Cocoa Application Tutorial Using Bindings

Cocoa > Design Guidelines

2007-07-10

Contents

Introduction **Introduction to Developing Cocoa Applications Using Bindings: A Tutorial 7**

Chapter 1 **Creating the Project and Interface 9**

Create a New Project 9
Build the User Interface 11

Chapter 2 **Creating the Model 13**

Create the Models 13

Chapter 3 **Creating an Object Controller 15**

Add a Controller 15
Establish the Model-Controller Relationship 15

Chapter 4 **Binding Views to Controllers 19**

Binding the Views to the Controller 19

Chapter 5 **Running the Application 21**

Build and Run 21

Document Revision History 23

Figures

Chapter 1 **Creating the Project and Interface 9**

Figure 1-1 Select the Cocoa Application project type 10
Figure 1-2 Controllers palette 10
Figure 1-3 Bindings pane 11
Figure 1-4 Final Currency Converter application 12
Figure 1-5 Adding numeric formatters to the text fields 12

Chapter 3 **Creating an Object Controller 15**

Figure 3-1 Create an instance of `NSObjectController` 15
Figure 3-2 Connect the controller's content outlet 16
Figure 3-3 Attributes pane for the `NSObjectController` instance 17
Figure 3-4 Attributes pane for the `NSObjectController` instance with keys 17

Chapter 4 **Binding Views to Controllers 19**

Figure 4-1 Bindings pane for Exchange Rate text field 19
Figure 4-2 Value binding for Exchange Rate text field with settings 20

Introduction to Developing Cocoa Applications Using Bindings: A Tutorial

This tutorial takes you through the steps of building the familiar Currency Converter application using Cocoa bindings. You might already be familiar with the Currency Converter example, which is the basis for *Cocoa Application Tutorial*. This article does not repeat fundamental Cocoa programming concepts and does not provide explicit instructions for common operations (such as applying number formatters to user interface objects).

Creating the Project and Interface

This part of the tutorial guides you through building the Currency Converter application and in the process teaches you the steps essential to building a Cocoa application using Cocoa bindings.

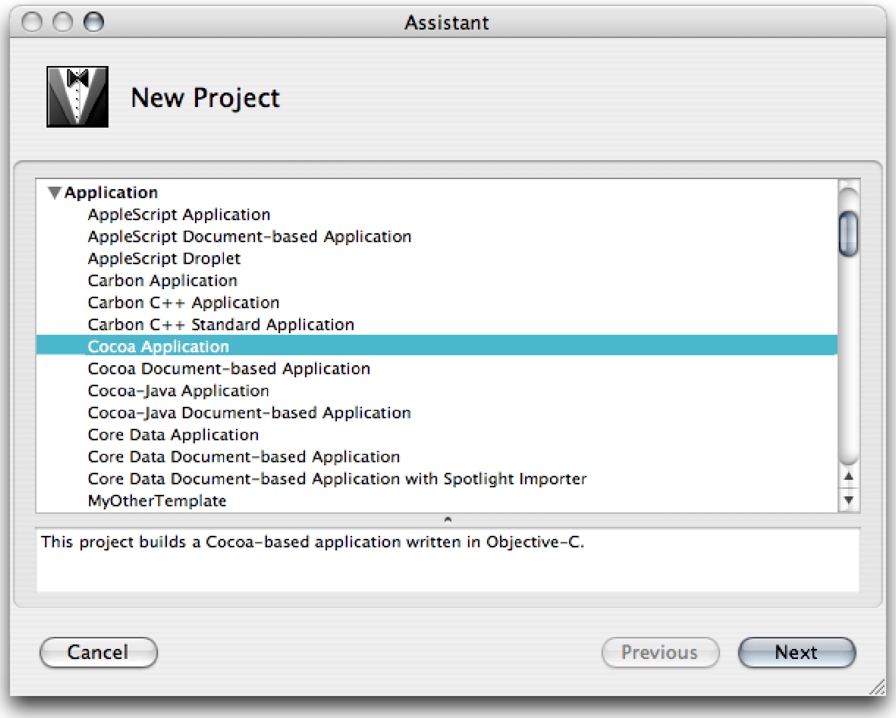
Create a New Project

Although the application you'll build in this tutorial is too simple to demonstrate the full power of Cocoa bindings, its simplicity will help you immediately grasp the potential of Cocoa bindings.

Cocoa bindings is integrated into the Cocoa framework, so any Cocoa application can use it. The Currency Converter application you'll build is a Cocoa document-based application. Follow these steps to create the initial project:

1. Launch the Xcode application, located in `/Developer/Applications`.
2. Choose New Project from the File menu.
3. Select Cocoa Application in the Xcode Project Assistant, and click Next (as shown in Figure 1-1).
4. Enter the project name (for example, enter "CurrencyConverter") and a destination folder for the project.
5. After creating the project, open `MainMenu.nib` in Interface Builder by double-clicking its icon in the project Resources folder.

Figure 1-1 Select the Cocoa Application project type

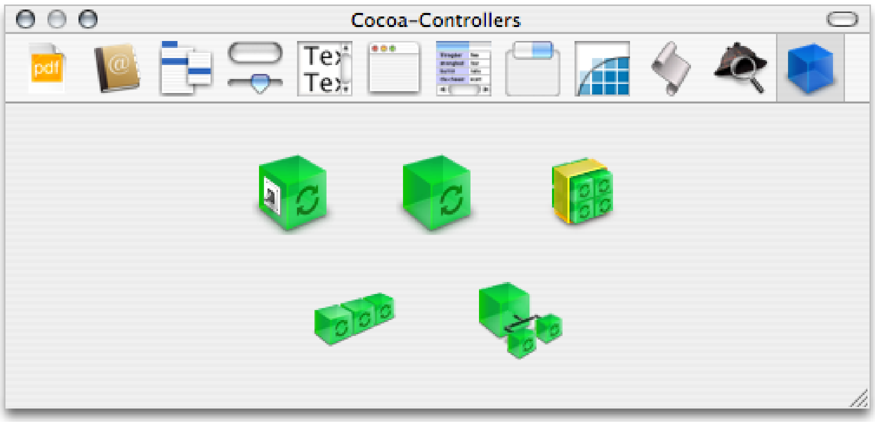


▼ Application
- AppleScript Application
- AppleScript Document-based Application
- AppleScript Droplet
- Carbon Application
- Carbon C++ Application
- Carbon C++ Standard Application
- Cocoa Application
- Cocoa Document-based Application
- Cocoa-Java Application
- Cocoa-Java Document-based Application
- Core Data Application
- Core Data Document-based Application
- Core Data Document-based Application with Spotlight Importer
- MyOtherTemplate

This project builds a Cocoa-based application written in Objective-C.

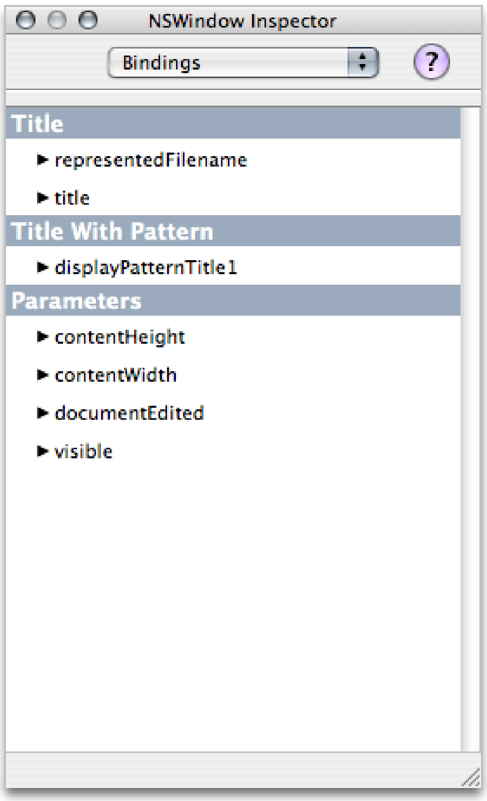
Interface Builder contains a palette—called Controllers, shown in Figure 1-2 —that contains the bindings controllers you can drag to your nibs.

Figure 1-2 Controllers palette



The Bindings pane in the inspector allows you to view the bindings for a selected object. The contents of the Bindings pane changes depending on the object selected in Interface Builder. Figure 1-3 shows the pane when the nib file's main window is selected.

Figure 1-3 Bindings pane



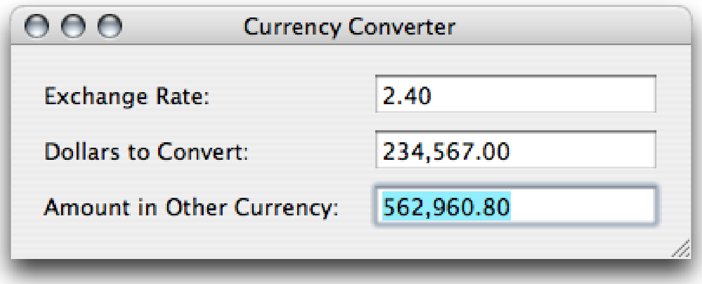
Build the User Interface

It's common to begin developing a Cocoa application by prototyping the application's user interface. Cocoa bindings complements this development approach by helping you build more full-featured prototypes than ever before.

The final Currency Converter user interface for this tutorial is shown in Figure 1-4. Based on the value of the Exchange Rate and Dollars to Convert fields, the application computes and updates the value displayed in the Amount in Other Currency field.

Notice that unlike the traditional Currency Converter, the application doesn't include a Convert button. Rather, it uses the infrastructure of Cocoa bindings to automatically perform the conversion without requiring the user to click a button.

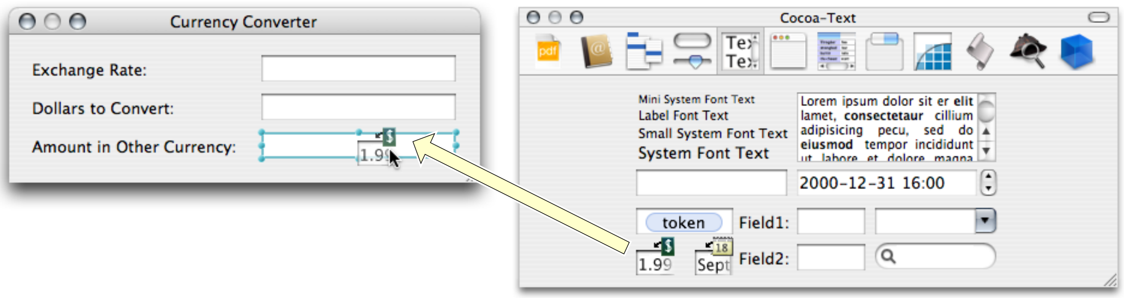
Figure 1-4 Final Currency Converter application



Follow these steps to build this user interface:

1. Set the title of the application window to `Currency Converter`.
2. Drag three text fields and three labels from the Text palette.
3. Change the three labels to "Exchange Rate", "Dollars to Convert", and "Amount in Other Currency".
4. Arrange the objects and change their labels to match Figure 1-4 (page 12).
5. Add a numeric formatter to each of the text fields, as shown in Figure 1-5.

Figure 1-5 Adding numeric formatters to the text fields



Creating the Model

Create the Models

Next you need to implement your application's model objects. The Currency Converter application solves a simple problem in a well-defined problem space, so the requirements of its model objects are rather simple: Based on the user's input into the Exchange Rate and Dollars to Convert text fields, compute a value representing the amount in the other currency.

To do this, the application's model object needs to know about the values in the Dollars to Convert and Exchange Rate fields. But unlike traditional Cocoa applications, you don't need to write a special controller class that mediates between the model and views. That is, Cocoa bindings eliminates the need to reference a nib file's user interface objects by `IBOutlet`, to manually extract values from them, and to manually set values in them. Rather, Cocoa bindings uses a more abstract mechanism to get and set values in model objects: key-value coding.

Any subclass of `NSObject` can be a model. You can use Interface Builder to create a subclass of `NSObject` as follows:

1. Click Classes in the `MainMenu.nib` window.
2. Select `NSObject` and choose Classes > Subclass NSObject.
3. Change the name of the new class to "Converter".
4. While the `Converter` class is selected in the `MainMenu.nib` window, choose Create Files for Converter from the Classes menu.

All you need to do to make this class participate in Cocoa bindings is add the necessary instance variables and accessor methods to the `Converter` class as follows:

1. Add the following declarations to `Converter.h`:

   ```
   /* Converter */

   #import <Cocoa/Cocoa.h>

   @interface Converter : NSObject
   {
       double dollarsToConvert;
       double exchangeRate;
   }

   - (double)amountInOtherCurrency;

   @end
   ```

2. Add the following method implementation to `Converter.m`.

```
- (double)amountInOtherCurrency{
    return (double)(dollarsToConvert * exchangeRate);
}
```

As you can see from above code, the value of `amountInOtherCurrency` is derived, and simply setting the values of `dollarsToConvert:` or `exchangeRate:` doesn't recompute this value. So how do we trigger the invocation of the `amountInOtherCurrency` method?

Cocoa bindings provides a class method you can use to trigger a change notification to update a dependent key. For example, you can set up a dependency between changing the values of the `dollarsToConvert` and `exchangeRate` keys to trigger an update of the value of the `amountInOtherCurrency` key. You typically set up this dependency in the `initialize` class method belonging to the model class:

1. Implement the `Converter +initialize` method:

```
+ (void)initialize {
    [Converter setKeys:
        [NSArray arrayWithObjects:@"dollarsToConvert", @"exchangeRate", nil]
        triggerChangeNotificationsForDependentKey:@"amountInOtherCurrency"];
}
```

Finally, add your model, an instance of `Converter`, to the nib file:

1. Click the classes tab in the `MainMenu.nib` window in Interface Builder.
2. Select the `Converter` instance and choose Instantiate Converter from the Classes menu.

Creating an Object Controller

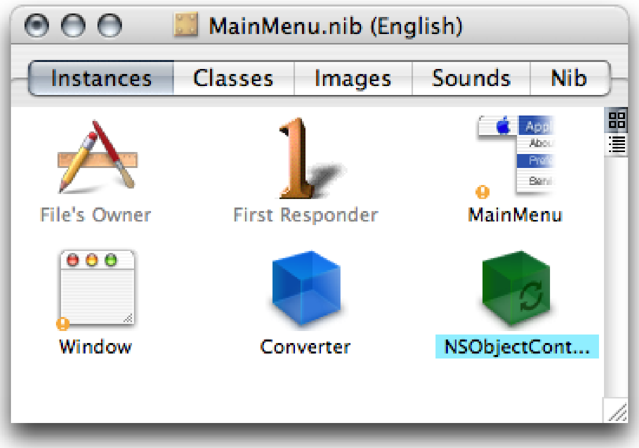
Add a Controller

To start using Cocoa bindings, you need to add a controller object to the nib file. This controller object is associated with the application's model object and provides data to the application's view objects. As Figure 1-2 (page 10) shows, the Controller palette provides various controller objects, including NSObjectController, NSArrayController, and NSUserDefaultsController. Each controller provides specialized functionality: NSObjectController manages a single object, NSArrayController manages an array of objects, and NSUserDefaultsController interacts with the user defaults database.

The Currency Converter application has a single model object, so NSObjectController is the appropriate choice. You add an object controller as follows:

1. Drag NSObjectController from the Controller palette to the nib file window, which should then appear as shown in Figure 3-1.
2. Optionally, rename the controller by double-clicking its label in the `MainMenu.nib` window, and entering a new name. For this example, the object controller's name is left unchanged.

Figure 3-1 Create an instance of `NSObjectController`



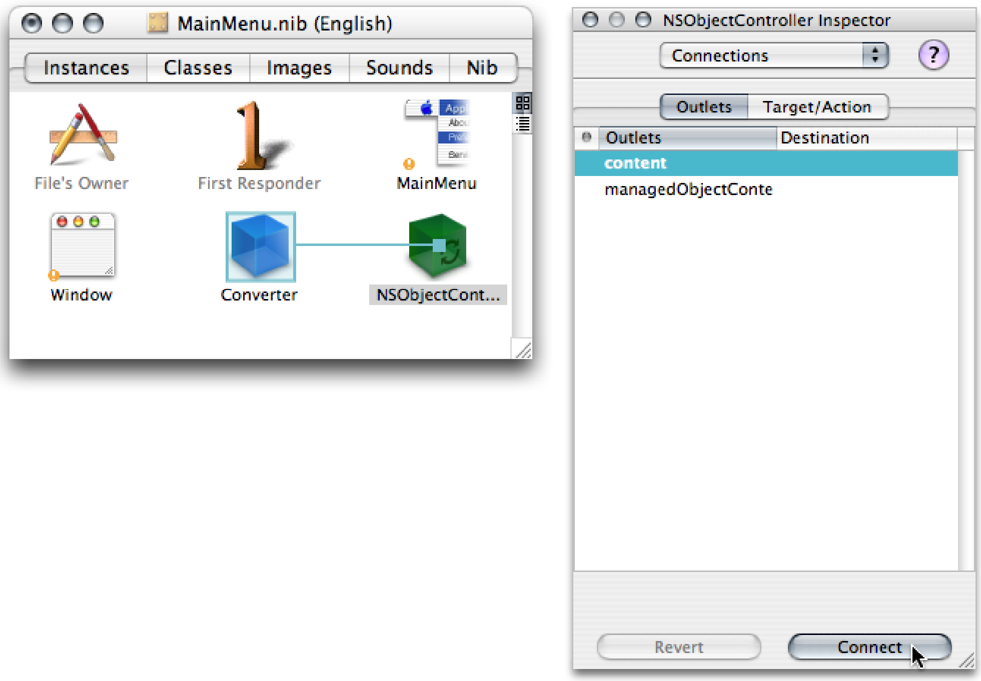
Establish the Model-Controller Relationship

Now that you've instantiated an object controller by dragging it from the Controller palette, you need to associate it with a model object by specifying which model object it controls. You can connect the controller to the model object, the instance of Converter you created above, as follows:

1. Control-drag a connection from the object controller to the `Converter` object in the `MainMenu.nib` window.

2. Connect the controller's `content` outlet, as Figure 3-2 shows.

Figure 3-2 Connect the controller's content outlet



There are a number of ways to set a controller's content, the easiest of which is to set its `content` outlet, as you did here. When you set a controller's `content` outlet to another object, you are telling the controller to use the data that object provides and also to provide that object with data. You are telling the controller to invoke key-value coding operations on the target of the content outlet. Controllers provide a two-way binding between model and view objects by tying together the data represented by both so that changes to the data in either object changes the data in the other object.

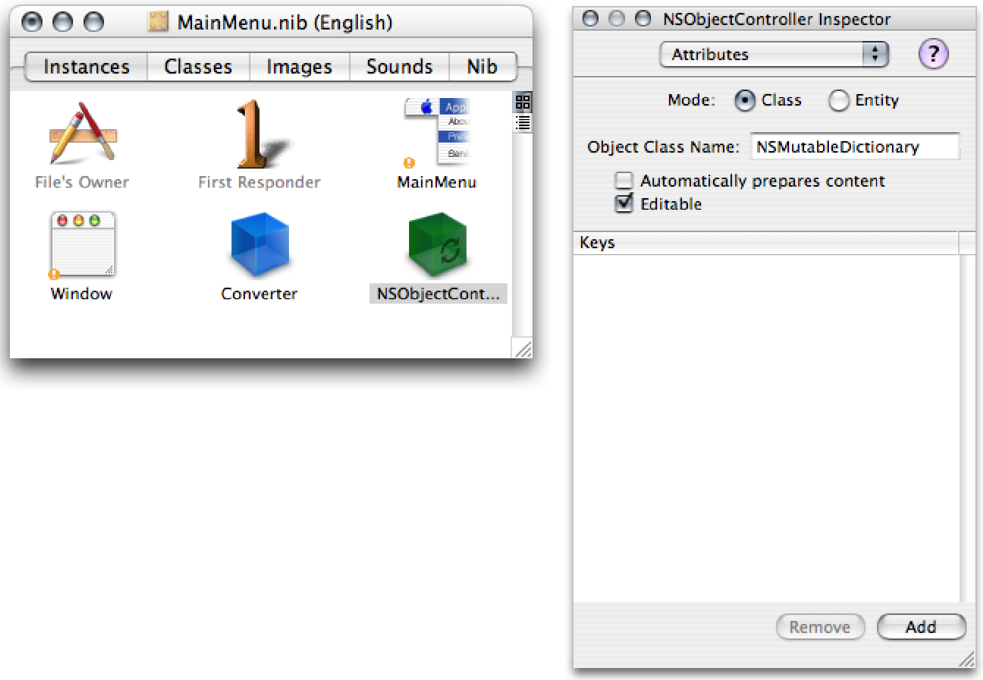
Now that you've instantiated a controller and associated it with the application's model object, you need to help the controller identify the keys in the model object with which it interacts. Simply setting a controller's `content` outlet to a model object does not provide the controller with enough information to know how to access the data in the model object. The most important attributes of a controller are the model keys it exposes to the views. These keys match the keys in the model class with which the controller is associated. You expose model keys as follows:

1. Select the object controller in the nib file window and choose the Attributes pane in the inspector, as shown in Figure 3-3.

2. Add three keys to the object controller by clicking the Add button: `dollarsToConvert`, `exchangeRate`, and `amountInOtherCurrency`.

3. The pane in Figure 3-3 should now look like Figure 3-4.

Figure 3-3 Attributes pane for the `NSObjectController` instance

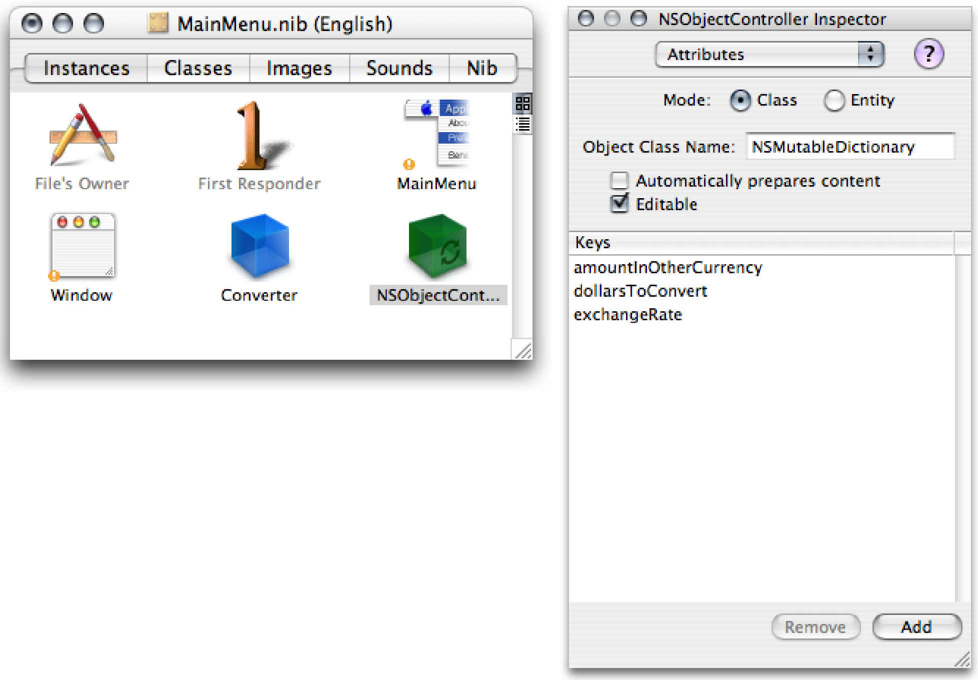


Mode: [✓ Class] Entity

Object Class Name: NSMutableDictionary

☐ Automatically prepares content
☑ Editable

Figure 3-4 Attributes pane for the `NSObjectController` instance with keys



Mode: [✓ Class] Entity

Object Class Name: NSMutableDictionary

☐ Automatically prepares content
☑ Editable

Keys
amountInOtherCurrency
dollarsToConvert
exchangeRate

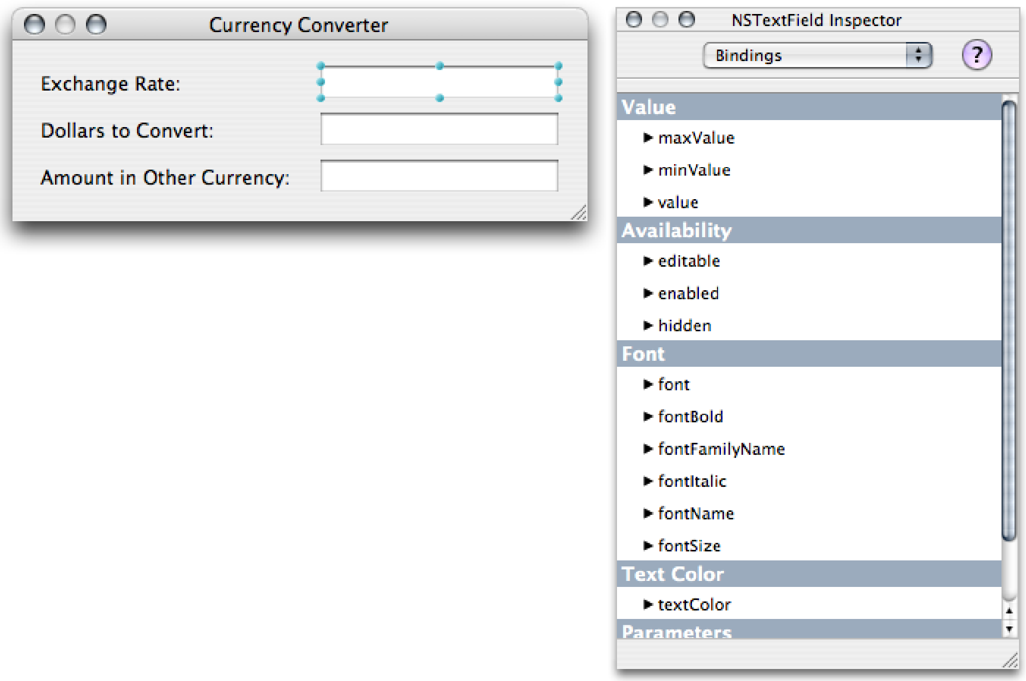
Binding Views to Controllers

Binding the Views to the Controller

Now that you have connected the controller and the model, and exposed the desired model keys, you can start binding user interface elements to the controller.

To bind a user interface element, simply select it and display the Bindings pane in the inspector. Figure 4-1 shows the Bindings pane as it appears when you select the Exchange Rate text field.

Figure 4-1 Bindings pane for Exchange Rate text field



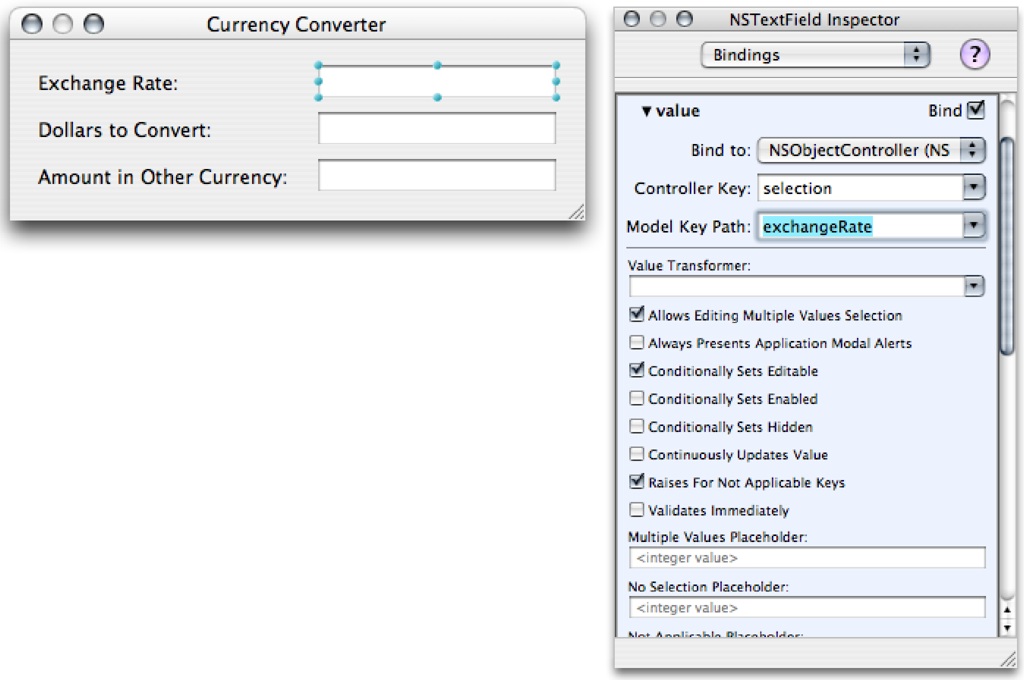
The Bindings pane organizes individual bindings into binding groups. Different user interface elements have different sets of available bindings. The bindings available to a window, for example, differ in part from those available to a text field.

Every binding provided by Cocoa bindings includes at least three aspects: Bind to, Controller Key, and Model Key Path. The Bind to aspect identifies the controller to which the binding is attached. The Controller Key aspect specifies the access point in the controller object that the binding uses to get and set data. The Model Key Path aspect specifies the key path in the model object to which the binding applies.

In this example, you need to bind the text fields to the corresponding model keys. Follow these steps to bind each text field:

1. Select the text field, display the Bindings pane in the inspector, and disclose the `value` binding.
2. Choose NSObjectController or whatever name was given to the controller from the Bind to menu.
3. Choose `selection` from the Controller Key menu.
4. Choose the appropriate model key from the Model Key Path menu as shown in Figure 4-2.

Figure 4-2 Value binding for Exchange Rate text field with settings



▼ value — Bind ✓

Bind to: NSObjectController (NS

Controller Key: selection

Model Key Path: exchangeRate

Value Transformer:

- ✓ Allows Editing Multiple Values Selection
- ☐ Always Presents Application Modal Alerts
- ✓ Conditionally Sets Editable
- ☐ Conditionally Sets Enabled
- ☐ Conditionally Sets Hidden
- ☐ Continuously Updates Value
- ✓ Raises For Not Applicable Keys
- ☐ Validates Immediately

Multiple Values Placeholder: <integer value>

No Selection Placeholder: <integer value>

This concrete example should help you understand what the `value` binding's configuration implies: The content of the text field is bound to the value of the `exchangeRate` key, which Cocoa bindings finds on the model object using key-value coding. Cocoa bindings knows to look for that key in the model object that is bound to the `content` outlet of the controller specified by the Bind to aspect—in this example, the controller is the `NSObjectController` instance you configured earlier, whose `content` outlet points to the `Converter` object you instantiated in the nib file. The Controller Key aspect specifies the access point in the controller on which the binding acts.

Array controllers, for example, manage an array of objects. In some contexts, such as when binding the contents of a table view, you may want to access all the objects an array controller manages, which it provides through its `arrangedObjects` controller key. In other contexts, such as when displaying the value of a cell in the table row a user selected, you may be concerned with a single object in the array, which the array controller provides through its `selection` controller key.

Running the Application

Build and Run

When you build and run the application and you should have a fully functional currency converter that resembles Figure 1-4 (page 12).

Users enter data in the Dollars to Convert and Exchange Rate text fields, and when they leave either of those fields, the views send messages to controller that cause Cocoa bindings to attempt to set the value of the `dollarsToConvert` and `exchangeRate` instance variables and also to get the value of the `amountInOtherCurrency` method because a dependency was established between these model keys in "Creating the Model" (page 13).

So what is happening behind the scenes? In the running application, when a user enters data in the Dollars to Convert text field, Cocoa bindings attempts to use that value to set the value of the data member to which the text field's value aspect is bound—the `dollarsToConvert` model key. It does this by invoking the key-value coding accessor `setValue:forKey:`.

At exactly what point does Cocoa bindings invoke `setValue:forKey:`? It invokes it when the text field sends an `endEditing:` message. If you configure the text field to continuously update its value by selecting its `value` binding's Continuously Updates Value option, `setValue:forKey:` is invoked each time the text field receives a keystroke.

Since bindings are bidirectional, is it possible for each end of the binding (meaning the model and view) to provide conflicting values? No, because when the text field sends an `endEditing:` message, key-value coding takes over and invokes `setValue:forKey:`. When that operation finishes, key-value observing takes over to display the value of the key to which the text field is bound. So, the value displayed in a bound user interface object is ultimately resolved to the result of key-value observing operations.

Note that unlike the traditional Currency Converter application, you didn't need to add any IBOutlets to the application's model class, manually get the values the user enters, or invoke `setStringValue` on any of the user interface's text fields to populate their values. All this kind of glue code is handled for you automatically by Cocoa bindings.

Document Revision History

This table describes the changes to *Cocoa Application Tutorial Using Bindings*.

Date	Notes
2007-07-10	Corrected minor typos.
2007-02-08	Corrected figure titles in "Binding Views to Controllers".
2006-10-03	Corrected value display in example. Updated for Xcode 2.4.
2006-04-04	Corrected minor typos; changed title from "Developing Cocoa Applications Using Bindings: A Tutorial."
2004-08-22	Corrected figure in "Binding the Views to the Controller" (page 19).
	Clarified the description of the Controller palette and Bindings inspector in "Creating the Project and Interface" (page 9).
2004-04-22	Initial release.