Interface Builder

(Legacy)

Tools > Interface Builder

2007-04-05

Apple Inc.
© 2007 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

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

Apple, the Apple logo, AppleScript, AppleScript Studio, Aqua, Carbon, Cocoa, Mac, Mac OS, MacApp, Objective-C, and Xcode are trademarks of Apple Inc., registered in the United States and other countries.

Java and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

OpenGL is a registered trademark of Silicon Graphics, Inc.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Figures

Introduction to Interface Builder

Important: The information in this document applies to Interface Builder version 2.5, which is not supported in Mac OS X v10.5 and later. Instead, you should refer to *Interface Builder User Guide*, which documents the features and usage of Interface Builder 3.0.

Interface Builder 2.5 was the Xcode application used to create graphical user interfaces (GUIs) prior to Mac OS X v10.5. This programming topic provides some tips to help you make more effective use of this application and gives pointers to other documentation for Interface Builder.

Before you read this document, you should have basic familiarity with Interface Builder 2.5 and with the Xcode application. This document should be useful to anyone using Interface Builder, no matter whether you are writing Carbon, Cocoa, Java, or AppleScript applications.

Note: This is preliminary documentation that is currently in the process of review and revision.

Organization of This Document

This topic consists of several articles and is supplemented by a variety of other documentation as described here.

- "Compatibility Checking" (page 9) describes how to make sure your project does not use any interface elements that are not supported by the target version of Mac OS X.
- "Cocoa Custom Views" (page 13) explains how to use the Custom View widget in Cocoa applications and describes what you can do with a custom view.
- "Making Connections in Cocoa Applications" (page 17) explains how to make connections between elements of your Cocoa application's user interface, including widgets, controllers, menus, the Files Owner, and the First Responder. This article also explains the roles of actions and outlets, and tells you where you have to add your own code to make the connections work.
- "Frequently Asked Questions" (page 23) lists a number of questions users have had about Interface Builder and gives brief answers to them.

See Also

In Interface Builder, use the Help menu to open Interface Builder online help and release notes. When you install the Xcode tools on your system, the Interface Builder online help is also available in /Developer/Applications/Interface Builder.app/Contents/Resources/English.lproj/InterfaceBuilder_Help.

Developing Cocoa Objective-C Applications: A Tutorial in Cocoa Objective-C Language Documentation shows the development of a simple application using Objective-C and Cocoa and illustrates the use of Interface Builder.

A Quick Tour of Xcode in Tools Documentation includes a tutorial that shows the development of a simple application using Objective-C and Carbon and that illustrates the use of Interface Builder.

Developing Cocoa Java Applications: A Tutorial in Cocoa Java Documentation shows the development of a simple application using Java and Cocoa and illustrates the use of Interface Builder.

"Building a Text Editor in 15 Minutes" in the Cocoa programming topic Text System Architecture in Cocoa Text & International Documentation shows how to use Interface Builder to add a text view to a window, write a simple controller, and connect the text view to the controller.

Resource Programming Guide provides information about nib files in the runtime environment, including information about the nib-loading process for Carbon and Cocoa applications.

Interface Builder Services Reference in Carbon User Experience Documentation describes the Interface Builder Services manager. This manager provides functions that unarchive interface objects from nib files created using Interface Builder.

Unarchiving Interface Objects With Interface Builder Services in Carbon User Experience Documentation discusses key concepts needed to understand Interface Builder Services and strategies for storing interface objects. This topic also shows you how to unarchive interface objects from a main file, auxilliary nib files, and bundles other than the main bundle.

Handling Carbon Windows and Controls in Carbon User Experience Documentation has a section on Using Interface Builder in the chapter "Window and Control Tasks." Although geared to Carbon programmers, many of the principles and techniques described here apply to any project that uses Interface Builder.

Controller Layer in Cocoa Design Guidelines Documentation describes the purpose and use of controller objects and includes tutorials that show how you use Interface Builder to add controller objects to your Cocoa application.

Building Applications With AppleScript Studio in AppleScript Documentation has a section on Interface Builder Features for AppleScript Studio in the chapter "AppleScript Studio Components." This document also includes two tutorials with sections that illustrate the use of Interface Builder: "Currency Converter" and "Mail Search."

Compatibility Checking

Interface Builder stores your user interface design in one or more resource files called nib files. A nib file contains a representation of a set of interface objects and their relationships. With Mac OS X v10.2, Interface Builder started using a new format for Cocoa nib files. The older format is not capable of archiving some interface elements introduced with Mac OS X v 10.2 and later versions of Mac OS X. However, Mac OS X v10.1 and earlier systems cannot unarchive nib files saved with the newer format. Therefore, it is necessary to save your nib files in the format that is appropriate to your target version of Mac OS X (that is, the earliest version of Mac OS X on which you certify your application to run). For Carbon applications, there is only one nib file format, which is used for all versions of Mac OS X.

In addition, for both Carbon and Cocoa, new interface elements are introduced from time to time with new versions of the operating system. These elements are not available to programs running on older versions of Mac OS X, regardless of which nib file format is used.

Interface Builder lets you specify which format to use when saving a Cocoa nib file. It also provides a way to check the interface elements in your GUI for compatibility with specific versions of Mac OS X.

Nib File Format Compatibility

For Cocoa applications, three file formats exist for Interface Builder documents: "pre-10.2," "10.2 and beyond," and "both." The pre-10.2 format uses sequential archiving; the 10.2 and later format uses keyed archiving. See Archives and Serializations in Cocoa Data Management Documentation for details about these two archiving methods.

When you are editing your UI in Interface Builder, its default behavior is to preserve the file format of the current document. For example, if the existing nib file is in the old format, re-saving will use the old format. If the current nib file exists in both formats, then re-saving will use both formats.

To specify the format in which to save a new Cocoa nib file, open the General pane of the Interface Builder Preferences dialog and select one of the following options:

- Pre-10.2 format
- 10.2 and later format
- Both formats

If you select Pre-10.2 format, Interface Builder cannot archive any new types of objects introduced since Mac OS X 10.1 (such as the circular progress indicator or the brushed-metal, textured window appearance).

If you select 10.2 and later format, Interface Builder can archive all new features, but your application is not guaranteed to run in versions of Mac OS X prior to version 10.2.

If you select Both formats, Interface Builder saves the nib file twice, once in each format. When the user runs the application, the runtime environment automatically selects the appropriate nib file. In this case, you can add new features to your GUI; however, when the program runs in Mac OS X v10.1 or earlier systems, the new features will not be available. For example, a brushed-metal window displays with the older appearance instead. If you want to use the Both formats feature with newer interface elements, you need to test your application in Mac OS X v10.0 or v10.1 to make sure that the appearance and function of the windows and controls are satisfactory in all cases.

When you re-save an existing nib file, Interface Builder's default behavior is to retain the original format of the file, regardless of the format setting in Preferences. To change the format of the nib file, select File > Save As and select the format you want in the Save dialog.

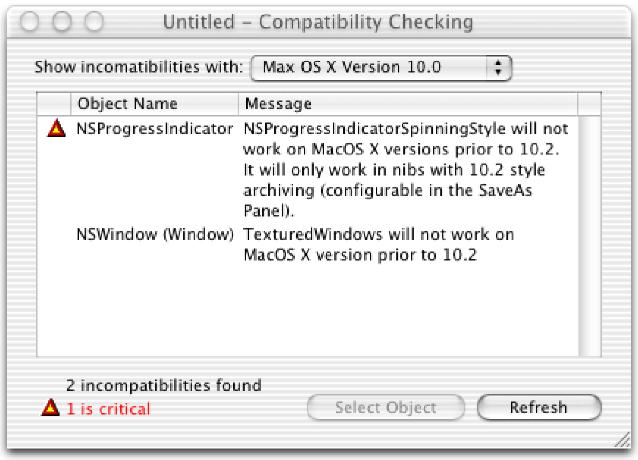
GUI Element Compatibility

New interface elements and attributes are added to Mac OS X from time to time and are represented in Interface Builder by new widgets and, sometimes, new palettes. Some elements are not supported by older version of Cocoa or Carbon.

Before saving a Cocoa nib file, Interface Builder performs a compatibility check to make sure that all of the objects in the nib can be archived using the nib's current format. Specifically, if the nib file format is "Pre-10.2", Interface Builder makes sure that each object in the nib can be fully archived using sequential archiving. If newer objects (that use keyed archiving only) are present in the nib, Interface Builder does not allow the save and displays an alert dialog indicating that saving with an incompatible nib format would result in data loss for the objects in question. To save such a nib, its format must be changed to "10.2 and later" or "Both."

Keep in mind that, although saving in both formats preserves the data in the nib file, newer keyed-archived objects will not work properly at runtime on older versions of Mac OS X. To check your Carbon or Cocoa nib file for compatibility with different versions of Mac OS X, select File > Compatibility Checking. This check will give warnings based on the operating system version you specify in the popup menu at the top of the dialog. If a feature you used in your nib will not work with the version of Mac OS X you specified, you will receive a warning. If your application will not be able to run using this nib file on this version of Mac OS X, the error is flagged as critical. Figure 1 illustrates the Compatibility Checking dialog.

Figure 1 Compatibility Checking Dialog


Untitled – Compatibility Checking
Show incomatibilities with:
Max OS X Version 10.0
Select Object
Refresh

Object Name	Message
⚠ NSProgressIndicator	NSProgressIndicatorSpinningStyle will not work on MacOS X versions prior to 10.2. It will only work in nibs with 10.2 style archiving (configurable in the SaveAs Panel).
NSWindow (Window)	TexturedWindows will not work on MacOS X version prior to 10.2

2 incompatibilities found
⚠ 1 is critical

Note that the interface elements available in Carbon are not identical to those available in Cocoa, and that similar elements might have been introduced at different times for these two development environments. To avoid unpleasant surprises, if your application is targeted to any version of Mac OS X older than the current version, run the compatibility checker before you finalize your design.

Legacy Document | 2007-04-05 | © 2007 Apple Inc. All Rights Reserved.

Cocoa Custom Views

One of the widgets provided with the Interface Builder palettes for Cocoa is the CustomView object, which is an instance of NSView. Because NSView is an abstract class, the CustomView object is usually used only as a stand-in for your own custom subclass of NSView,which will be instantiated at runtime.

Before you read this article, you should read "What Happens When a Nib File is Loaded," which describes what happens to standard objects, custom subclasses of standard objects, and custom objects (subclasses of NSView) when a Cocoa nib file is loaded.

How to Create a Custom View

To add a custom subclass of NSView to your interface, open the Classes pane in the Nib file window, click on NSView in the class tree, and select Classes > Subclass NSView. (If you prefer, you can subclass one of the existing subclasses of NSView instead.) Give the subclass a name. Or, if you have an Xcode project linked to your Interface Builder file, you can write a header file for your custom view in Xcode and drag the header file into Interface Builder's Nib file window.

Once you have defined your custom subclass, drag the CustomView widget from the palette into your design window. With the CustomView object selected, open the Custom Class pane of the Info window and select your subclass.

You can now make connections and set up targets and actions as you wish.

Because the code for your custom subclass is not resident in Interface Builder, you must first implement your custom subclass and build the program before you can test it. To implement your subclass, first be sure your project is open in Xcode. Then, select the subclass in the Classes pane of the Nib file window and select Classes > Create Files For *nameofsubclass*. (You can skip this step if you started by creating the header file in Xcode.) A stub header file and implementation file are added to Xcode, where you can add the methods you need to make the view functional.

Note that Interface Builder will continue to display the generic CustomView object (labeled with the name of your subclass). At runtime, when the custom view is unarchived, an instance of your subclass is created instead.

Because the default implementation of the `drawRect:` method for NSView does nothing, you must override this method if you want your custom view to be visible to the user. You must also override any event methods for events that interest you; the default implementations just pass the event on to the next responder. See the documentation for NSView for details.

To receive keyboard events, your custom view must accept first responder status. If your view needs preparation or cleanup as its responder status changes, or if it only accepts or relinquishes first responder status conditionally, you must implement the appropriate responder methods. See the documentation for NSResponder for more information.

Advantages and Disadvantages of Custom Views

As discussed in "What Happens When a Nib File is Loaded," your custom view object's `initWithFrame:` method is called when the window is first displayed. Only custom view objects receive this call; `initWithFrame:` is never called for custom subclasses of other objects. If you are not using a custom view object, you must implement a `-(void)awakeFromNib` method to handle any setup at runtime.

For example, suppose you create a subclass of NSOpenGLView and name it MyOpenGLView. You can add this subclass to your nib in either of two ways: you can add a CustomView widget to the design window and set its class to MyOpenGLview; or you can drag an NSOpenGLview widget into the design window and set its custom subclass to MyOpenGLview. Although the two methods seem very similar, the effect is quite different:

- If you use a CustomView object, then `initWithFrame:` is called when the window opens. However, because NSView is an abstract class, Interface Builder does not let you set any attributes in the Info window and you can't test the object in Interface Builder's Test Interface mode.
- If MyOpenGLview is set as a custom subclass of an NSOpenGLview object, the `initWithFrame:` method is never called for that object. In this case, on the other hand, Interface Builder's Info window displays a set of attributes for NSOpenGLview that you can set for your custom subclass of that object.

Other Uses for Custom View Objects

You can use a custom view object as an accessory view for classes such as NSSavePanel. To do so, drag a CustomView widget from the palette into your Nib file window. It appears as a top level object and a small window opens, labeled "View." You can add widgets to this window as needed. You must have an outlet in your controller for each object in the accessory view; use the Connections pane of the Info window to connect the outlets. You also have to add actions to objects as needed (buttons, check boxes, and so forth) and connect them to the appropriate controller. Remember to release the custom view when you're done with it.

A similar use of custom views is as the content view for a drawer. If you want to create a new window with a drawer, you can just drag the composite drawer object (the one that has a window with a drawer sticking off the left side of it) off the window palette onto the desktop. The composite drawer object contains a main window, a content view, and an NSDrawer object that come pre-wired. On the other hand, if you want to add a drawer to an existing window, drag the NSDrawer object (the one that says Drawer) into the Nib file window and a CustomView widget onto the design window. Then connect the NSDrawer object's `contentView` outlet to the custom view and its `parentWindow` outlet to the existing window. To test the interface, you can add a button to the design window and connect the button to the NSDrawer with the `toggle:` action.

You can also use custom views as simple containers of type NSView. To do so, drag a CustomView widget into your design window. Then, drag other widgets from the palette and drop them on the custom view. Or, you can select the objects that you want to group in the NSView container and select Layout > Make subviews of > Custom View. Using a custom view as a container allows you to manipulate the items in the view as a group. For example, you can make connections to the container and manipulate the items in the view at runtime. Or, you can swap out the contents of the custom view and replace them with the contents of another view.

Releasing Custom Views

The file's owner is responsible for releasing any resources created by the nib. These include any top level objects such as formatters, custom views, extra windows, or extra menus. See "Releasing Nib File Objects" in "What Happens When a Nib File is Loaded."

Making Connections in Cocoa Applications

For Cocoa-based applications, you can use Interface Builder to connect UI elements to each other and to your controller code so they can pass messages back and forth. Any object (and controllers in particular) can have an outlet, which is an instance variable pointing to some other object. It is often useful to have your program's controller objects have outlets to specific user interface elements, so that the controller can tell the UI elements to change their state, display text, and so forth.

Controls have a special way they can connect to other objects: they can assign an object to be the target of the control, and a particular method inside that object to be the action that gets called when the control is triggered. How the control looks and functions are handled by the Cocoa infrastructure, so that all you have to implement are the target objects and their action messages. For more information on the target-action paradigm, see Communicating With Objects in Cocoa Design Guidelines Documentation.

To establish either outlet or target-action connections, you control-click the object that will be the source of the messages and drag to the object that will be the destination of the messages. Interface Builder displays a gray line between the two objects, with a square at one end showing the origin of the message. When the connection is displayed, you can open the Connections pane in the Info window to see lists of the available outlets and actions in the inspector, select the outlet or action you want, and click the Connect button to establish the connection. The key thing to remember is to always control-drag in the direction that messages will flow.

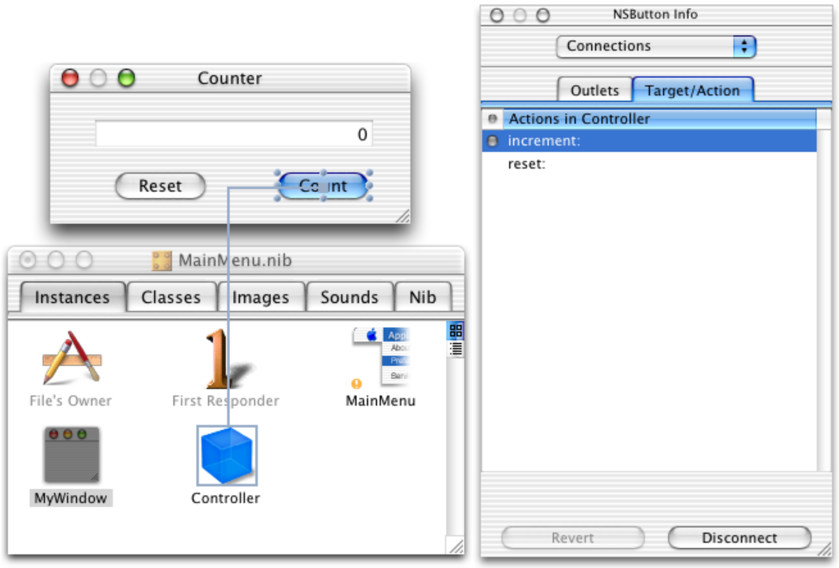
Target-Actions

Certain objects in your user interface, when manipulated by the user, send a message to another object (the *target*) telling it to perform some *action*. For example, a button can send a message to a drawer, telling it to open or close. For Cocoa applications, Interface Builder makes it easy to "wire up" these relationships. To do so, you Control-click on the object that sends the message (the control) and drag to the object that performs the action (the target). The Connections pane in the Info window then shows a list of the actions the target knows how to perform. Select the action you want and click Connect.

Actions are methods implemented by the target. When you connect a control to a target and select an action, that tells the control to send a message (containing any necessary data) to the target telling it to perform that method when the user manipulates the control. To summarize: for target-action connections, the action message is sent from a control to a target; the action method is implemented in the target.

For example, Figure 1 shows the connection of a button labeled "Count" to the Controller object so that the `increment:` action will be called when the button is clicked. You do this by control-dragging from the button to the controller, selecting "increment:" in the Target/Action tab of the Connections pane, and clicking Connect.

Figure 1 Connecting an action to the Controller



In many cases, the target already has an implementation of the action method you need and you do not have to write any code to make this connection work. For example, the Drawer object has implementations of the actions `open:`, `close:`, and `toggle:`. However, to perform an action not already provided, you must add the action to the target before you make the connection.

To add actions to a target in your project, you use one of the following conventions for the method prototype in your code:

```
-(void) myActionMethod: (id)sender;-(IBAction) myActionMethod:(id)sender;
```

In the example in Figure 1, the `increment:` action is prototyped in the Controller code as:

```
- (IBAction)increment:(id)sender;
```

Select the class to which you want to add the action in the Classes pane of the Nib file window and select Classes > Add Action to [*name of class*].

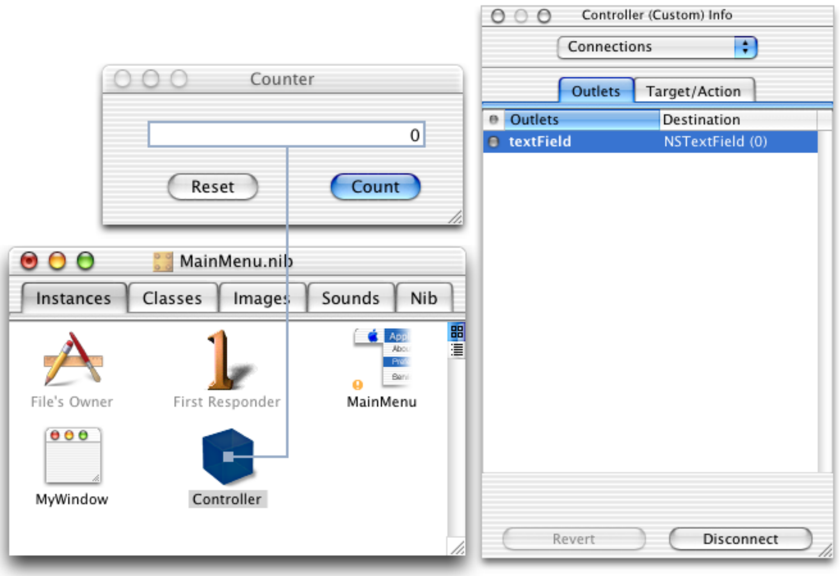
For more information on the target-action paradigm, see Communicating With Objects in Cocoa Design Guidelines Documentation.

Outlets

An outlet is an instance variable in one object that can be linked, through an ID, with another object. When you make an outlet connection, you are in effect telling the object that contains the instance variable the location of the object to which you are making the connection.

For example, in Figure 1, the `increment:` message is sent from the button to the Controller object. The Controller object executes the `increment:` method. In this example, the application increments an integer and then displays the new value in a text field. In order for the Controller object to send a message to the text field telling it to display this data, it must have a pointer to that control at runtime. To give the Controller that pointer, you Control-click the Controller object and drag to the text field. You then select the outlet in the Outlets tab of the Connections pane (`textField` in Figure 2) and click Connect. That tells Interface Builder that the instance variable `textField` in the Controller is the outlet; that is, it should contain a pointer to that text field. At runtime, when an instance of the Controller object is created, the outlet `textField` is given a pointer to the text field.

Figure 2 Connecting an outlet



Outlets	Destination
textField	NSTextField (0)

Many objects include instance variables that can be used as outlets. For example, an NSDrawer object has the outlets `contentView`, `delegate`, and `parentWindow`. If the object you are using doesn't already have the outlets you need, you can add your own outlets to it. To add outlets to your project, you use one of the following conventions for the variable declaration in your code:

```
id aDynamicallyTypedInstanceVariable;IBOutlet NSButton*
aStaticallyTypedInstanceVariable;
```

For example, the `textField` outlet is declared in the Controller code as:

```
id textField;
```

Select the class to which you want to add the outlet in the Classes pane of the Nib file window and select Classes > Add Outlet to [*name of class*].

The runtime code uses the list of outlets you generated in Interface Builder. For example, if you specified an outlet called `textField`, Interface Builder first investigates the class containing the outlet to see if it responds to `setTextField:(id)`. If it does, that method is called with the pointer to the text field you established in Interface Builder. If `setTextField:(id)` doesn't exist, the instance variable `textField` itself is filled with the pointer. The object with the outlet must know how to send data to the object pointed to by `textField`. For standard outlets listed for an object in an Interface Builder palette, see the documentation for the class of that object to find out what you can use the outlets for. If you are writing your own custom subclass or your own controller, look at the documentation for the class of the receiving object to find out what messages it can handle.

Locking Connections

Connections are a fundamental part of an application's object infrastructure. That is, object-oriented code depends as much on the messages sent between objects as on the definitions of the objects themselves. Once you have developed and debugged an application, therefore, it is highly desirable to keep the connections from being inadvertently altered. Towards this end, Interface Builder has a preference that locks all connections for a user. You might use this feature, for example, if you are localizing a nib and do not want to accidentally change any connections. This setting is on the Editing pane of the Preferences dialog.

Connecting to First Responder

First Responder is your portal to the responder chain. You can add actions to First Responder in the Classes pane of the nib file window. Then, connect buttons and menu items to First Responder so that they call the desired action. The first object in the responder chain that understands this action will be called.

Because keystrokes do not point anywhere on the screen, all keystrokes are sent to the first responder. You can designate the first responder as the target of a control by connecting the control to the First Responder icon in the Instances pane of the Nib file window. When you do so, you can choose from the list of supplied actions, or you can add your own actions to the project and use one of those for the First Responder.

The responder chain is discussed in the programming topic Introduction to Cocoa Event-Handling Guide in the Cocoa Events and Other Input documentation area.

Connecting to File's Owner

File's Owner represents the object that will be passed in for owner in the method `[NSBundle loadNibNamed: owner]`. That is, File's Owner is a proxy for an object that exists outside of the nib file but that can be connected to objects in the nib file. You can use the Custom Class Info Panel to specify what kind of object File's Owner is. Once you've indicated what File's Owner is, Interface Builder knows what outlets and actions are available for it and you can make connections to it.

Connecting to the Menu Bar

Items in menus are connected to objects in your application in a similar fashion to that used for controls and views. As an example, the following procedure could be used for multi-document applications with multiple nibs.

Use the following procedure to make connections from the menu bar to your window controller:

1. In your main nib, instantiate a subclass (usually of NSObject or NSDocument) to use as your window controller.
2. Select the FirstResponder class in the Classes pane of the Nib file window (or double-click the First Responder icon in the Instances pane) and add the actions that you want to connect to a menu item.

3. Control-drag from the menu item in the Menu Editor window to the First Responder icon in the Instances pane of the Nib file window.

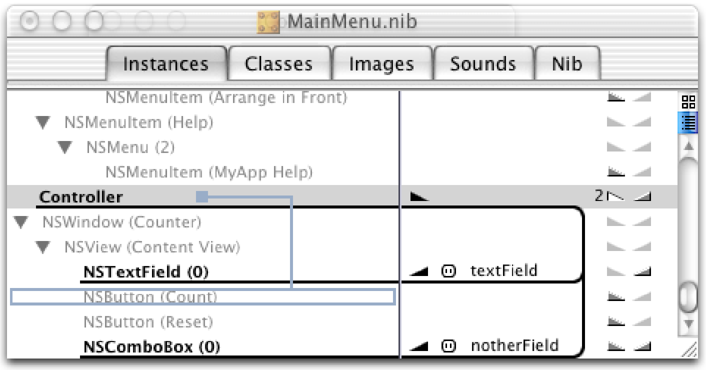
4. Connect the action you want the menu item to initiate.

At runtime when one of your nib file windows is key, its window controller will be in the responder chain (because the window controller is the delegate of the window) and the menu item will work. This is one of the reasons for the existence of the FirstResponder class.

Making Connections in Outline View

You can make connections in outline view as well as in icon view. There are some advantages to outline view: you can see multiple connections at one time, you can find deeply-nested subclasses more easily, and you can tell which objects have both outlet and target-action connections. Figure 3 shows an example of connections displayed in outline view.

Figure 3 Making connections in outline view



In the figure, the two outlet connections for the Controller object are highlighted by clicking on the right-facing triangle with the 2 next to it. The left-facing triangle on the same line highlights the one target-action connection that has the Controller object as a target. A new connection is being made from the Controller object to the NSButton(Count) object, as indicated by the gray line. To make the connection, you Control-click on the origin of the connection and drag to the class name representing the object to which you want to connect.

Use the icon at the top of the vertical scroll bar to toggle between icon view and outline view.

Legacy Document | 2007-04-05 | © 2007 Apple Inc. All Rights Reserved.

Frequently Asked Questions

This article gives brief answers to a number of questions that developers have had about Interface Builder.

General

- What do the non-integral rectangle warnings mean and how do I get rid of them?

  Non-integral rects are frame definitions that contain float values. Although not critical, these can lead to performance problems and blurring. Interface Builder will discover and correct any non-integral rects it finds when you open a nib.

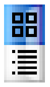
- How do I put widgets inside a container such as a box or tab view?

  Click on a category in the Palette toolbar. Drag a widget from the palette over your design window. A selection ring will appear around the deepest element that can accept the widget. Release the mouse button to drop it. Once an element is embedded in a container, it can only be removed by deleting or cutting it.

- How do I see the containment hierarchy of my document?

  To see a hierarchical list of all the objects within your document, click the outline-mode button located above the scrollbar in the nib file window. You can use this view to select objects and widgets that are difficult to reach through the design window. You can also make connections from the outline view. To switch back to the icon view, click the icon-mode button immediately above the outline-mode button. The buttons are shown in Figure 1.

  Figure 1 Buttons for Outline and Icon View modes

- How do I switch tabs in a Tab view in the design window?

  Double click an area of the tab view free of embedded widgets to activate the tab view editor (a ring is drawn around the editor to show it is active). Then click once on the tab you would like to view, click the stepper control in the Info window, or press the Tab key to go to the next tab or Shift-Tab to go to the previous one.

- How do I edit tab titles?

  Double-click an area of the tab view free of embedded widgets to activate the tab view editor (a ring is drawn around the editor to show it is active). Double click the text of the tab you wish to change. Enter the new text and press Return.

- How do I add a tab to my tab view?

Single-click the tab view to select it. Open the Attributes panel in the Info window. Add new tabs by increasing the number listed in that pane. Alternatively, you can double click the tab view to activate the tab view editor. Select the tab that is closest in design to what you want for the new one. Then copy and paste. You get a new tab that is a replica of the selected tab.

- How do I change the tab order of a tab view?

  Double click the tab view to open its editor. Select the tab you wish to reorder and press the arrow keys to reorder it. If this procedure doesn't work, try using the stepper control in the Info window to select the tab you want to reorder before pressing the arrow key.

- How do I edit a widget's properties?

  Select the widget in the design window. Open the Info window with the Tools > Show Info menu item. Make your changes using the various info panels.

- How do I add menus to the menu bar?

  To add individual items, drag an "Item" menu from the menu palette into your menu editor. To add a new submenu, drag a "Submenu" item from the menu palette into the menu editor. Double click these items in the menu editor to change their names. The other menu types in the menu palette represent various system menus.

- How do I add menus to a popup button?

  Double click the popup button. A menu editor will appear. Use the same techniques described above to add menu items to the editor.

- Can I create an interface without using Interface Builder?

  Interface Builder is a tool that manipulates Cocoa and Carbon interface elements. You could manipulate these same elements programmatically, but using Interface Builder is easier.

- Why is some text gray, and other text black?

  Gray elements represent items defined by system frameworks. Often, these elements can't be deleted or changed. Files Owner, the NSApplicationIcon and the description for NSObject are examples of system defined elements.

- How can I turn off the Aqua guides?

  Use the Layout > Disable Aqua Guides menu item. You can hide any user guides you have created with the Layout > Hide Guides menu item. Hold down the Command key while you drag widgets to temporarily reverse your guide settings.

- How do I remove a user guide?

  User guides can be removed from the document by dragging them off the edge of the window.

- Does File Merge work with nibs?

  Yes. You can compare two nib files using File Merge to see what has changed, but you can't merge to a new file. Behind the scenes, File Merge runs nibtool on the two nib files and diffs the textual output.

- Something is wrong with my Interface Builder preferences. How do I reset my preferences?

  Quit Interface Builder. Open a Terminal window and type the following: `defaults remove com.apple.InterfaceBuilder` The next time you launch Interface Builder, a new set of preferences will be created.

- How can I easily see the header for my classes?

The “view in PB” button in the Attributes Info Panel is active while Xcode is running. Clicking the button opens the class’s header file for Cocoa objects and for custom classes if the source files for the class have been created.

- Can I add UI elements to closed containers

  A container does not have to be “open” (that is, you do not have to double-click it) before it will accept new UI elements or formatters. When you drag a UI element from the palette to a container, the deepest possible destination displays a ring. You can release the mouse to deposit the UI element in the container.

- Can I change the configuration of the tool bar in the Palette window?

  The control used to select the currently active palette is an NSToolbar. You can reorder the palette items and do anything that the NSToolbar allows you to do in the palette window. This allows you to move your most frequently used palettes closer together and allows easier loading of palettes.

Cocoa-Specific

- What kinds of images can I add to my Cocoa nib?

  You can add any image supported by NSImage. Any image you wish to use must have an extension understood by NSImage (.tiff, .jpeg, and so on). See the question “How do I get Interface Builder to display images from my project?” to learn how to use these images.

- Should I release top level objects at runtime?

  Usually, yes. The file’s owner is responsible for releasing any resources created by the nib. These include any top level objects such as formatters, custom views, extra windows, or extra menus. However, there are a few exceptions to this rule. Windows with the “Release on Close” flag set do not need to be released; they will release themselves when they close. Windows controlled by a window controller will also be released automatically.

- What does Instantiate do?

  A frequent misconception of nibs is that they are simply recipes for recreating user interfaces at runtime. In reality, nibs contain actual archived objects. Although most of these objects are in fact user interface elements, nibs can contain archived instances of other objects such as controllers and formatters. The “instantiate” command in the Classes pane causes the selected object to be created at runtime. It’s common to instantiate a window’s controller class alongside the actual window in the nib. When the nib is loaded, the window and controller are created and then any connections between them are established.

- How do I put widgets inside a scroll view?

  Select the widget(s) and use the menu item Layout > Make subviews of > Scroll view. If only one widget is selected, Interface Builder attempts to make it the NSScrollview’s document view. If more than one widget is selected, they are enclosed in an NSview and that is made the document view.

- How do I put widgets inside a split view?

  Select the widgets and use the menu item Layout > Make subviews of > Split view. If the widgets are arranged side to side, the split is vertical. Conversely, if they are arranged from top to bottom, the split is horizontal.

- How do I make a drawer?

There are two ways of creating drawers in Interface Builder. To create a new window with a drawer, drag the composite drawer object off the window palette onto the desktop. The composite drawer object is the one that has a window with a drawer sticking off the left side of it. It's NOT the object labeled "Drawer". The composite drawer object contains a main window, a content view, and an NSDrawer object. These objects come pre-wired. All you have to do is connect a widget to the "toggle" action of the NSDrawer object. Double-click the content view icon in the Instances pane of the nib file window to open the content view so that you can add views to it. To add a drawer to an existing window, drag the NSDrawer object (the one labeled "Drawer") into the nib file window and a custom view object (labeled CustomView) onto the design window. Use the custom view object as the content view for the drawer by connecting the NSDrawer's `contentView` outlet to the custom view and its `parentWindow` outlet to the existing window.

- Why does my bottom or top drawer size itself improperly?

  There is a known problem in the underlying NSDrawer implementation that adversely affects proper drawer size calculations. Specifically, when a drawer's preferred edge is left or right, the leading and trailing offsets are used to automatically calculate the height of the drawer. When a drawer's preferred edge is top or bottom, the leading and trailing offsets should be used to automatically calculate the width of the drawer. However, these values are incorrectly applied to the height.

- What do File's Owner and First Responder represent?

  File's Owner and First Responder are proxies for objects that will exist at runtime. File's Owner represents the object that will be passed in for owner in the method [NSBundle loadNibNamed: owner]. You can use the Custom Class Info Panel to specify what kind of object File's Owner will be. Once you've indicated what File's Owner is, you can make connections to it. First Responder is your portal to the responder chain. You can add actions to First Responder in the Classes pane of the nib file window. Then, connect buttons and menu items to First Responder so that they call the desired action. The first object in the responder chain that understands this action will be called. See the Cocoa documentation for more information about how the responder chain works.

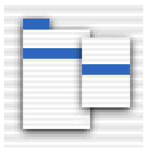
- How do I set up the springs and struts so my views autosize?

  Views can be made to resize automatically when the window or split view that contains them is resized. You will probably need to set the springs and struts for every view in your window. The easiest way to see how they work is to assign a few views with different settings and then use File > Test Interface to see the results. To see how this works, bring up the Size info panel and select a square bevel button. There are two parts to Cocoa's auto sizing: growing and anchoring. How the button grows is determined by the strut/spring setting shown inside the button picture in the info panel. If an axis (horizontal or vertical) is a spring, then the view can grow when its container is resized. If it's not a spring, but rather a strut, the button will remain the same size. Click on the horizontal axis inside the mock button to turn it into a curly spring. Then use the menu item File > Test Interface to test the interface. Grow the window in various directions. The button gets longer, but never taller or shorter. Click the switch icon in the menu bar or press Command-Q to stop testing the interface. Repeat this process, but with the vertical axis set as a spring. Now it can get taller or shorter but never wider. The outer set of springs and struts represents how the view is to be anchored. If a segment is a strut, the view maintains a constant distance to the side of its container along that side. However, if it's a spring the view allows the distance to vary. Select the sample square bevel button and click the left-most segment so that it turns into a spring. Because the segment at right is still a strut, the view maintains a constant distance to the edge of the window from the view's right side, but allows a variable distance along the left side. Test the interface to see the effect.

- How do I add a contextual menu with Interface Builder?

Select the Menu palette and drag the icon that looks like Figure 2 into your nib file window. A menu editor window opens for the new menu. Control-drag from the view for which you want a contextual menu to this menu icon to connect the `menu` outlet. You can use the menu editor window and the info panel to edit and add items to the contextual menu.

Figure 2 Contextual menu icon



- How do I use a formatter?

  There are two ways to add a formatter to a widget in Interface Builder. The easiest way is to drag it from the palette and drop it on the widget (providing that widget supports formatters). You can then select the widget and bring up the Formatter Info pane to set the formatter's attributes. Alternatively, you can share a single instance of a formatter with several different widgets by dragging the formatter to the Nib file window and connecting the widget's `formatter` outlet to the formatter in the Nib file window. As with any top level object, be sure to release it when you're done.

- How do I make a matrix (rows and columns) of a widget?

  Any widget that has an NSCell counterpart can be made into a matrix. Hold down the Option key while you drag a selection handle. The farther you drag the handle, the more new rows and columns are appended to the matrix. Some other widgets, such as NSBrowser and NSForm, work the same way.

- How do I give a browser more than one column?

  Hold down the Option key while you drag one of the side selection handles. The farther you drag the handle, the more new columns are appended to the browser. To add or remove columns without changing the overall size of the browser, change the number in the Visible Columns field of the browser's Attributes pane in the Info window.

- How do I add a table column?

  There are two ways to add columns to a table. The easiest way is to select the table and use the "#Colms" field in the table's Attributes pane to add new columns. Another way is to open the table's editor by double clicking it. Now select a table column. Copy and paste it to make new ones.

- How do I add a class definition?

  Switch to the Classes pane in the Nib file window. In this pane, you'll see a listing of all the class types this nib understands. All of the Cocoa objects are predefined, but you'll have to tell Interface Builder about your own classes before you can make connections to them. You can do this in either of two ways: If you have header files for your new classes, select the Classes > Read Files menu item and specify the header for the class you wish to describe. Interface Builder creates a new listing for this class under its superclass and fills it with any actions or outlets defined in the header. If you don't have the header file, you can subclass one of the class descriptions already defined in the Classes pane. Select a class, such as NSObject, select the Classes > Subclass menu item, and name the subclass. Then select your subclass and bring up the Attributes pane of the Info window. Select the Objective-C or Java radio button, as appropriate for your class.

- How do I add outlets and actions to a class?

You can add outlets only to a user defined subclass. Because of the protocol feature of Objective-C, however, you can add an action to any class, whether built in or user defined. Select the Classes pane in the Nib file window. Select the class definition to which you wish to add outlets and actions. The actions and outlets defined by the superclass are listed in gray in the Attributes pane of the Info window. Click the Add button to add actions and outlets to your subclass.

- After I've made class definitions in Interface Builder, how do I write them out?

  Select the class definition in the Classes pane and use the menu Classes > Create Files. If this nib is part of a loaded Xcode project, it may ask you if you would like to include the file(s) in a target. Depending on the type of class you're creating, you'll get a .m/.h set or a .java file.

- What are IBOutlet and IBAction doing in my code?

  To understand why they're added to your code, you must understand how Interface Builder parses your code. When you choose Classes > Read Files, Interface Builder tries to find the outlets and actions you've declared in your header. You can use these signatures: for Outlets: `id someOutlet;` for Actions: `-(void)myAction:(id)sender` Some people prefer stronger typing of their outlets, so they would try: for Outlets: `NSTextField* someOutlet;` for Actions: `-(void)myAction:(MyApplication*)sender;` However, Interface Builder can't tell the difference between these signatures and other non-outlet/non-action declarations, so it ignores them. To solve this dilemma, the no-op keywords IBOutlet and IBAction were introduced. Although the first set of signatures still works, Interface Builder also parses methods and fields containing these keywords: for Outlets: `IBOutlet NSTextField* someOutlet;` for Actions: `-(IBAction)myAction:(MyApplication*)sender;`

- How are outlets connected at runtime?

  The runtime code uses the list of outlets you generated in Interface Builder at design time. For example, if you specified an outlet called `foo`, Interface Builder first investigates the class containing the outlet to see if it responds to `setFoo:(id)`. If it does, that method is called with the contents for `foo`. If `setFoo:(id)` doesn't exist, the instance variable `foo` itself is filled with the contents for `foo`.

- What is an object's custom class?

  Many widgets have a Custom Class pane in the Info window. You can use this pane to specify that, at runtime, an object should be instantiated as a specific subclass. For example, you could make a subclass of NSApplication called MyApplication. To ensure that an instance of MyApplication is used at runtime, click the File's Owner of your main nib file. Bring up the Custom Class pane and select MyApplication. Many controls can have custom classes. You could subclass NSSlider to add specific behavior. However, you can still use Interface Builder's slider facilities. By setting the slider's custom class, your custom slider is implemented at runtime, but a regular slider is used at design time.

- • How do I subclass a widget in Cocoa Java?

  To subclass a Cocoa Java widget, in addition to selecting your custom Java subclass in the Custom Class pane of the Info window, you must also create the .java file for the subclass and implement a special constructor that takes an NSCoder and a long. Here is an example: `protected MyButton(com.apple.cocoa.foundation.NSCoder decoder, long token) { super(decoder, token); }`

- Why does my application crash when I implement `initWithCoder` (or the above referenced Java Coder constructor) in my custom subclass of a Cocoa widget?

  If you have implemented a subclass of a Cocoa object and then specified it as the class of a widget with the Custom Class pane of the Info window, your custom subclass must not implement any decoding of its own if the nib file format uses non-keyed archiving (pre-10.2). This is because the class swapper that installs your subclass relies upon the coding signature of the real Cocoa widget's class rather than that

Legacy Document | 2007-04-05 | © 2007 Apple Inc. All Rights Reserved.

of your actual subclass. Non-keyed archiving is very particular in such situations, and will crash if the coding signature is different. Note: even if it did not crash, such a scheme would not be useful since Interface Builder has no way of encoding anything useful to the subclass.

- How do I subclass NSTextview in Interface Builder?

  Select NSTextview in the Classes pane of the Nib file window and create a subclass for it. Drag an NSTextview widget from the palette into your design window, double-click the widget to edit it, and then select the desired subclass in the Custom Class pane of the Info window. Please note that the nib file format MUST be 10.2 or beyond for this feature to work, which means that the nib will not properly load on pre-10.2 systems.

- What is the difference between using a custom class and a custom object?

  A custom class is a subclass of one of the interface objects provided in an Interface Builder palette. For a custom class, you can set some properties of the object with the info panel in Interface Builder rather than having to do it all in your code.

  A custom view object is a subclass of NSView. You can define this view any way you like; it is more versatile than a custom class of an existing interface object. However, you must set all the properties programmatically. For more information, see "Cocoa Custom Views" (page 13).

- What is a dynamic palette and how do I make one?

  Dynamic palettes are custom palettes created by the user. Select the menu item Tools > Palettes > New Palette. A blank palette is created for you. You can now Option-drag widgets from the design window onto the palette. Option-drags containing multiple widgets come off the palette as one unit. Save the palette using the menu item Tools > Palettes > Save.

- How do I remove palettes?

  Palettes can be removed from the toolbar through the Tools > Palettes > Customize Toolbar menu item. Drag them off the toolbar after the customize sheet drops. To remove the palette from the list of available palettes, select the Palettes pane in Interface Builder Preferences. Remove the palette from the palette directory if you don't want it to appear in this list. Built-in Interface Builder palettes cannot be removed.

- How do I make a UI element smaller than the smallest size allowed by Interface Builder?

  When resizing a UI element such as a button or group box, Interface Builder doesn't let you resize it past the minimum size defined by the Aqua User Interface Guidelines. If you really need to resize it smaller, then first resize it to its smallest size and then resize again. Interface Builder will let you resize a UI element past its minimum size if the current size is the minimum size.

- I have an outlet named `rate` and in the same class a method named `-setRate:`. When my nib is loaded the `-setRate:` method is called. Why?

  The rule to set outlet connections is as follows: Given an outlet named `foo`, the loader first looks for a setter method named `-setFoo:`. If this method exists, the loader calls it, passing the value of the outlet as a parameter. If this method doesn't exist, the loader then looks for an instance variable named `foo`. If this instance variable exists, it is set directly. If this instance variable doesn't exist, the loader logs an error.

- I have a control in a container (for example, a tab view) and I want to move this control outside of the container. Cut and Paste break connections. How can I move a control between containers without breaking connections?

  This is a known limitation. The workaround is to Option-Drag the control to another window within the same nib file and then Option-drag it back to the original window (if you don't have a second window in your nib, then create a temporary window). Using Option-drag between windows in the same nib doesn't break connections.

- How do I change the data cell object of an NSTableColumn?

  To set a custom data cell (other than NSTextFieldCell) as the data cell for an NSTableColumn, drag a dataCell object from the Data Views palette and drop it on a column header. To inspect the data cell, select the table column first, and notice that the selected table column displays a small triangle in the corner of the header. Click that triangle once to inspect the data cell. Note that the Info window will let you set certain attributes that may not make sense for display in a table view, such as bezeled background.

- When I add more than 10K of text in an NSTextview in Interface Builder, it crashes.

  This is a known bug in the archiving mechanism in NSTextview. If you have added more than 10288 characters of text to an NSTextview, Interface Builder crashes immediately. The workaround is to set up the text view in Interface Builder, but use your `awakeFromNib` method to load the text programmatically into the text view.

- What are those little yellow badges that show up on the objects in the Instances pane?

  In the Instances pane of the Nib file window in Cocoa projects, a little yellow exclamation point (!) is displayed whenever the object, or a child of that object, is missing a connection. Hold the mouse over the (!) symbol for a tool tip that further defines what's wrong. Switch to the Outline view to see which child of the object is causing problems. The symbol is displayed for any object that meets any of the following criteria:

  - The object is an instance of NSWindowController and you haven't connected the window outlet.
  - The object follows the target/action paradigm and there is no target/action specified.
  - Not all of the outlets of a custom object are connected.
  - The delegate and data source of NSOutlineview and NSTableview are not connected.
  - The delegate of NSBrowser is not connected.

- What is the Document Info item in the File Menu?

  Choosing File > Document Info generates a sheet that tells the user almost everything that Interface Builder can ask Xcode about the current document and the project the nib is in.

- What is the Bindings pane and how does it work?

  Cocoa includes a feature that allows you to bind the value and behavioral elements of UI widgets to a controller object. Controller objects (such as NSUserDefaultsController, NSObjectController, and NSArrayController) establish and manage a relationship between UI elements and the data that drives them, significantly reducing or eliminating the need for custom code to manage a detailed user interface. For each object selected in Interface Builder, the Bindings pane of the Info window displays a list of possible bindings that can be bound to a controller object. For example, to bind the value of an NSTextField to an NSUserDefault value, you can use the following procedure:

  1. Drag an NSTextField to a window and select it.
  2. Open the Bindings pane of the Info window.
  3. Scroll to the "value" binding and click to display its details.
  4. Use the Shared User Defaults controller.
  5. Select the "value" model key.
  6. Enter a name for a key. This key is used to store the value in the Shared User Defaults controller.
  7. Do the same for a second NSTextField, using the same name for the key.

8. Test the interface: changing the value in one text field should cause the other to change as well.

9. Quit the interface test mode and then enter the interface test mode again. The fields should both contain the last value entered, preserved by the Shared User Defaults controller.

- How can I add a new controller to my nib for use in the Bindings inspector?

  Drag a controller of the type you want from the Controller palette to your Nib file window. If the Controller palette does not automatically show up in the palettes toolbar, it might be because you have a customized toolbar. Select > Tools > Palettes > Customize Toolbar to add the Controller palette to your palette toolbar.

- How do I specify and use a custom subclass of an NSController?

  Use the following procedure:

  1. Create an instance of NSController (or an NSController subclass) either by dragging one from the Controller palette to the Nib file window, or by instantiating NSController from the Classes pane of the Nib file window.

  2. Select the NSController class or subclass in the Classes pane of the Nib file window and use Classes > Subclass NSController to create your custom subclass.

  3. Select the instance of NSController in the Instances pane of the Nib file window and use the Custom Class pane of the Info window to set its class to your custom subclass.

  This method allows Interface Builder to use the concrete class and all of its facilities while you edit your nib, including displaying it in the Bindings pane of the Info window. Note that first creating a subclass of NSController and then instantiating the new subclass would not allow Interface Builder to use the class and display it in the Bindings pane. Therefore, for NSController and its subclasses, instantiation of subclasses is disabled.

- How do I use a WebView object in my nib?

  Drag a WebView object from the WebKit palette to you design window. WebView is a subclass of NSView, and thus behaves similarly in terms of size and auto-size behavior.

  The WebKit inspector has settings that allow you to tune its behavior. The Preferences ID is a way to allow multiple WebView instances to share the same settings across instances and nibs. If no ID is set, the default shared WebPreferences object is used.

  If the WebKit palette does not automatically show up in the palettes toolbar, it might be because you have a customized toolbar. Select > Tools > Palettes > Customize Toolbar to add the WebKit palette to your palette toolbar.

Carbon-Specific

- Do nibs work on MacOS 8/9?

  Yes. IBCarbonRuntime.h has been a part of CarbonLib since version 1.1 (System 8.6). Because IBCarbonRuntime uses CFBundle, you need to package your application correctly.

- How do I load a nib in Carbon or CarbonLib?

See the sample code in /Developer/Examples/InterfaceBuilder/IBCarbonExample. It uses the `CreateNibReference`, `CreateWindowFromNib`, and `SetMenuBarFromNib` functions to unarchive a Carbon nib. This example also shows how to register a simple Carbon event handler.

- How do I import a .rsrc file from Carbon?

  Interface Builder can import the following compiled resource types: DLOG, DITL, CNTL, MBAR, MENU, WIND, and tab#. DITL and CNTL resources are mapped directly to appearance-savvy controls. Be sure to select the correct text encoding before you import your resources. Interface Builder does not support rez files.

- What kinds of images can I add to my Carbon nib?

  Interface Builder currently understands the following Carbon resource types: ICON, icns, cicn, PICT, and Icon Ref. Tiffs and other NSImage types are not understood by Carbon nibs. See the question "How do I get Interface Builder to display images from my project?" to learn how to use these images.

- Why don't my Carbon tab views switch at runtime or when I test the interface?

  The Carbon tab control does not automatically switch tab panes for you. Interface Builder does this for you as a convenience at design time. The example in /Developer/Examples/InterfaceBuilder/IBCarbonExample shows one way of switching tabs.

- Can I set up my custom controls in Interface Builder?

  Two types of custom controls are supported: CDEF Proc Pointer and event-based custom controls. In both cases, drag a Custom Control widget from the Other palette into your design window and use the Info panel to select a type of custom control. Look at the IBCarbonExample sample located in the /Developer/Examples/InterfaceBuilder folder for more details on how to use custom controls in your application.

  Note: A Carbon nib file that contains a custom control will not work on MacOS X 10.0.x machines.

- Does Interface Builder support MacApp?

  No. However, you can unarchive a Carbon nib and manipulate it with standard Carbon functions.

Integration With Xcode

- How do I get Interface Builder to display images from my project?
  - For Carbon nibs:

    Make sure the project is open in Xcode. Add a compiled resource file containing your images to your Xcode project and open the nib. Carbon nibs look inside compiled resource files for PICT, icns, cicn, and ICON resources. These resource files must have the extension .rsrc.
  - For Cocoa nibs:

    Make sure the project is open in Xcode. Add your images to your Xcode project. Any images you add must have an extension understood by NSImage (.tiff, .jpeg, and so forth). Then open the nib.
  - General rules:

    Interface Builder only shows images it can guarantee will exist at runtime, so make sure your images or resources are included in the target that contains the nib. If a nib belongs to multiple targets, it will only show images contained within every one of those targets.

- My nib lost all of its images! Where did they go?

  Xcode must be running with your project open in order for Interface Builder to discover which images are available to you. If your project is open, but you still don't see any images, or only see a subset of images, check your target settings. Interface Builder only shows images it can guarantee will exist at runtime, so make sure your images are included in the target that contains the nib. If a nib belongs to multiple targets, it only shows images contained within every one of these targets.

- Will Classes > Create Files overwrite my existing files?

  If the files Interface Builder is trying to write already exist, you are given the option to merge your changes through File Merge. You can also tell Interface Builder to stop, or overwrite the existing files.

- I tried to drag an image into Interface Builder and it said I should use Xcode, but I don't want to.

  If you really want to store files within the nib, you can check the preference "Allow images to be stored inside the nib". This is not a recommended practice, however.

- I have problems adding nib files to my CVS repository

  The easiest way to add a nib file to a CVS repository is to use the cvswrappers tool. Create a .cvswrappers symlink in your home directory that points to the cvswrappers file located in the /Developer/Tools directory: `ln -s /Developer/Tools/cvswrappers ~/.cvswrappers` You can now add nib files to a CVS repository the same way you do with regular files.

nibtool Specific

- Why doesn't nibtool understand my objects.nib file?

  Don't run nibtool against the object.nib file. Run it against the enclosing .nib directory. objects.nib, objects.xib, classes.nib, and info.nib are internal data structures packaged in a document wrapper.

- How can I localize my nib with nibtool?

  Use the following procedure:

  1. Generate the strings file for a particular nib file, using the following command: `nibtool -L myfile.nib > file.strings` file.strings will now contain entries such as "key" = "key" and be in Unicode (UTF-16) format.

  2. Give the resulting strings file to a translator and have it convert the second "key" entry to "key in other language".

  3. Reprocess the foreign nibs using the converted strings file: `nibtool -d file.strings -w newLocalization.nib myfile.nib` nibtool will take the contents of file.strings and replace "key" in myfile.nib with "something in other language".

- I have already translated my nib file and adjusted the sizes of my widgets so the text doesn't clip. But now I have to add another widget to the development nib. When I translate this nib again, will I loose all of my size changes to the localized nib? How do I get the translation to maintain the size from the old nib file?

In this example, let's say we're translating an English nib to an Italian localization. You need to get the file.strings file and the Italian.nib file and use nibtool to generate a newItalian nib file: `nibtool -d file.strings -I Italian.nib -w newItalian.nib English.nib` nibtool uses the file.strings file for the translation and Italian.nib for the size of objects. The resulting file newItalian.nib will have all of the objects' size matching Italian.nib.

Document Revision History

This table describes the changes to *Interface Builder*.

Date	Notes
2007-04-05	This document describes an older version of the Interface Builder application, which is not supported in Mac OS X v10.5 and later.
2003-08-21	Incorporated review comments
2003-08-07	First draft