. How do you add folders to the application bundle?

Adding folders is no different than adding individual files. Add the folder to your project using the steps I detailed last section. There is one thing to keep in mind. When you add a file or a folder to your Xcode project, you will see a sheet like Figure 1.

Xcode has two options for adding folders to a project: recursively create groups and creating folder references. Create a folder reference when you want to copy the folder to the application bundle.

Reading Files from a Bundle

At this point you can create application bundles and add files and folders to the bundle. The last major task is to read the files from the bundle. This involves three steps.

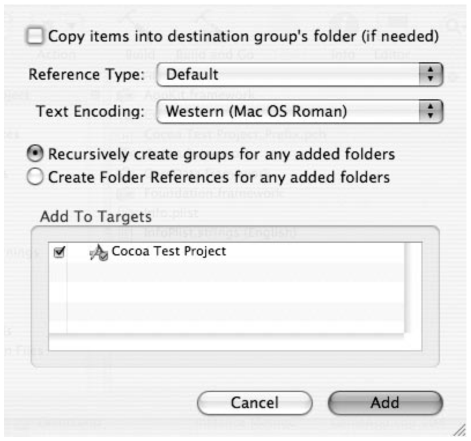
1) Get the application bundle.
2) Find the file in the bundle.
3) Open the file so you can read data from it.

I'm going to use the Core Foundation framework functions. If you're writing a Cocoa application, look at Cocoa's `NSBundle` class, which performs the same tasks as Core Foundation's bundle functions.

Getting the Application Bundle

Call the function `CFBundleGetMainBundle()`, which returns the main bundle in the application.

```
CFBundleRef gameBundle = CFBundleGetMainBundle();
```



[ ] Copy items into destination group's folder (if needed)

Reference Type: Default

Text Encoding: Western (Mac OS Roman)

(●) Recursively create groups for any added folders

( ) Create Folder References for any added folders

Add To Targets

[✓] Cocoa Test Project

Figure 1

Xcode's dialog box to add a file or folder to a project.

Finding a File in the Application Bundle

After getting the application's main bundle, you must find the file in the bundle. Call the function `CFBundleCopyResourceURL()`, which returns the file's location in the bundle. This function takes four arguments.

- The bundle you retrieved by calling `CFBundleGetMainBundle()`.
- The file name.
- The file extension.
- A subdirectory in the bundle where the operating system should look for the file. Passing NULL tells the operating system to search the entire bundle. Unless you have a huge bundle with hundreds of files, passing NULL should be sufficient.

```
CFBundleRef gameBundle;
CFStringRef filename;
CFStringRef fileExtension;
CFStringRef subdirectory;
CFURLRef fileLocation;

fileLocation = CFBundleCopyResourceURL(gameBundle, filename,
      fileExtension, subdirectory);
```

Suppose you want to load a file named `Background.png`. There are two ways you can call `CFBundleCopyResourceURL()`. First, you can use `Background` as the file name and `png` as the extension. Second, you can use `Background.png` as the file name and NULL as the extension.

Opening the File

There are several ways to open files on Mac OS X. I'm going to focus on Apple's technologies in Carbon to open files.

`CFBundleCopyResourceURL()` returns a variable of type `CFURLRef`, which you can use to open the file and read data from the file. The code to open the file depends on the type of file. For graphics and sound files, you would use QuickTime to open and read the file. I provide detailed instructions on using QuickTime in my QuickTime audio and texture loading articles.

To open data files you should call the function `CFURLGetFSRef()`. This function gives you a file system reference, a `FSRef`, that you pass to either a File Manager or Resource Manager function to open the file.

```
CFURLRef fileLocation;
FSRef fileToRead;
Boolean openedSuccessfully;

openedSuccessfully = CFURLGetFSRef(fileLocation, &fileToRead);
```

Call the File Manager function `FSOpenFork()` to open a data file.

```
OSErr error;
HFSUniStr255 forkName;
short refNum;

error = FSGetDataForkName(&forkName);

error = FSOpenFork(&fileToRead, forkName.length, forkName.unicode,
      fsRdPerm, &refNum);
```

Call the Resource Manager function `FSOpenResFile()` to open a resource file.

```
short refNum;

refNum = FSOpenResFile(&fileToRead, fsRdPerm);
```

Conclusion

Normally I include source code with my articles that provides the simplest possible example of the material I covered in the article. But it is hard to write an application that just loads a file from an application bundle. However, I have three code samples from previous articles that use application bundles.

- The code for my tiling article is the most comprehensive sample. It has code to load a texture and load a data file using the Resource Manager. The functions `GameLevel::ReadLevelData()` and `GameTexture::ReadFromFile()` are the functions of interest.
- The code for my QuickTime audio article is the simplest sample, providing an example of loading an audio file.
- The code for my texture loading article provides a simple example of loading a graphics file from an application bundle.

Sorry, I currently don't have any Cocoa or File Manager examples.

For more information on application bundles, refer to Apple's Bundle Programming Guide.