Unarchiving Interface Objects With Interface Builder Services

Carbon > User Experience

2004-02-17

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

Simultaneously published in the United States and Canada.

Contents

Listings

Introduction to Unarchiving Interface Objects With Interface Builder Services

Interface Builder is an Apple development tool that makes it easy to create application interfaces by dragging objects (buttons, windows, menus, text boxes, controls, and so forth) from a palette. The standard palettes for Carbon—Menu, Text, Controls, Enhanced Controls, and Data Views—provide a variety of ready-made objects.

When you create your application's user interface, Interface Builder archives the elements of the interface as objects. When your application is ready to use the interface objects, you need to call functions from Interface Builder Services to unarchive the objects. If you aren't using Interface Builder to create your application's user interface, then you don't need to use Interface Builder Services.

Interface Builder Services Concepts

Interface Builder Services provides functions that unarchive interface objects from **nib files**. A nib file is an Interface Builder file; it contains a description of one or more objects in a user interface. Before you use the functions in Interface Builder Services, you may find it helpful to know a bit more about Interface Builder, nib files, and strategies for storing interface objects.

Interface Builder

Interface Builder is an Apple development tool you can use to build interfaces for your Carbon applications. It is a WYSIWYG tool that lets you lay out user interfaces in a simple, intuitive manner. You use it along with Project Builder, Apple's main development tool. Interface Builder and Project Builder are available on the Mac OS X Developer CD.

You can find more information about using Interface Builder in the online help provided with it.

Nib Files

A nib file contains the application's interface-based resources. It's an Interface Builder file (the "ib" in "nib" stands for Interface Builder) that contains descriptions of the interface elements in your application. These descriptions use XML (extended markup language), although you'll never see, nor should you try to edit, the XML in a nib file. A nib file can contain user-interface objects, and references to any sounds, and images used in the interface.

A nib file can describe all or part of a user interface. Most applications use two or more nib files, with one of them designated as the main nib file. The main nib file contains the main menu and any windows and panels you want to appear when your application starts up. In addition to the main nib file, you can have one or more nib files that you load whenever you need them. The additional nib files are called **auxiliary nib files**. For example, if your application is a word processor, you might have an auxiliary nib file for a document window. Each time your user creates a new document, you'd use Interface Builder Services to unarchive a document window from the auxiliary nib file.

Strategies for Storing Interface Objects

You can strategically store an application's interface objects in several nib files. When the application needs an interface object, you can load the nib file which contains it. You'll conserve memory and improve program efficiency by following these guidelines:

- Store the main menu and perhaps a window in the main nib file. You should store a window in the main nib file only if the window always opens when the application starts up.

- Store each window or menu (such as an About window) that is likely to be used occasionally in a separate nib file. That way, you can call Interface Builder Services to load the window or menu only when a user requests it or when conditions warrant it.
- If your application uses a repeatable object, such as a word-processor document or a spreadsheet, store it in a document nib file. A **document nib file** is an auxiliary nib file that's used as a template for a document: it contains the user interface objects and other resources needed to make a document.

Interface Builder Services Tasks

The primary tasks for which you'll need to use Interface Builder Services are:

- Unarchiving the menu bar and main window from your application's main nib file
- Unarchiving an object from an auxiliary nib file
- Unarchiving an object from a framework or other bundle

Each of these tasks are described in the following sections.

Unarchiving Objects From the Main Nib File

When your application starts up, you need to call Interface Builder Services functions to open the main nib file and unarchive the interface objects that should be open after start-up. As the "Strategies for Storing Interface Objects" (page 9) listed in "Interface Builder Services Concepts" (page 9) suggest, the main nib file should contain only those items that are essential when your application starts up. In most cases, the main nib file should only contain the menu bar and perhaps a main window.

The steps below outline how to open a main nib file and unarchive the objects in it. Listing 3-1 shows how to implement the steps for a nib file that contains a menu bar and a main window. If your application needs only one of these objects at startup, you can easily modify the sample code.

1. Call the function `CreateNibReference` to create a reference to the main nib file.
2. Unarchive the menu bar from the main nib file by calling the function `SetMenuBarFromNib`. This function also sets the menu bar so users can use the menu bar when your application has started up.
3. If your application has a main window, you call the function `CreateWindowFromNib` to unarchive the main window.
4. After you have unarchived the objects from the main nib file, you must dispose of the nib reference by calling the function `DisposeNibReference`.
5. The function `CreateWindowFromNib` unarchives the window so it's hidden. If you want the window to be visible, you must call the Window Manager function `ShowWindow`.

It is good practice to check for errors each step of the way, as shown in <xRefNoLink>Listing 3-1 (page 11). If the interface can't be created, you should halt the start up process. Without an interface, your application is likely to be useless.

Listing 2-1 Unarchiving the menu bar and main window from the main nib file

```
int main (int argc, char* argv[])
{
```

```
    IBNibRef        nibRef;
    WindowRef       window;
    OSStatus        err;

    // Create a nib reference to a nib file.
    err = CreateNibReference (CFSTR ("main"), &nibRef);
    // Call the macro require_noerr to make sure no errors occurred
    require_noerr (err, CantGetNibRef);

    // Unarchive the menu bar and make it ready to use.
    err = SetMenuBarFromNib (nibRef, CFSTR("MainMenu"));
    require_noerr (err, CantSetMenuBar);

    // Unarchive the main window.
    err = CreateWindowFromNib (nibRef, CFSTR("MainWindow"), &window);
    require_noerr (err, CantCreateWindow);

    // Dispose of the nib reference as soon as you don't need it any more.
    DisposeNibReference (nibRef);
    // Make the unarchived window visible.
    ShowWindow (window);

    // Start the event loop. RunApplicationEventLoop is a
     // Carbon Event Manager function.
    RunApplicationEventLoop ();

    // You'll jump to one of the "Cant" statements only if there's
    // an error.
    CantCreateWindow:
    CantSetMenuBar:
    CantGetNibRef:
    return err;
}
```

Unarchiving an Object From an Auxiliary Nib File

The "Strategies for Storing Interface Objects" (page 9) suggest that you create several nib files to store your application's interface objects. The main nib file should contain at the most, the menu bar and the window that opens (if any) when your application starts up. Document windows, palettes, toolbars, contextual menus, and other interface objects should be stored in separate, auxiliary nib files.

The steps for opening an auxiliary nib file and unarchiving an object from it are similar to those used to open a main nib file:

1. Call the function `CreateNibReference` to create a reference to the auxiliary nib file that contains the object you want to unarchive.

2. Call the appropriate function to unarchive the object from the nib file. To unarchive a window, call the function `CreateWindowFromNib`, to unarchive a menu, call the function `CreateMenuFromNib`.

3. After you have unarchived the object from the auxiliary nib file, you must dispose of the nib reference by calling the function `DisposeNibReference`.

One common use of an auxiliary nib file is to store an object that's used repeatedly in an application, such as a document window. Another use is to store objects that are rarely needed, such as an About window. Listing 3-2 shows how to implement a `MyCreateNewDocument` function that your application would call each time the user creates a new document. Note that similar to Listing 3-1 (page 11), the code shown in Listing 3-2 implements error checking.

Listing 2-2 Unarchiving a document window from an auxiliary nib file

```
WindowRef MyCreateNewDocument (CFStringRef inName)
{
    IBNibRef documentNib;
    OSStatus err;
    WindowRef theWindow;

    // Create a nib reference to an auxiliary nib file with
    // the name document.nib.
    err = CreateNibReference (CFSTR ("document"), &documentNib);
    // Call the macro require_noerr to make sure no errors occurred
    require_noerr (err, CantGetNibRef);

    // Unarchive the document window. Use the name you gave to the
    // window object in the Instances pane in Interface Builder.
    err = CreateWindowFromNib (documentNib, CFSTR("MyDocument"),
                                            &theWindow);
    require_noerr (err, CantCreateWindow);

    // Dispose of the nib reference as soon as you don't need it anymore.
    DisposeNibReference (documentNib);

    // Call the Window Manager function to set the title shown in the
    // window's title bar to the name passed to MyCreateNewDocument.
    err =  SetWindowTitleWithCFString (theWindow, inName);
    // In this example, the window gets returned. Remember, it's been
    // unarchived, but it is still not visible. It won't be visible
    // until you call the Window Manager function ShowWindow.
    return theWindow;

    // You'll jump to one of the "Cant" statements only if there's
    // an error.
    CantCreateWindow:
    CantGetNibRef:
     return NULL;
}
```

Unarchiving an Object From a Bundle

Your application is not limited to using interface objects contained within its own bundle. You can unarchive interface objects from another bundle or framework to which your application has access. For example, you could unarchive a tools palette or other object provided by a plug-in bundle.

The steps for unarchiving an object from a nib file in a framework or other bundle are similar to those used to open an auxiliary nib file. They are listed below. The main difference is that you need to get a reference to the bundle. Then you must call the function `CreateNibReferenceWithCFBundle` instead of `CreateNibReference`.

1. Call the Core Foundation URL Services function `CFURLCreateWithFileSystemPath` and the Core Foundation Bundle Services function `CFBundleCreate` to create a reference to the bundle that contains nib file you want to open. See the Core Foundation reference documentation for more information.
2. Call the function `CreateNibReferenceWithCFBundle` to create a reference to the nib file that contains the object you want to unarchive.
3. Call the appropriate function to unarchive the object from the nib file. To unarchive a window, call the function `CreateWindowFromNib`, to unarchive a menu, call the function `CreateMenuFromNib`.
4. After you have unarchived the object from the nib file, you must dispose of the nib reference by calling the function `DisposeNibReference`.

The function `MyCreateWidgetFromFramework`, shown in Listing 3-3 (page 14), shows how to unarchive a "widget window" from a bundle whose path you pass to the function.

Listing 2-3 Unarchiving a widget window from a nib file in a bundle

```
WindowRef MyCreateWidgetFromBundle (CFStringRef widgetBundlePath
                                    CFStringRef widgetFileName,
                                    CFStringRef  widgetWindowName)
{
    IBNibRef widgetNib;
    OSStatus err;
    WindowRef theWindow;
    CFBundleRef mainBundle;
    CFURLRef    bundleURL;
    CFBundleRef widgetBundle;

    // Look for a resource in the bundle passed to
    // the function MyCreateWidgetFromBundle
    bundleURL = CFURLCreateWithFileSystemPath(
                            kCFAllocatorDefault,
                            widgetBundlePath,
                            kCFURLPOSIXPathStyle,
                            TRUE);
    // Make a bundle instance using the URL Reference
    widgetBundle = CFBundleCreate (kCFAllocatorDefault, bundleURL);

    // Create a nib reference to the nib file.
    err = CreateNibReferenceWithCFBundle (widgetBundle,
                            widgetFileName, &widgetNib);
    // Call the macro require_noerr to make sure no errors occurred
    require_noerr (err, CantGetNibRef);

    // Unarchive the widget window.
    err = CreateWindowFromNib (widgetNib, widgetWindowName, &theWindow);
    require_noerr (err, CantCreateWindow );

    // Dispose of the nib reference as soon as you don't need it anymore.
    DisposeNibReference (widgetNib);
    // Release the Core Foundation objects
    CFRelease (bundleURL);
    CFRelease (widgetBundle);

    // In this example, the window gets returned. Remember, it's been
```

```
    // unarchived, but it is still not visible. It won't be visible
    // until you call the Window Manager function ShowWindow.
    return theWindow;

    // You'll jump to one of the "Cant" statements only if there's
    // an error.
    CantCreateWindow:
    CantGetNibRef:
     return NULL;
}
```

Document Revision History

This table describes the changes to *Unarchiving Interface Objects With Interface Builder Services*.

Date	Notes
2004-02-17	Removed stale link to *Learning Carbon*.
2001-05-01	First release of this document.

Document Revision History