Creatingdynamicallyloadablebundles;¬Creating dynamically loadable bundles

1. **Create a project or subproject of type Loadable Bundle.**
2. **In the Project Attributes Inspector, enter the name of the bundle's controller class in the Principal Class field.**
3. **Add classes, interfaces, and resources as you would for any other project.**
4. **Create a class outside of the bundle that loads the bundle.**

The other tasks in this chapter show how to separate the interface into multiple nib files so that infrequently used parts of the interface are loaded only if needed. You can do the same thing with the application's executable codeÐdivide it into wholly contained pieces that are loaded only if needed.

To separate out a portion of executable code, you create a loadable bundle. *Loadable bundles* are file packages that can contain executable code, resources, and nib files. The main difference between a loadable bundle and an application is that an application has a **main()** function and an NSApplication instance. Loadable bundles typically don't have **main()** functions.

The key attribute of a bundle project is its principal class. The principal class is essentially the controller class for the bundle. You must specify the principal class in the bundle project's attributes inspector.

_CreatingBundles1.eps ¬

Loading the Bundle Programmatically

Because a loadable bundle doesn't have a **main()** function, you must write code that loads the bundle and starts executing. The following method does just that:

```
- (void)showPreferences:(id)sender
```

```
{
    Class bundleClass;
    id newInstance;
    NSBundle *bundleToLoad =
        [NSBundle bundleWithPath:[[NSBundle mainBundle]
        pathForResource:@"Preferences" ofType:@"bundle"]];

    if (bundleClass = [bundleToLoad principalClass]) {
        newInstance = [[bundleClass alloc] init];
        [newInstance loadPanel];
     }
}
```

Notes on the code: This method loads the bundle into memory. It starts by telling the NSBundle class where to find the bundle. In this case, the bundle is in a subproject, which means it resides in the **Resources** directory inside the main bundle, so sending **pathForResource:ofType:** to the main bundle returns the correct location. The **principalClass** method finds out the bundle's principal class, loading the bundle if necessary. Once the principal class is known, this method creates an instance of that class and sends it a message. (In this example, the message is to load a panel defined in the bundle.)

Adding a Nib File to a Bundle Project

Because loadable bundles can contain nib files, it's often convenient to create a bundle containing an infrequently used part of the interface and the code that controls it. For example, you could put the Preferences panel and an object that controls it in a separate bundle project.

To create a nib file in a bundle project, use the Interface Builder command Document**arrow.eps ¬** New Module **arrow.eps ¬** New Empty. Add the bundle's principal class to the nib file and set the File's Owner to be that class.

_CreatingBundles2.eps ¬

You'll need to connect this part of the interface to the main nib file. To do so, have the application's controller object load the bundle in response to an action message. In the example shown here, the bundle is loaded when the user chooses the Preferences command. (**showPreferences:** method is shown above.)

_CreatingBundles3.eps ¬

Although Loadable Bundle projects can be stand-alone projects, they are often created as subprojects of an application or framework. For more on subprojects, see ªGrouping projectsº in Chapter€1. ;../../01_Starting/01_CreatingManaging/GroupingProjects.rtfd;;¬

ªLoading nib files dynamically: an info panelº in this chapter walks through the major steps of creating a nib file from the New Module menu and assigning the File's Owner. ;LoadingNibFilesDynamically.rtfd;;¬

Related Concept: ;DynamicLoadingConcepts.rtfd;linkMarkername InsidetheNSBundleClass;, Inside the NSBundle Class