Copyright ©1996 by NeXT Software, Inc. All Rights Reserved.

MultipleNibFiles:GoodThingsinSmallPieces;¬Multiple Nib Files: Good Things in Small Pieces

Why have multiple nib files in an application? Why not put everything in the main nib file? The answer is simple: Because multiple nib files enhance the performance of the application.

You can strategically store the resources of an application (including pieces of the interface) in several nib files. When the application needs a resource, it loads the nib file containing it. Because you don't have to load the entire application into memory at once, the program is more efficient. The application also will launch faster.

When many sophisticated applications start up, they load only a minimum of resources in the main nib fileÐthe main menu andperhapsa window. They display other windows (and load other nib files) only when users request it or when conditions warrant it.

Types of Auxiliary Nib Files

Nib files other than an application's main nib file are sometimes called *auxiliary nib files*. There are two general types of auxiliary nib files: special-use and document.

Special-use nib files contain objects (and other resources) that *might* be used in the normal operation of the application (like a Preferences panel). Document nib files contain objects that represent some repeatable entity, such as a word-processor document. A document nib file functions as a template for documents: it contains the UI objects and other resources needed to make a document. (Creating document nib files is described at length in the book *Discovering OPENSTEP*. ;/NextLibrary/Documentation/NextDev/TasksAndConcepts/DeveloperTutorial/0_Contents.rtfd;;¬)

File's Owner

The key step in creating applications with multiple nib files is assigning the auxiliary nib file's File's Owner. The file's owner object is always external to the nib file it owns. It channels messages between the objects unarchived from the nib file to the other objects in your application.

The global NSApplication object owns the main nib file. Special-use nib files are often owned by the application's controller object, which you typically define in the main nib file. A document nib file is typically owned by a separate controller object, a document controller.

The main job of the File's Owner object is to load the auxiliary nib file. To do so, it sends the message **loadNibNamed:owner:** to the NSBundle class object. In the main nib file you define an action method in the controller class and hook that action up to a control in the interface. That action method's implementation sends the **loadNibNamed:owner:** message. In this way, the nib file is loaded only if the user requests it.

Creating Auxiliary Nib Files

To create an auxiliary nib file, you use one of the commands on the New Modules menu (which is under the Documentmenu) in Interface Builder. New Modules gives you several choices of the type of nib file to create:

- New InfoPanel Creates an infopanel.
- New Attention Panel Creates an attention panel.
- New Empty Creates an empty nib file.
- New Palette Creates a static palette.
- New Inspector Creates an inspector panel.

The last two commands (New Palette and New Inspector) are used when creating static palettes. If you're not working on a palette project, you use the New Empty command to create a nib file, unless you're specifically creating an infopanel or an attention panel.

You might have noticed that the New Application command also creates a nib file. This command creates a main nib fileÐone that contains a main menu and is owned by the NSApplication object. However, you usually let Project Builder create the main nib file for you when you create an application project.

28012_TableRule.eps ¬

InsidetheNSBundleClass;¬Inside the NSBundle Class

If you look at the NSBundle class specification in the *Foundation Framework Reference*, you'll notice that

NSBundle can tell you a lot of useful things: where your program's resources are, where its frameworks are, which framework defines a particular class. It can even tell you how your application's interface ought to be localized. Why is it so smart?

Every bundle contains a property list that defines the bundle's attributes. This property list is the real brains behind the NSBundle class; NSBundle is simply reading the property list and returning the information it contains. Project Builder uses the information you specify in Project Builder's Inspector panel to create and update this property list.

The Principal Class

At the very least, the property list contains the name of the bundle's executable. Most property lists (in fact, all of them besides those used for frameworks) must contain one other important piece of information: the principal class's name.

The principal class is the class that performs the main work of the bundle. For applications, the principal class is either NSApplication or a subclass of NSApplication. NSApplication runs the application event loop, during which the custom code you have written for your application is executed.

For Loadable Bundle projects, the principal class is often a controller-style class. It knows about all of the other objects inside of the bundle and can send them messages to have them perform work. If the bundle contains a nib file, the bundle's principal class is often the appropriate choice for the owner of that nib file (just as NSApplication owns the main nib file of an application bundle).

The principal class is important because the NSBundle class uses it to load a bundle into memory. Loading a bundle is typically a two step process. First you create an NSBundle object using the location of the bundle in the file system as input. Then, you send that bundle the message **principalClass**. This method returns the principal class in the bundle. In order to do this, it must read the property list, which in turn means it must load the bundle into memory if that bundle has not already been loaded. Thus, asking an NSBundle for its principal class is the main way you load a bundle into memory. From there, you can create an instance of the principal class and send it a message to have it perform work.

If NSBundle can't find out the name of the principal class from the property list, it assumes that the first class loaded is the principal class. This is determined by the order in which the object files are linked.

Application Property Lists

A simple application project contains two more pieces of information in addition to the executable name and principal class name: the name of the main nib file, and a list of file formats the application can read and write. Most applications also have a line that identifies the application's icon.

Adding Information to the Property List

Your project is not limited to the information that Project Builder stores in the property list. You can use this list to store other information specific to your application. However, because Project Builder maintains this list, you should never update it directly. Instead, create a file named **CustomInfo.plist** and add it to the project under Other Resources. Project Builder looks for such a file and merges it with the other information to create property list. Two reasons that you would create a **CustomInfo.plist** file are to advertise a service that your application performs (on the Services menu of other applications) or to add on-line help to your application.