INSIDE COCOA

Developing Cocoa Java Applications: A Tutorial

Preliminary

May 2001

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

Simultaneously published in the United States and Canada

Contents

CHAPTER 1

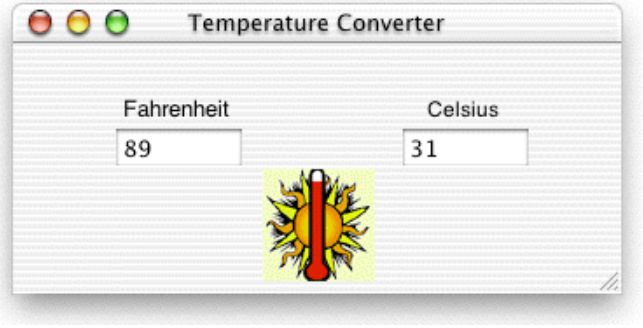
Introduction

This tutorial introduces the Cocoa application framework of Mac OS X, and teaches you how to leverage Java to build robust, object-oriented applications using Apple's Mac OS X tools. Cocoa provides the best way to build modern, multimedia-rich, object-oriented applications for consumers and enterprise customers alike. This tutorial assumes a working knowledge of the Java language and concepts, but does not assume any previous experience with Cocoa, Project Builder, or Interface Builder.

What You'll Learn in This Tutorial

In this tutorial you will build a simple application that converts temperature values between Celsius and Fahrenheit. The application will display a different image depending on the temperature range. Here's what the finished application looks like:

Figure 1-1 Finished Temperature Converter



The tutorial has three chapters, which you should complete in order:

1. "Building a Simple Application" (page 9). Explains how to create a project and a graphical user interface, define a custom controller class, and connect an instance of that class to other objects in the application. It also shows how you must change the source code files generated by Interface Builder to be valid Java files.
2. "Creating a Custom View Class" (page 33). Shows how to create a custom view object using Interface Builder and Project Builder. (The procedure varies from that for controller classes.)
3. "Debugging Java Applications" (page 55). Illustrates the use of Project Builder and its JavaDebug facility to debug Cocoa Java applications. It also shows how you can debug projects that contain both Java code and Objective-C code.

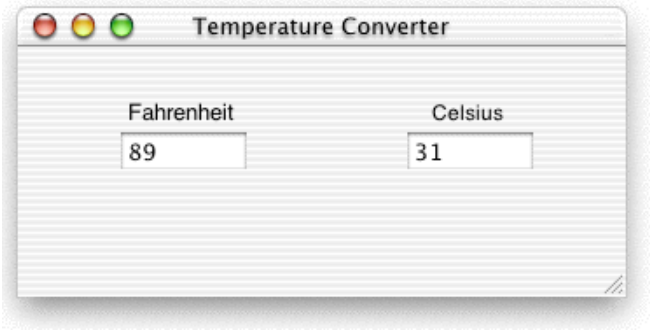
Building a Simple Application

This part of the tutorial guides you through the creation of a very simple application and in the process teaches you the steps essential to building a Cocoa application using Java. The tasks in this chapter include:

1. "Creating a Project" (page 10)
2. "Creating the Interface" (page 14)
3. "Implementing the Controller Class" (page 28)
4. "Connecting Objects" (page 22)
5. "Defining the Controller Class" (page 19)
6. "Building and Running the Application" (page 31)

By the end of this chapter you will have created an application called Temperature Converter that looks like this:

Figure 2-1 Temperature Converter



This application does exactly what its name suggests, converting Celsius values to Fahrenheit, and vice versa. The user just types a value in one of the text fields and presses the Return key. The converted value is shown in the other field.

Although this application is simple, the experience of putting it together will clarify a few central techniques and paradigms in Cocoa Java development, among them:

- Creating graphical user interfaces with Interface Builder
- Defining custom Java controller classes with Interface Builder
- Connecting an instance of the controller class with other objects in the application
- Generating source files from Interface Builder definitions and modifying those files to be suitable for Java compilation
- Building the project, after writing the necessary Java code

Creating a Project

Cocoa projects contain source code files, user interface files (nib files), and application property settings (plists). Cocoa projects also link to application frameworks (`Cocoa.framework`, `AppKit.framework`, and `Foundation.framework`) and

other libraries. Project Builder creates a folder structure for your project, links your project to the system frameworks, and sets the default property list settings depending on the type of project you choose.

Launch Project Builder

Project Builder is the application typically used for creating and managing development projects which use the Cocoa framework. To launch Project Builder:

1. Find Project Builder in `/Developer/Applications`.
2. Double-click the icon in the Finder.

Choose the New Project Command

When Project Builder is launched, only its menus appear. To create a project, choose File > New Project. This action causes the New Project pane to appear.

Select Project Type

Project Builder can build many different types of applications, including everything from Cocoa Java applications to Mac OS X kernel extensions and Mac OS X frameworks. For this tutorial, select Cocoa-Java Application and click Next.

Figure 2-2 Interface Builder's New Project assistant

1. Using the file-system browser, navigate to the directory where you want your project to be.
2. Type the name of the project in the Project Name field. For the current project, type the name `Temperature Converter`.
3. Click Finish.

Figure 2-3 Specifying your project's location

When you click Finish, Project Builder creates and displays a project window.

You might want to look in the project directory to see what kind of files it now contains. Among the project files are:

`English.lproj/`
: A directory containing resources localized to your preferred language. In this directory are nib files automatically created for the project. For more information on nib files, see the developer documentation for Interface Builder.

`main.m`
: A file, generated for each project, that contains the entry-point code for the application in `main()`. Usually, this file is not edited by the developer.

`Temperature Converter.pbproj`

This contains information that defines the project, including the information property list (`InfoPlist.strings`) and project-specific preferences for Project Builder. It too should not be modified. You can open your project by double-clicking `Temperature Converter.pbproj`.

Creating the Interface

The process of developing a Cocoa application begins at the user interface level, rather than at the code level, and allows developers to create and connect objects visually. This allows for rapid application development while providing a genuine object-oriented development environment. Creating the interface includes these steps:

1. "Open the Main Nib File" (page 14)
2. "Resize the Window" (page 15)
3. "Rename the Window" (page 15)
4. "Put Text Fields in the Window" (page 17)
5. "Set Attributes of the Text Fields" (page 18)
6. "Add Labels for the Text Fields" (page 18)

Open the Main Nib File

Each application project, when first created, includes a blank nib file known as the **main nib file**. This nib file contains the application menu and perhaps one or more windows. Applications automatically load the main nib file when they are launched.

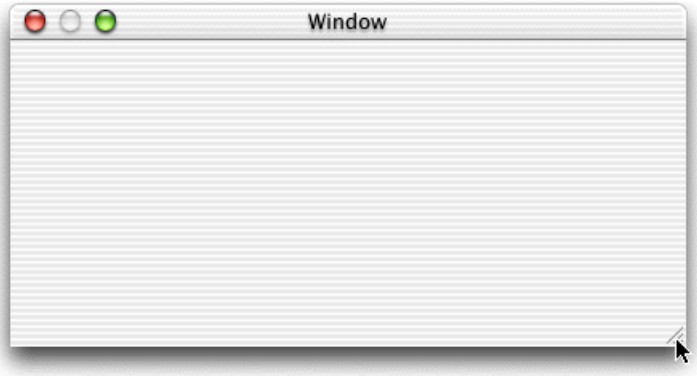
1. Locate the main nib file for your project. By default, Project Builder names the main nib file `MainMenu.nib`. Find this file in the Resources category in the Groups and Files pane of the project target.
2. Double-click `MainMenu.nib` to launch Interface Builder.

Note: To learn more about Interface Builder, see the Interface Builder reference.

Resize the Window

The window provided for you in the main nib file is too large. To resize a window, drag the lower right corner of the window in any direction (up, down, diagonal).

Figure 2-4 Resizing a window



You can resize a window to exact dimensions by entering pixel values in the Size pane of Interface Builder's info window (see "Place and Resize the CustomView Object" (page 34) in the second part of the tutorial for an example of how this is done).

Rename the Window

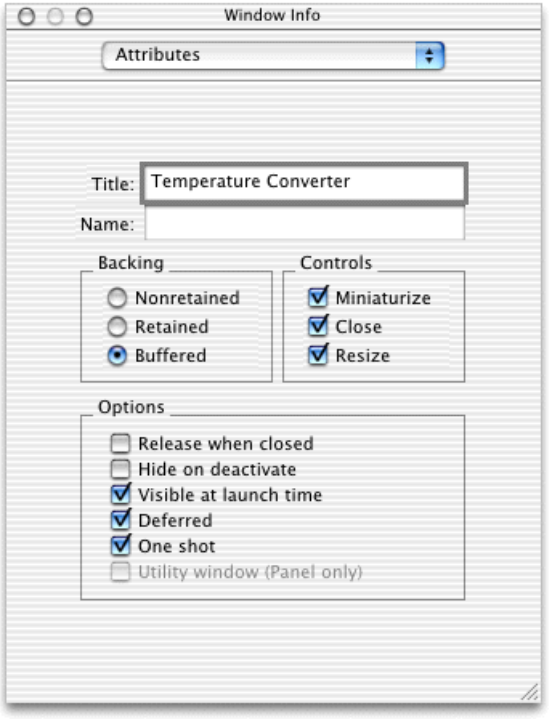
Windows usually carry distinctive titles in their title bars. In Cocoa, each window in a screen is based on an instance of `NSWindow`, which is analogous to the `Frame` class in Java. The title of a window is an attribute of this object. Interface Builder allows you to set the attributes of NSWindows and many other objects in its info window.

To set the title of the Temperature Converter window:

1. Select the window by clicking it.
2. Choose Tools > Show Info.

3. Choose Attributes from the pop-up menu (if it is not already chosen).
4. In the Attributes pane, enter `Temperature Converter` in the Title field, replacing "Window."

Figure 2-5 Renaming a window



5. Uncheck the Close and Resize checkboxes. These window attributes don't make sense with an application as simple as this.

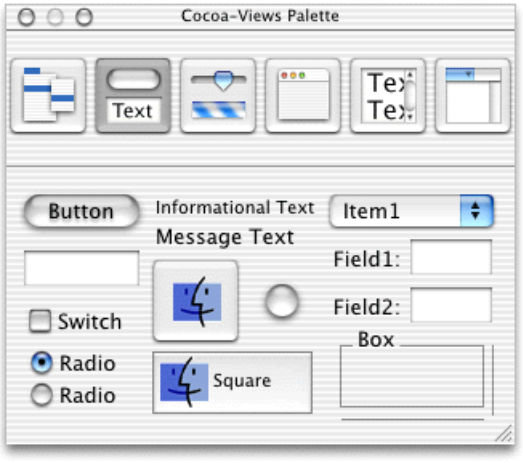
Don't worry about putting a value into the Name field.

Put Text Fields in the Window

Interface Builder's Palette window has several palettes full of Application Kit objects. The Views palette holds many of the smaller objects, among which is the text-field object. You now will add two text fields to the Temperature Converter interface:

1. Select the Views palette:

Figure 2-6 Cocoa views palette



2. Drag a text field object from the palette toward the Temperature Converter window.
3. Drop the object (by releasing the mouse button) in the window at one of the locations shown in Figure 2-1 (page 10).

   Once you drop the object, you can reposition it in the window by dragging it.
4. Repeat steps 2 through 4 for the other text field.

If you need to delete an object in the interface, just select it and press the Delete key.

Set Attributes of the Text Fields

Earlier you set an attribute of the Temperature Converter window (its title). Now you need to set an attribute of each of the text fields. All objects on Interface Builder's palettes have attributes that you can set through the info window.

1. Select a text field.
2. Choose Tools > Show Info.
3. Choose Attributes from the window's pop-up list, if it is not already selected.
4. Click on the radio button labeled "Only on enter" in the Send Action group.

Repeat this sequence for the other text field.

Note: You do not have to set the "Send Action—Only on enter" attribute. When this button is turned on, the text field sends its action message (which will be connected to the `convert()` method) only when the user presses the Enter or Return key while the insertion point is in the field. If what you want is the default behavior for text fields—the field sends its action message to its target whenever the insertion point leaves it—then leave the "Only on enter" radio button turned off.

Add Labels for the Text Fields

Text fields without labels would be confusing to users, so solve that problem by labeling each field.

1. Drag the Message Text object from the Views palette and drop it so it's just above the left text field.
2. Double-click the text to select all of it.
3. Type `Fahrenheit`.

Repeat the steps for the other label (Celsius).

Important
Occasionally you should save the nib file containing your work. Now is a good time: choose File > Save.

Apply Aqua Layout Guidelines

Interface Builder provides tools to help you build applications that are consistent with the Aqua user interface of Mac OS X. The complete Aqua layout guidelines are available in *Inside Mac OS X: Aqua Human Interface Guidelines*, available on the Apple developer web site.

You may have noticed the guides that appeared when you were adding the text fields and their labels. These guides help you position items on a view consistent to the Aqua guidelines.

Defining the Controller Class

Interface Builder not only lets you construct the user interface of an application from real objects stored on palettes, but lets you partially define a class in terms of its name, its superclass, its outlets, and its actions. This process involves three basic steps:

1. "Identify the Class and Its Superclass" (page 19)
2. "Specify the Outlets of the Class" (page 20)
3. "Specify the Action of the Class" (page 21)

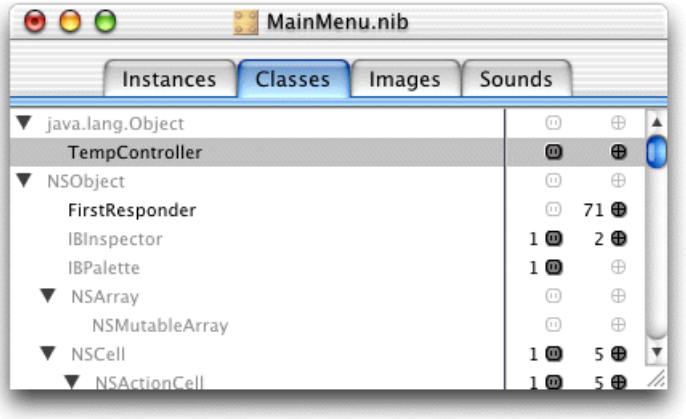
Identify the Class and Its Superclass

You define a custom class using the Classes menu and the Classes display of the nib file window.

1. Click the Classes tab of the `MainMenu.nib` file window.
2. Highlight `java.lang.Object` in the list of classes.
3. Choose Classes > Subclass.

   "MyObject" appears in an editable field below `java.lang.Object`.
4. Type `TempController` in place of "MyObject" and press Return.

Figure 2-7 Creating TempController



Specify the Outlets of the Class

An outlet is a reference one object holds to another object so that it can easily send that object messages; it is an instance variable of Objective-C type `id` or `IBOutlet`. The TempController class has two outlets, one to each of the text fields in the user interface.

1. Click the small electrical outlet icon to the right of TempController in the Classes display.

   The area under TempController expands to include Outlets and Actions.

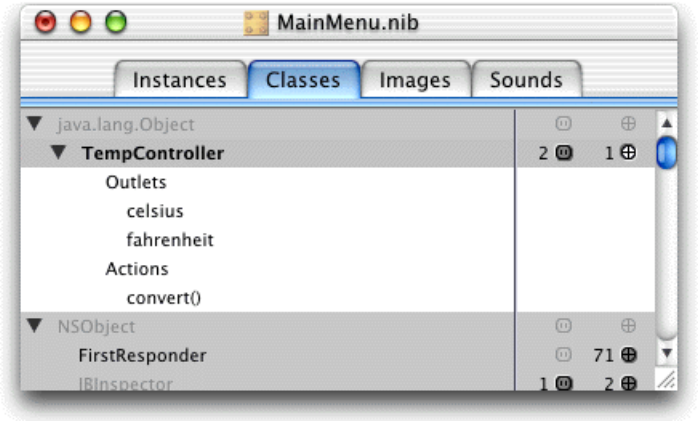
2. Choose Classes > Add Outlet.

   You can press the Return key instead of choosing the menu command.

3. Type `celsius` in place of “myOutlet”.

Repeat steps 2 and 3, this time naming the outlet `fahrenheit`.

Figure 2-8 Adding outlets and actions



To collapse the TempController item, click any other class in the Classes display.

Specify the Action of the Class

An action refers to a method invoked in a target object when a user event occurs, such as the click of a button or the movement of a slider. We want a method in TempController to be invoked whenever the user presses the Return key in a text field.

1. Click the small crosshairs icon to the right of TempController in the Classes display.

   The area under TempController expands to include Outlets and Actions.

2. Choose Classes > Add Action.

   You can press the Return key instead of choosing the menu command.

3. Type `convert` in place of "myAction" (parentheses are automatically appended to the method name).

See Figure 2-8 (page 21) for an example of what the Classes display looks like when you complete this task.

Connecting Objects

Interface Builder enables you to connect a custom object to its outlets and to the objects in the user interface that invoke action methods of the custom object. This connection information is stored in the nib file along with the user interface objects, class definitions, and nib resources. The process involves these steps:

1. "Create an Instance of the Controller Class" (page 22)
2. "Connect the Controller to its Outlets" (page 22)
3. "Connect the Action of the Controller" (page 24)
4. "Connect the Responders" (page 27)
5. "Test the User Interface" (page 28)

Create an Instance of the Controller Class

Before you can connect a custom object to objects in the user interface, you must create an instance of the object. (This is not a real instance, but a "proxy" instance representing the connections to the object. The real instance is created when the nib file is loaded.)

1. Select the TempController class in the Classes display of the nib file window.
2. Choose Classes > Instantiate.

The nib file window automatically changes to the Instances display, and an instance of the TempController class (depicted as a cube) appears in the display.

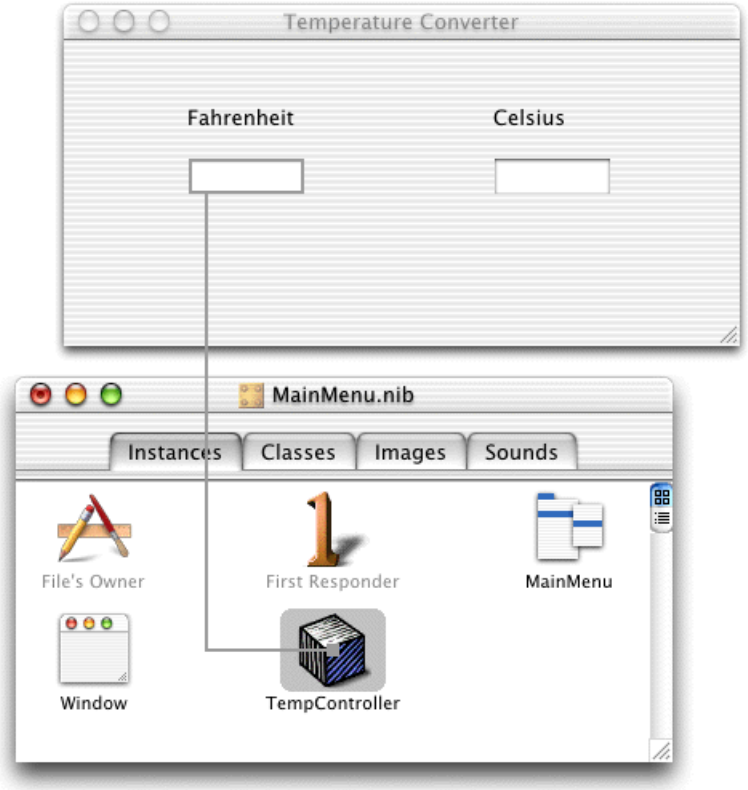
Connect the Controller to its Outlets

This procedure connects the TempController custom object to its outlets:

1. Control-drag from the cube representing the custom object to the Fahrenheit text field (the editable field, not the label). A thick black line follows the cursor while you drag.

"Control-drag" means to hold down the Control key while dragging the mouse (moving it with the mouse button pressed).

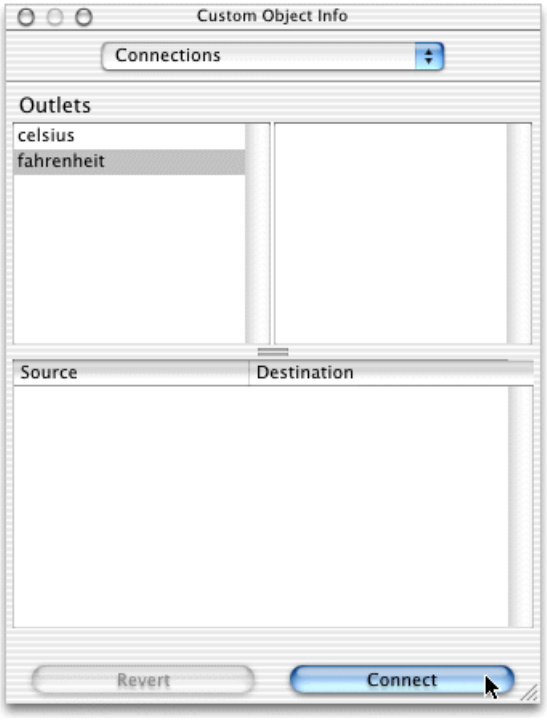
Figure 2-9 Control-dragging to an outlet



2. When a box encloses the Fahrenheit field, release the mouse button.

   Interface Builder shows the Connections pane in the info window. The left column lists the outlets defined by TempController.

Figure 2-10 Connecting to an outlet



3. Select the fahrenheit outlet.
4. Click the Connect button.

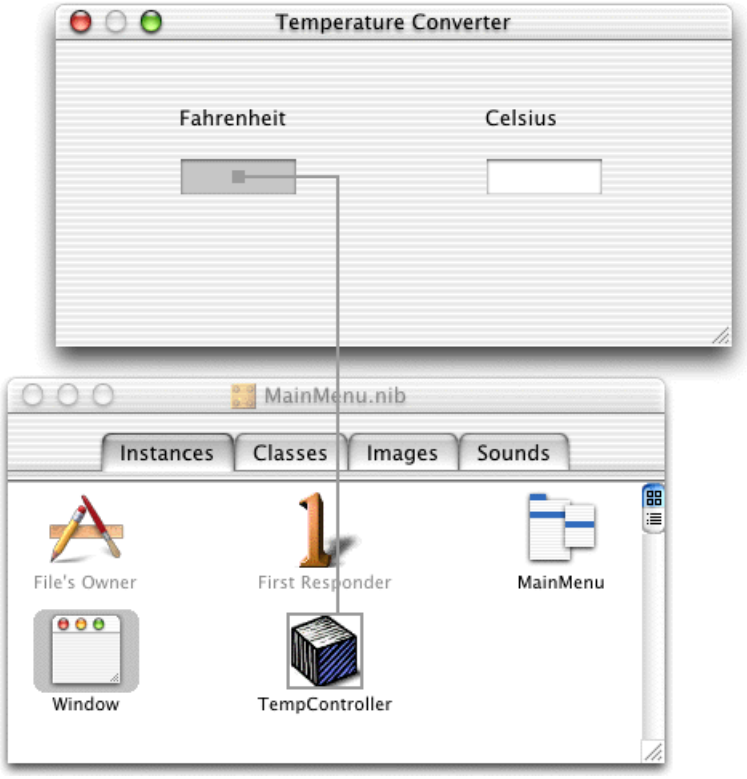
Repeat steps 1 through 4 for the celsius outlet.

Connect the Action of the Controller

This procedure connects the action method defined by TempController to the objects that might invoke that method:

1. Control–drag from the Fahrenheit field (the editable field, not the label) to the cube representing the custom object. A thick black line follows the cursor while you drag.

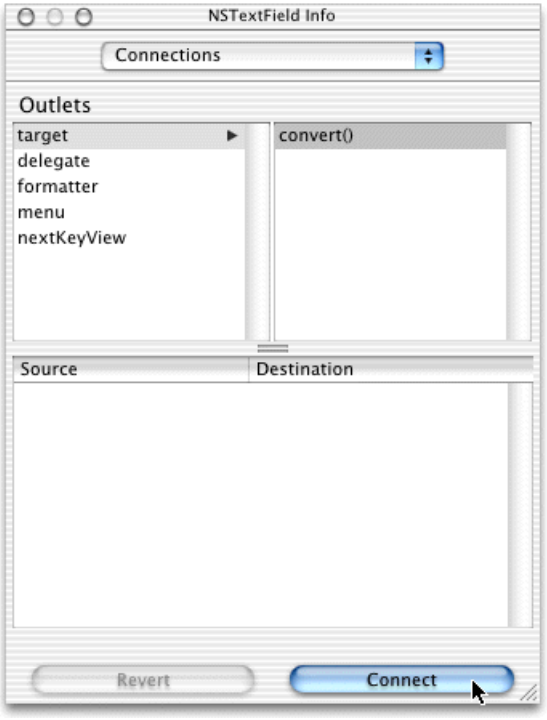
Figure 2-11 Control-dragging to an action



2. When a box encloses the cube, release the mouse button.

   Interface Builder shows the Connections pane in its info window. The right column lists the action defined by TempController.

Figure 2-12 Connecting to an action



3. Select the `convert()` action.

   You might first have to click the target item under Outlets to get to the action.

4. Click the Connect button.

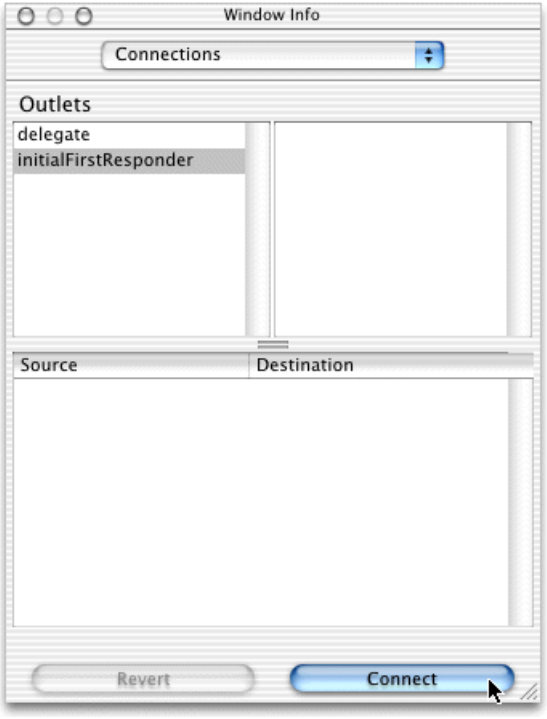
Repeat steps 1 through 4 for the Celsius field.

Connect the Responders

As a convenience to users, you want the insertion point to be in a certain field after the application is launched. For the same reason (convenience), you want users to be able to switch between the fields without having to use the mouse—they should be able to tab between the fields. You can specify this behavior entirely in Interface Builder:

1. Click the Instances tab of the nib file window.
2. Control–drag a connection line from the window icon to the Fahrenheit field.

Figure 2-13 Setting the initialFirstResponder



3. In the Connections pane of the info window, select initialFirstResponder.

4. Click the Connect button.
5. Control–drag a connection line from the Fahrenheit field to the Celsius field.
6. In the Connections pane, select nextKeyView and click Connect.
7. Control–drag a connection line from the Celsius field to the Fahrenheit field.
8. In the Connections pane, select nextKeyView and click Connect.

What have you just done? You've specified the sequence of responder objects in the user interface that are to receive the focus of keyboard events when users press the Tab key.

Test the User Interface

You can now test the user interface you've constructed with Interface Builder. Save the nib file and choose File > Test Interface. Interface Builder goes into test mode and the window and text fields you've just created behave as they would in the final application—except, of course, there is yet no custom behavior.

Notice that the insertion point is initially in the Fahrenheit field. Press the Tab key; note how the insertion point jumps between the fields. Type something into one of the fields, then select it and choose Edit > Cut. Click in the other text field and choose Edit > Paste. These are but a couple of examples of features you get in any application with little or no work on your part.

Quit from test mode to return to Interface Builder when you're done testing Temperature Converter.

Implementing the Controller Class

You're now ready to generate source-code template files from the nib file you've created with Interface Builder. After that, you'll work solely with the other major development application, Project Builder.

1. "Generate the Source Code Files" (page 29)
2. "Modify the Source Code Files" (page 29)

3. "Implement the convert Method" (page 30)

Generate the Source Code Files

To generate the source-code templates files for TempController, in Interface Builder:

1. Click the Classes tab in the nib file window.
2. Select the TempController class.
3. Choose Attributes from the info window and verify that the Java radio button is selected.
4. Choose Classes > Create Files.
5. Ensure that "TempController.java" is selected in the "Create Files" pane.
6. Ensure that "Temperature Converter" is selected in the "Insert into targets" pane.
7. Click Connect.

Interface Builder creates the file `TempController.java` and puts it in the Classes category of Project Builder. You can now quit Interface Builder (or, better still, hide it) and click in Project Builder's project window to bring it to the front.

Note: `TempController.java` may not have been put in the Classes category of Project Builder. Simply drag the file into this category.

Modify the Source Code Files

In Project Builder, perform the following steps to modify the generated files:

1. Select `tempController.java`.

   The following code is displayed in the code editor:

```
import com.apple.cocoa.application.*;
import com.apple.cocoa.foundation.*;

public class TempController {
    Object celsius;
    Object fahrenheit;
```

```
    public void convert(Object sender){
    }
}
```

2. Modify the above code so that it looks like this:

```
import com.apple.cocoa.application.*;
import com.apple.cocoa.foundation.*;

public class TempController {
    NSTextField celsius;
    NSTextField fahrenheit;
    public void convert(NSTextField sender) {
    }
}
```

Why is this modification necessary? Java is a strongly typed language and has no equivalent for the Objective-C dynamic object type `id`. When Interface Builder generates source-code files for Objective-C classes, it gives `id` as the type of outlets and as the type of the object sending action messages. This `id` is essential to the method signature for outlets and actions. However, when it generates Java source-code files, it substitutes the static Java type Object for `id`.

Implement the convert Method

Finally implement the convert method in Java, as shown here:

```
import com.apple.cocoa.application.*;
import com.apple.cocoa.foundation.*;

public class TempController {
    NSTextField celsius;
    NSTextField fahrenheit;
    public void convert(NSTextField sender) {
        if (sender == celsius) {
            int f = (int)((9.0/5.0 * celsius.intValue()) + 32);
            fahrenheit.setIntValue(f);
        } else if (sender == fahrenheit) {
            int c = (int)((fahrenheit.intValue()-32) * 5.0/9.0);
            celsius.setIntValue(c);
        }
    }
```

```
}
```

You can freely intermix Objective-C and native Java objects in the code. And you can use any Java language element, such as the try/catch exception handler.

Building and Running the Application

You've completed the work required from you for the Temperature Converter project. Now it's Project Builder's turn to work.

1. "Build the Project" (page 31)
2. "Launch and Test the Project" (page 32)

Build the Project

To build the project:

1. Click the Build icon in the project window to build your project.

Figure 2-14 Build icon

You can also press Command-B to build your project.

Project Builder begins compiling and linking the project code. It reports progress in the Build pane. If there are errors, Project Builder lists them in the upper part of the display area. Click a line reporting an error to have Project Builder scroll to the site of the error in the code editor.

Launch and Test the Project

Of course, once the application has been built, you'll want to launch the application to see if it works as planned. You have at least two ways of doing this:

- Locate the file `Temperature Converter.app` in the project directory. Double-click this file to launch the application.
- Click the Launch icon on the project window to run Temperature Converter.

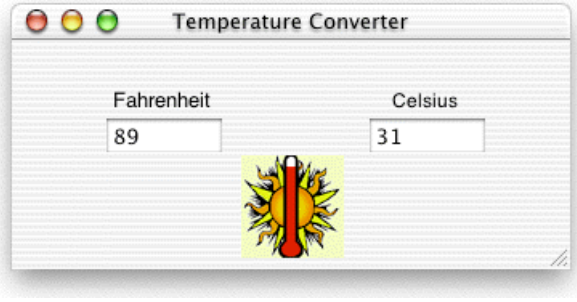
Figure 2-15 Launch icon

Creating a Custom View Class

This section of the tutorial describes the basic steps for creating a custom view and, more specifically, shows how to create a subclass of an Application Kit class that inherits from NSView. In this section, you will add a custom "image view" to the user interface you created in the first part of this tutorial. This custom object will respond to messages from the controller object, TempController, and change its image depending on the temperature entered.

Here's what the final Temperature Converter application will look like:

Figure 3-1 Final Temperature Converter



Major Tasks:

1. "Defining the Custom View Subclass" (page 34)
2. "Connecting the View Object" (page 38)
3. "Implementing the Custom View" (page 40)

4. “Completing the Application” (page 46)
5. “Creating a Subclass of NSView” (page 48)

The behavior that your custom view object adds to its superclass, NSImageView, is trivial. You could just as well accomplish the same behavior by sending messages to an “off-the-shelf” instance of NSImageView. But the subclass illustrates the basic procedure for making subclasses of Cocoa classes that don’t inherit from java.lang.Object.

“Creating a Subclass of NSView” (page 48) summarizes the procedure and provides example code for creating a subclass of NSView whose instances can draw themselves. This example subclass can replace the one you will create in this section, because it draws graphical shapes instead of displaying images when the temperature changes to another range.

Defining the Custom View Subclass

1. “Place and Resize the CustomView Object” (page 34)
2. “Specify the Subclass” (page 36)
3. “Assign the Class to the CustomView” (page 37)

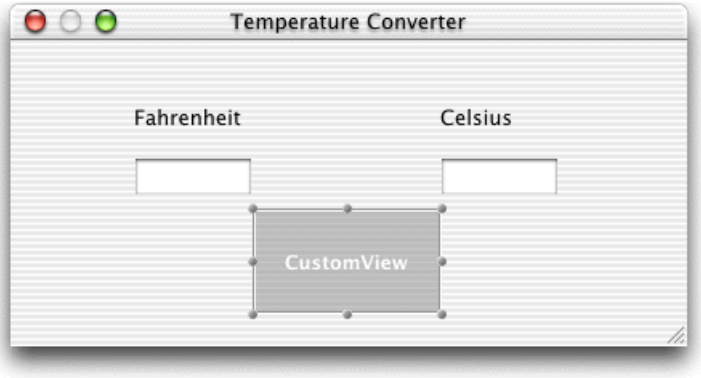
Place and Resize the CustomView Object

The CustomView object on Interface Builder’s Views palette represents an instance of any custom subclass of NSView or of any Application Kit class that inherits from NSView. The CustomView object lets you specify the basic attributes of all view objects: their location in a window and their size.

1. Drag the CustomView object from the Views palette and drop it in the window.

   Center it in the window beneath the text fields.

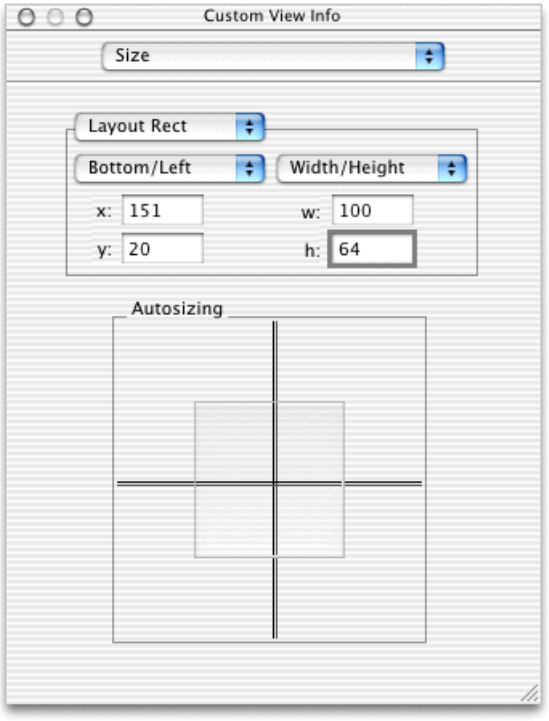
Figure 3-2 Placing a CustomView



2. Resize the CustomView using the Size pane.

   Choose Show Info from the Tools menu, select the Size pane, and enter 100 for the width (w) field and 64 for the height (h) field. You may have to select Width/Height from the right-most pull-down menu.

Figure 3-3 Resizing Custom View using Info window



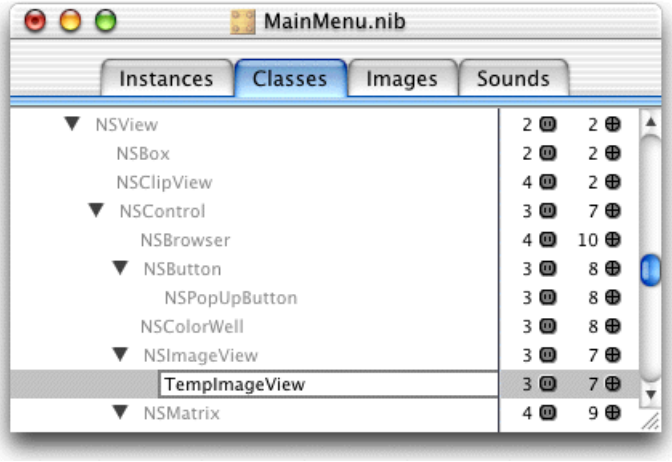
Specify the Subclass

As you did earlier with the controller object TempController, you must provide the name and superclass of your custom class. But now, instead of inheriting from java.lang.Object, your class inherits from an Application Kit class.

1. In the Classes display of the nib file window, select the NSImageView class.

   If a class in the display has a triangle next to it, you can click the triangle to reveal the subclasses of that class. The path you want to follow is this: NSObject, NSResponder, NSView, NSControl, NSImageView.

Figure 3-4 Subclassing NSImageView



2. Choose Classes > Subclass.
3. Name the class TempImageView.

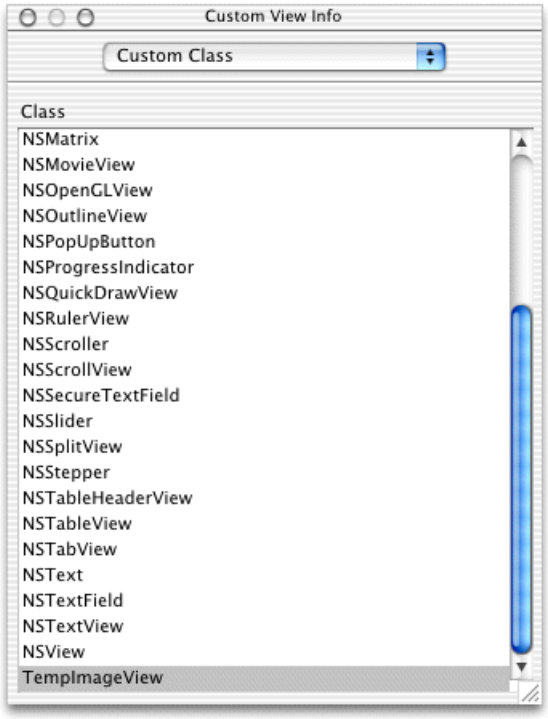
There is no need to specify any outlets or actions for this class.

Assign the Class to the CustomView

Unlike custom controller classes, where you use the Classes > Instantiate command to make an instance, you make an instance of a custom view in Interface Builder by assigning the class to the CustomView object.

1. Select the CustomView object in the window.
2. Select the Custom Class pane of the info window.
3. Select the TempImageView class in the list provided by that window.

Figure 3-5 Assigning the custom view to a class



Notice how the title of the custom view object changes to “TempImageView.”

Connecting the View Object

The TempImageView itself has no outlets or actions, but the controller object TempController needs to communicate with it to tell it when the temperature value changes. One additional outlet in TempController is needed for this purpose.

1. “Specify a Controller Outlet” (page 39)

2. “Connect the Instances” (page 39)

Specify a Controller Outlet

You can always add an action or outlet to an existing custom class. Just make sure the header and implementation files of the class (if created) reflect the new outlet or action.

1. Select the TempController class in the Classes display of the nib file window.
2. Click the electrical outlet icon next to the class.
3. Choose Add Outlet from the Classes menu (or just press Return).
4. Type the name of the outlet: “tempImage”.

Before you move on the next step, be sure to collapse the listing of outlets and actions by clicking another class.

Connect the Instances

You’ve already created an instance of TempController; all you need to do now is connect it to the TempImageView instance through the tempImage outlet.

1. Click the nib file window and the application window to bring them both to the front of the screen.
2. Control-drag a connection from the TempController instance in the Instances display to the custom view object (TempImageView) in the application window.

Figure 3-6 Connecting the controller and the view

3. Select the tempImage outlet in the Connections pane and click Connect.
4. Save the nib file.

Implementing the Custom View

The Application Kit's CustomView object would be useless without any programmatic alterations to it, so you must add some Java code to make the view a useful user interface element.

1. "Generate the .java File" (page 41)

2. “Implement the Constructor” (page 42)
3. “Implement the Image-Setting Method” (page 44)
4. “Call the Image-Setting Method” (page 45)

Generate the .java File

The classes under NSObject in the Classes display of the nib file window represent, in most cases, both Java and Objective-C versions of the same class. When you create source code files from your nib-file definitions, you must specify which language you want.

1. Select TempImageView in the Classes display of the nib file window.
2. Select the Attributes pane of the info window.
3. Click the Java radio button.

Figure 3-7 Make sure Java is selected before creating classes

4. Choose Classes > Create Files.
5. Make sure the project-related checkboxes are selected. See “Generate the Source Code Files” (page 29) if you need help with this step.

Implement the Constructor

The constructor for TempImageView loads image files from the application’s resources, converts them to NSImage objects, and assigns these to instance variables. (You’ll add these images to the project later in this tutorial.) It also sets certain inherited attributes of the image view. The following procedure approaches the implementation of this constructor in three steps.

1. Click `TempImageView.java` in Project Builder.
2. Add the instance variables for the three NSImages as shown here:

```
/* TempImageView */
import com.apple.cocoa.application.*;
import com.apple.cocoa.foundation.*;

public class TempImageView extends NSImageView {
    protected NSImage coldImage; // add this
    protected NSImage moderateImage; // add this
    protected NSImage hotImage; // add this
```

3. Add this constructor to load the images, create NSImages, and assign them to instance variables.

```
public TempImageView (NSRect frame) {
    // load images
    super(frame);

    String h = NSBundle.mainBundle().pathForResource (“Cold”, “tiff”);
    if (h != null) {
        coldImage = new NSImage(h, true);
    } else {
        System.err.println (“Image Cold.tiff not found.”);
    }

    h = NSBundle.mainBundle().pathForResource (“Moderate”, “tiff”);
    if (h != null) {
        moderateImage = new NSImage(h, true);
    } else {
    System.err.println (“Image Moderate.tiff not found.”);
    }

    h = NSBundle.mainBundle().pathForResource (“Hot”, “tiff”);
    if (h != null) {
        hotImage = new NSImage(h, true);
    } else {
        System.err.println (“Image Hot.tiff not found.”);
    }

    // Add code from step 4 here.
}
```

There are a few things to observe about the above excerpt of code:

- TempImageView's constructor is based on NSImageView's `NSImageView(NSRect)` method, so the first thing done is a call to super's constructor.
- To import the images, NSBundle's `pathForResource(String, String)` method is called to locate the image and return a path to it within the application bundle. This path is used in the `NSImage(String, boolean)` constructor.
- The error handling in this constructor is rudimentary. In a real application, you would probably want to implement something more useful. For instance, you could use the Java try/catch exception handler since Cocoa fully supports Java's JDK.

4. Add this code in place of the last comment in the code above to set attributes of the image view:

```
setEditable (false);
setImage (moderateImage);
setImageAlignment (NSImageCell.ImageAlignCenter);
setImageFrameStyle (NSImageCell.ImageFrameNone);
setImageScaling (NSImageCell.ScaleProportionally);
```

You might be wondering how you can learn more about the methods of a Cocoa class, especially their arguments and return types. The first place is look is in the developer center. Mac OS X also provides a tool to help you: JavaBrowser. Located in `/Developer/Applications`, this tool lists all Cocoa Java classes, their constructors, variables, and methods.

Implement the Image-Setting Method

TempImageView has one public method that TempController calls whenever the user enters a new temperature value: the `tempDidChange` method. Implement this method as shown here:

```
public void tempDidChange (int degree) {
    if (degree < 45) {
        setImage (coldImage);
    }
    else if (degree > 75) {
```

```
        setImage (hotImage);
    }
    else {
        setImage (moderateImage);
    }
}
```

These ranges are completely arbitrary, but could be influenced by a California climate. If you have lower or higher thresholds for hot and cold temperatures, you can specify your own ranges.

Call the Image-Setting Method

In the convert method of TempController, call TempImageView's `tempDidChange` method after converting the entered value. Copy the following example:

```
public void convert (NSTextField sender) {
    if (sender == celsius) {
        int f = (int)((9.0/5.0 * celsius.intValue()) + 32);
        fahrenheit.setIntValue(f);
    }
    else if (sender == fahrenheit) {
        int c = (int)((fahrenheit.intValue()-32) * 5.0/9.0);
        celsius.setIntValue(c);
    }
    tempImage.tempDidChange (fahrenheit.intValue()); // add this
}
```

You must also add the tempImage outlet you defined earlier in Interface Builder as an instance variable of type TempImageView in `TempController.java`. So, your instance variable declarations in `TempController.java` should now look like:

```
NSTextField celsius;
NSTextField fahrenheit;
TempImageView tempImage;
```

Completing the Application

You're almost ready to run the application. There are just a few more steps:

1. "Add Images to the Project" (page 46)
2. "Build the Project" (page 46)
3. "Test Drive the Application" (page 47)

Add Images to the Project

The TempImageView object must, of course, have images to show. Ready-made "climate" images come provided for this tutorial. You must add these image files to the application.

1. In Project Builder, choose Project > Add Files.
2. Navigate to the following directory:

   `/Developer/Documentation/Cocoa/JavaTutorial/ApplicationImages`

3. Add the following images: `Hot.tiff`, `Cold.tiff`, `Moderate.tiff`.

   Shift-click each file to select all images, then click Open to add them to the project.

4. Click Add to confirm adding the files to the project.
5. Choose Project Save from the Project menu.

Build the Project

Now you're ready to build the project. Click the Build button in the project window, or press Command-B to build the project. If there are errors in the code, they will appear in the Build pane. Click a line describing an error in the upper display to go to the line in the code containing the error; fix the error and rebuild.

Test Drive the Application

Launch the Temperature Converter application and see what it can do; this includes not only what you specifically programmed it to do, but the behavior the application gets "for free."

- Enter a low temperature value in the Fahrenheit field and press Return.

  The Celsius field displays the converted temperature and the image changes to the "cold" picture.

- Enter a high temperature value in the Celsius field and press Return.

  The Fahrenheit field displays the converted temperature and the image changes to the "hot" picture.

Now check out the "free" behavior.

- Click the window of another application or anywhere in the workspace.

  The Temperature Converter window loses key status (and as a result its title bar becomes opaque) and it might become obscured behind other windows on your screen. Temperature Converter is no longer the active application.

- Click the Temperature Converter window.

  It is brought to the front of the window system and is made key.

- Choose Hide from the Temperature Converter menu (the menu just to the right of the Apple menu).

  The Temperature Converter window disappears from the screen.

- In the dock, choose Temperature Converter from the applications currently running on your system.

  The Temperature Converter window reappears.

- Select a number in one of the text fields, choose Copy from the Edit menu, select the number in the other text field, and choose Paste from the Edit menu.

  The number is copied from one field to the other.

- Select a number again and choose a suitable command from the Services submenu of the Temperature Controller menu, such as Make Sticky.

The Services menu lists those applications that can accept selected data from your application and process it in specific ways. When you choose a Services command, the application associated with the command starts up (if it is not already running) and processes the selected number. (In the case of Make Sticky, the number appears in a Stickies window.)

- Chose Quit from the Temperature Controller menu.

Creating a Subclass of NSView

If you have completed the tutorial so far, you're done. You need not go any further—unless you would like to try a more interesting and challenging variation of the TempImageView class you created earlier. In this "extra credit" part of the tutorial, you will create a class for objects that know how to draw themselves and know how to respond to user events. These classes are custom subclasses of NSView. There are two major steps:

1. "Define a Custom Subclass of NSView" (page 48)
2. "Implement the Code for a Custom NSView" (page 49)

Custom subclasses of NSView are usually constructed differently than subclasses of other Application Kit classes because the custom NSView subclass is responsible for drawing itself and, optionally, for responding to user actions. Of course, you can do custom drawing in a subclass that doesn't inherit directly from NSView, but usually instances of these classes draw themselves adequately. This section describes in general terms what you must do to create a custom NSView subclass.

Define a Custom Subclass of NSView

The differences are slight between the Interface Builder procedures for defining a direct subclass of NSView and for defining a subclass of any Application Kit class that inherits, directly or indirectly, from NSView. The following is a summary of the required procedure in Interface Builder:

1. Drag a CustomView object from the Views palette and drop it in the window.
2. Resize the CustomView object to the dimensions you would like it to have.

3. Select NSView in the Classes display of the nib file window, choose Class > Subclass, and name your subclass "TemperatureView".
4. Add an outlet called "tempImage" to the TempController class so it can communicate with the subclass of NSView you just created. (If you completed the previous section on subclassing NSImageView, just reuse the tempImage outlet you created there).
5. Assign the class you defined to the CustomView object.

   Do this by selecting the object and selecting the class in the Classes pane of the info window.
6. Make a connection from the TempController instance variable to the CustomView object. (If you completed the previous section on subclassing NSImageView, the connection should still exist.)
7. Generate the "skeletal" .java file; before you choose the Classes > Create Files command, be sure to first select the TemperatureView class and then Java in the Attributes pane of the info window.

If you are unsure how to complete any of these steps, refer to the "Defining the Custom View Subclass" (page 34) and "Connecting the View Object" (page 38) sections of this tutorial.

Implement the Code for a Custom NSView

You can implement your custom NSView to do one or two general things: to draw itself and to respond to user actions. The basic procedures for these and related tasks are given below.

Important
The information provided in this section barely scratches the surface of the concepts related to NSView, including drawing, the imaging model, event handling, the view hierarchy, and so on. This section intends only to give you an idea of what is involved in creating a custom view. For a much more complete picture, see the description of the NSView class in the Cocoa reference.

Drawing

All objects that inherit from NSView must override the `drawRect` method to render themselves on the screen. The invocation of NSView's `display` method, or one of the `display` variants, leads to the invocation of `drawRect`. Before `drawRect` is invoked, NSView "locks focus," setting the Window Server up with information about the view, including the window device it draws in, the coordinate system and clipping path it uses, and other graphics state information.

In the `drawRect` method, you must write the code that transmits drawing instructions to the Window Server. The `drawRect` method has one argument: the NSRect object defining the area in which the drawing is to occur (usually the bounds of the NSView itself or a subrectangle of it). For drawing in Java, you can use the following classes:

- NSBezierPath offers methods for constructing straight or curved lines, rectangles, ovals, arcs, and polygons with bezier paths.
- NSAffineTransform has methods for translating, rotating, and resizing graphical objects, such as those created with NSBezierPath.
- The static methods of the NSGraphics class draw rectangles, including buttons of various styles. They also perform bitmap operations and provide various information about the Window Server and graphics context.
- Foundation's geometry classes—NSRect, NSSize, and NSPoint (and their mutable variants)—help you to compute the location and size of graphical objects.
- NSColor and NSFont, among other classes, have methods that directly set a parameter of the current graphics context.

Invalidating the View

With each cycle of the event loop, the Window Server ensures that each NSView in a window that requires redrawing is given an opportunity to redisplay itself. Besides implementing `drawRect` to draw your custom NSView, your application must indicate that an NSView requires redrawing when data affecting the view changes.

This indication is called **invalidation**. Invalidation marks an entire view or a portion of a view as "invalid," and thus requiring a redisplay. NSView defines two methods for marking a view's image as invalid: `setNeedsDisplay`, which invalidates the view's entire bounds rectangle, and `setNeedsDisplayInRect`, which invalidates a portion of the view.

You can also force an immediate redisplay of a view with the `display` and `displayRect` methods, which are the counterparts to the methods mentioned above. However, you should use these and related `display`... methods sparingly, and only when necessary. Constant forced displays can markedly affect application performance.

You should never invoke `drawRect` directly.

Event Handling

If an NSView expresses a willingness to respond to user events, it is made the potential recipient of any event detected by the window system. The view then just must implement the appropriate NSResponder method (or methods) that correspond to the event the view is interested in. (NSView inherits from NSResponder.)

What this means in practical terms is that an NSView must at a bare minimum do two things:

- Override NSResponder's `acceptsFirstResponder` method to return `true`.
- Implement an NSResponder method such as `mouseDown`, `mouseDragged`, or `keyUp`. The argument of each of these methods is an NSEvent, which provides information related to the event.

See the NSResponder and NSEvent class descriptions in the Cocoa reference for further information.

An Example

The TemperatureView class is similar to the TempImageView class implemented in the second part of the tutorial. Instead of displaying a different image when the temperature changes to a certain range, it draws a circle of a different color. To illustrate basic event handling, the TemperatureView class changes the thickness of the view's border each time the user clicks the view.

Follow these steps to create the example:

1. Change the contents of `TemperatureView.java` to the code given below.
2. Change the type of the TempImageView instance variable in TempController.java to type TemperatureView.
3. After you run the project with the new code given below, you may want to change the size and/or position of the custom view object in Interface Builder.

```
/* TemperatureView */

import com.apple.cocoa.application.*;
import com.apple.cocoa.foundation.*;

public class TemperatureView extends NSView {
    protected NSBezierPath sun;
    protected int temperature;
    protected int thickness;

    static public final int SpringSun=0;
    static public final int SummerSun=1;
    static public final int WinterSun=2;

    public TemperatureView (NSRect frame) {
        super (frame);

        NSRect rect;
        NSColor color;

        float shortest = frame.width() >=
            frame.height()?frame.height():frame.width();

        shortest *= 0.75;
        rect = new NSRect (((frame.width() - shortest) /2),
            (frame.height() - shortest) /2), shortest, shortest));
        sun = NSBezierPath.bezierPathWithOvalInRect(rect);
        thickness = 1;
    }

    public void drawRect (NSRect frame) {
        NSColor color;
```

```
        if (temperature == WinterSun) {
            color = NSColor.lightGrayColor();
        } else if (temperature == SummerSun) {
            color = NSColor.orangeColor();
        } else {
            color = NSColor.yellowColor();
        }
        color.set();
        sun.fill();
        NSGraphics.frameRectWithWidth(frame, (float)thickness);
    }

    public void tempDidChange (int degree) {
        if (degree < 45) {
            temperature = WinterSun;
        } else if (degree > 75) {
            temperature = SummerSun;
        } else temperature = SpringSun;
        setNeedsDisplay(true);
    }

    public void mouseDown (NSEvent e) {
        if (thickness == 3) {
            thickness = 1;
        } else {
            thickness++;
        }
        setNeedsDisplay (true);
    }


    public boolean acceptsFirstResponder() {
        return true;
    }

}
```

Debugging Java Applications

This part of the tutorial show you the basic steps for debugging Cocoa Java applications using the facilities provided by Project Builder. It tells you the steps you must take to prepare for debugging and illustrates several debugging commands.

Project Builder uses the Java Debugger for debugging Cocoa Java executables. The Java Debugger is a tool that uses a customized subset of *jdb* commands. Because it is implemented as a set of threads in the Java VM, it is always active when the VM is running. Thus you can interact with the Java Debugger even when the target is running (unlike gdb, which requires that the target be stopped before it can process commands).

Major Tasks:

1. "Preparing to Debug an Application" (page 55)
2. "Using the Java Debugger" (page 59)

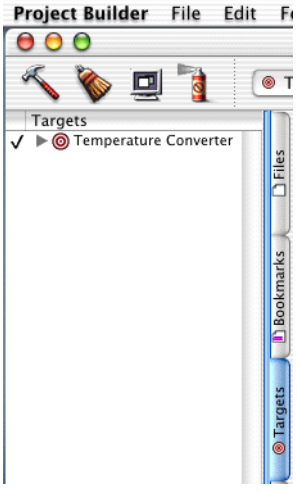
Preparing to Debug an Application

Before you can debug a Cocoa Java application, you must build the application with the proper debugging symbols and then run the Java Debugger, after which you can set breakpoints and begin debugging.

Prepare Project for Debugging

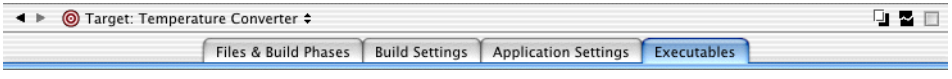
1. Display the Executables pane.
   a. Click the Targets tab on Project Builder's main window, and select Temperature Converter:

Figure 4-1 Selecting the targets tab



   b. Click the Executables tab in the Targets pane.

Figure 4-2 Selecting the executables tab

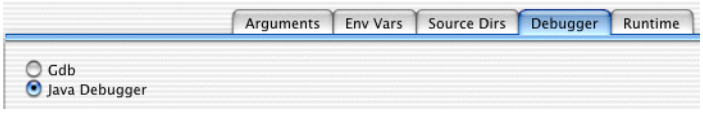


2. Verify that `Temperature Converter.app` is the selected executable, as shown:

Figure 4-3 Selecting the executable

3. Select Java Debugger in the Debugger pane:

Figure 4-4 Selecting the Java debugger

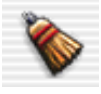


Build for Debugging.

1. Remove object files and other files generated by previous builds by clicking the "make clean" button. This step is necessary only if you had previously built an application executable without debugging symbols.

   The "make clean" button looks like a broom:

Figure 4-5 Make clean button



2. Build the project.

Access the Java Debugger

When you have built the project and have an executable containing symbols understood by *jdb*, run the Java Debugger from Project Builder. Project Builder knows which debugger to run, based on the type of executable it is debugging.

To launch the debugger, click the Debug button on Project Builder's main window.

Figure 4-6 Debug button

The Java Debugger starts up and launches Temperature Converter in the debugger. Because the Java VM is an interpreter, you do not need to suspend the Java Debugger to perform tasks such as setting breakpoints. You can, however, pause and resume the Java Debugger if you wish. The pause button looks like:

Figure 4-7 Pause button

The resume button looks like:

Figure 4-8 Resume button

Using the Java Debugger

Project Builder provides a sophisticated user interface for controlling Java debugging sessions. Unlike using gdb in Project Builder, you cannot interact with the Java Debugger through a command line interface. In this section, you'll learn how to use Project Builder's user interface to interact with the Java Debugger.

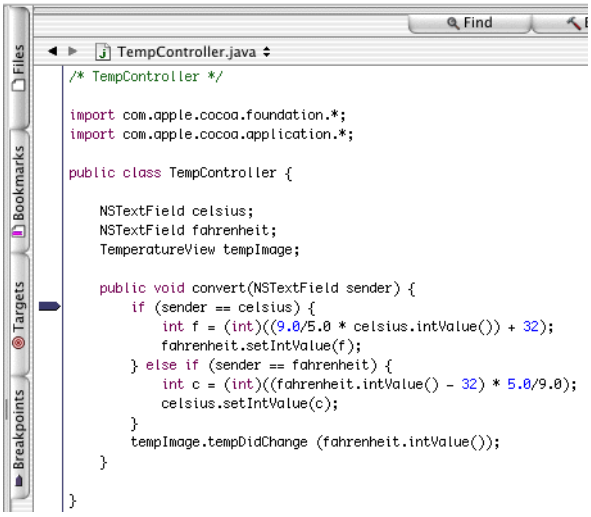
Set and Manipulate Breakpoints

When you were editing Temperature Converter's Java files in the code editor, you may have noticed the gutter on the left of the code editor. You set breakpoints by clicking in this gutter to the left of your code.

1. Locate the `convert` method in `TempController.java`.
2. Click in the gray band on the first line after the initial brace.

   A small pointer appears in the gray band.

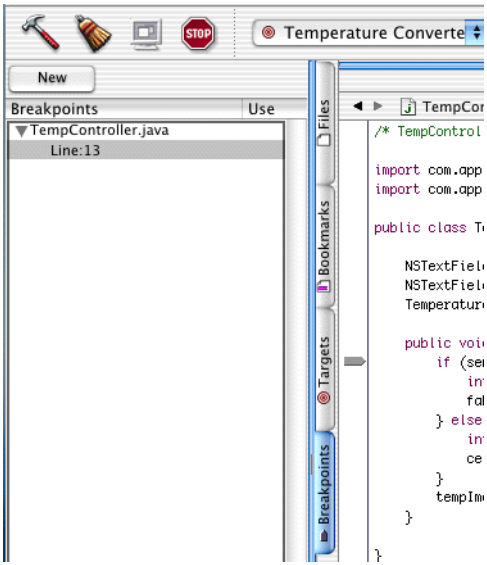
Figure 4-9 Setting a breakpoint



Once you set a breakpoint you can move it within a file by dragging a pointer up and down the breakpoint gutter. You can remove it by dragging the pointer off the gutter into the code editor. And you can temporarily disable a breakpoint, by clicking the pointer, after which it turns gray (re-enable it by clicking the pointer again). You can also use the Breakpoint pane of Project Builder's main window to enable and disable breakpoints:

1. Click the Breakpoints tab of Project Builder's main window to display the current breakpoints.
2. Click the breakpoint line under the Use column. (The breakpoint is disabled when the checkmark does not appear.)

Figure 4-10 Clicking the Breakpoints tab

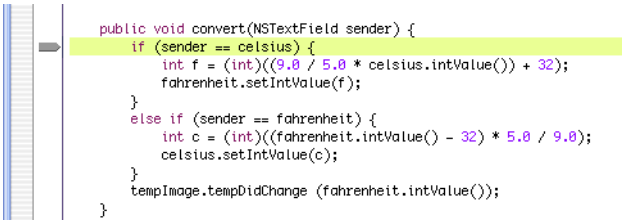


Step Into and Over Code

As in any kind of debugging, before you can perform any useful debugging task, such as stepping through code, you must run the program until it hits the next breakpoint.

1. Switch to the running Temperature Converter application, and enter a number in the Fahrenheit field and press Return. Execution stops at the breakpoint you just set.

Figure 4-11 Stopping at a breakpoint



2. Step over the first few lines of code by clicking the appropriate control on the Debug pane. The Step Over button looks like:

Figure 4-12 Step over button

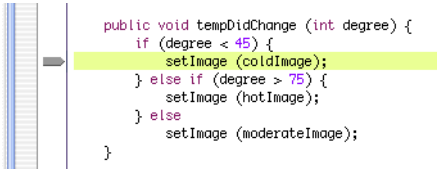
Notice how as you step over lines the program counter changes to the next line to be executed.

3. When you arrive at the last statement of the method (which calls `tempDidChange`), click the Step Into button, which looks like:

Figure 4-13 Step into button

The code editor displays `TempImageView.java` (or `TemperatureView.java` if you are using the view object implemented by that code) and the program counter points to the first line of the `tempDidChange` method.

Figure 4-14 Stepping into a method

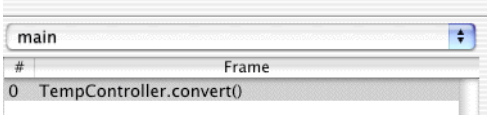


Get a Backtrace and Examine a Frame

At any point in debugging you can view the current stack frame.

1. Look for the Frame pane of the Debugger pane:

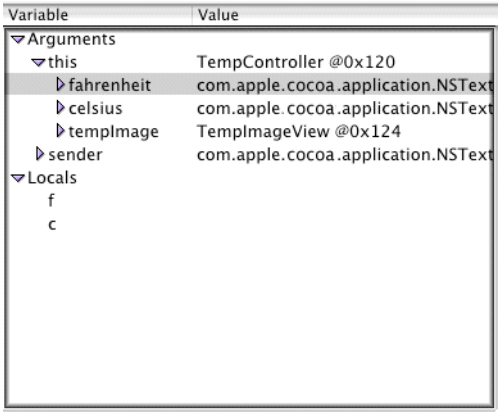
Figure 4-15 Frame pane



The pop-up menu lists the methods in the stack frame in the order of invocation.

2. Click in a method in the list to list its variables in the Variable/Value pane to the right of the frame pane.

Figure 4-16 Variable value pane



3. Click the disclosure triangle next to the argument `this` to display the instance variables.
4. Clicking next the disclosure triangle next to a variable will list its value in the Value column.

Debug Multiple Threads

With the Java Debugger, you can debug multiple threads in an application, switching among them to set breakpoints, examine stack frames and data, and perform other debugging tasks. The Java Debugger displays threads in groups; each thread has its own unique number within a group. To switch to another thread, you choose a thread in the threads pop-up menu above the Frame pane.

Figure 4-17 Selecting a thread