# 6

# A Cocoa Application: Views

Now that the model is taken care of, we turn to the other end of the application: the views. Our design for the application relies on standard elements in the Mac OS X Aqua interface.

Interface Builder (IB) is the indispensible tool for laying out human interfaces for Mac OS X applications. It edits *Nib files*, which are archives of human-interface objects that are reconstituted by your application when the files are loaded. All Cocoa application have a main Nib file, which the Xcode application template names `MainMenu.nib`. This file contains at least the main menu bar for the application and may contain other applicationwide windows and views.

An application can have more than one Nib, and a Nib can be loaded and its contents instantiated more than once. For instance, the Xcode template for a document-based Cocoa application includes a `MyDocument.nib` file. The file contains a trivial window for documents of the `MyDocument` class, and the template for the `MyDocument` class implementation specifies `MyDocument.nib` as the Nib to load when creating a new document.

Because our design calls for a window that displays the data points and regression statistics for each document, we want to edit the window in `MyDocument.nib` to match our design. In the Groups & Files list, click the triangle next to **Resources** to open that group. You should see **MyDocument.nib**, with the gold icon of a Nib file, in that group. Double-clicking this item launches Interface Builder and opens `MyDocument.nib` for editing.

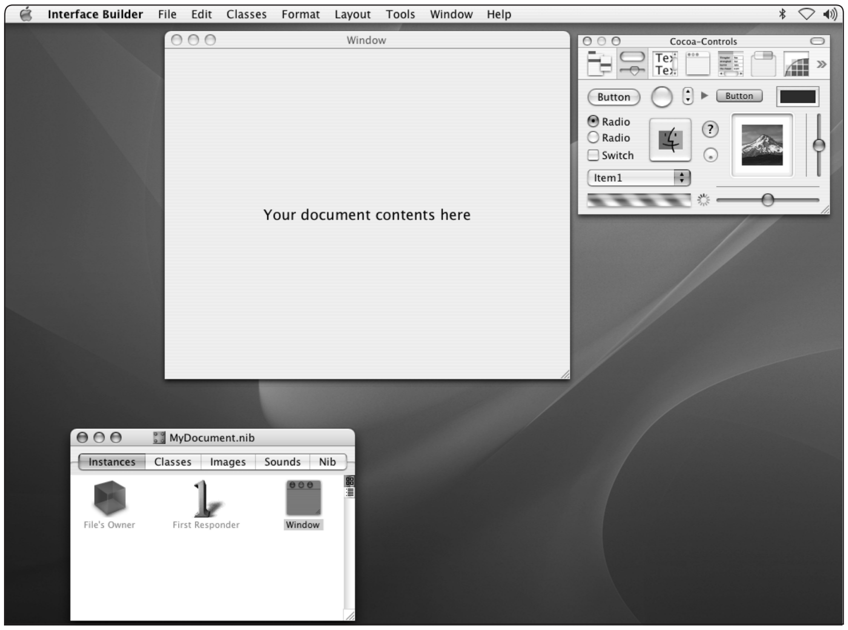


**Figure 6.1** Interface Builder on opening `MyDocument.nib`. The window that represents the Nib is at lower left; above it is the simple window, with some filler text, that comes in `MyDocument.nib` as provided in the template. To the right is the palette containing standard Aqua controls that can be dragged into windows.

## 6.1 Interface Builder

The newly opened Interface Builder will show you three windows ( Figure 6.1). (Use **Hide Others** in the **Interface Builder** application menu to reduce the clutter.) The largest, named Window, contains a text element saying "Your document contents here" in the middle. Close this window.

The window at the lower left, MyDocument.nib, shows the Nib file as a file opened by Interface Builder (Figure 6.2). It shows three icons. The first two—**File's Owner** and **First Responder**—are placeholders for certain objects outside the Nib. Any other icons are for objects in the Nib—in this case, a window named Window.

Double-click the **Window** icon. The window we're building for the `MyDocument` class opens again.

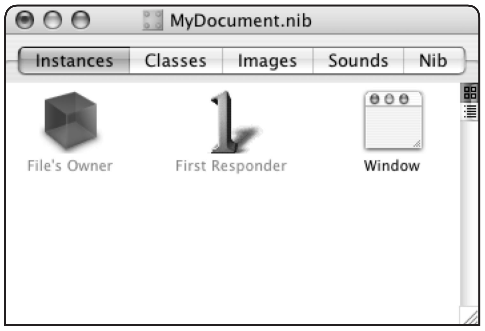


**Figure 6.2** The window for the `MyDocument.nib` file contains icons for **File's Owner** and **First Responder**, which are placeholders for objects outside the Nib, along with an icon for every top-level object in the Nib. In this case, `MyDocument.nib` contains only a window, named Window.

I'll have to be careful with my terminology here. At the bottom of your screen is the window for the `MyDocument.nib` file. You opened that file with Interface Builder, you're editing the file, and you'll save it when you're done. Above it is a window you're building for the use of the `MyDocument` class. In a sense, you're opening, editing, saving, and closing it, too, but the window is *inside* the `MyDocument.nib` file. I'll try to be as explicit as posssible in distinguishing the Nib window from the prototype document window, but you may have to watch out for the distinction.

The third window, to the right, is a utility window embodying a series of palettes containing objects you can add to a Nib file. We'll be using this window a lot.

## 6.2 Layout

First, we will use Interface Builder as a pure layout tool for our human interface. We'll start by getting rid of that "Your document contents here" placard. Click it, and press the **Delete** key. It's gone.

Next, let's add the buttons. If the palette window doesn't show "Cocoa – Controls" as its title, click the second toolbar icon—the one that shows a button and a slider. At the top left of the palette is a regular Aqua-style button labeled **Button**. Drag this

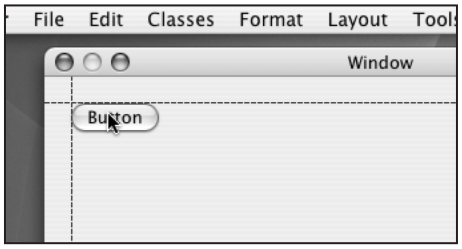


**Figure 6.3** Placing a button in Interface Builder. Drag the button from the Cocoa–Controls palette to the window being built. Lines will appear in the window when the button is placed properly according to the Aqua human-interface guidelines.

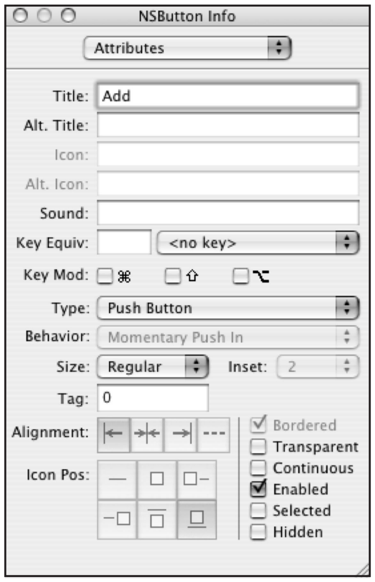
button from the palette into the window we're building for `MyDocument` (Figure 6.3). As you drag the button into the upper-left corner of the target window, blue lines appear at the window's margins. The Aqua human-interface guidelines specify certain margins between controls and within windows, and Interface Builder puts up guides to help you place your elements properly.

**Button** is not an especially informative title for a control. Our design calls for this button to be named **Add**. The easiest way to change the label is to double-click the button, making the text editable, and replace the title. Instead, we'll take this opportunity to have our first look at Interface Builder's Inspector. Select **Show Info** from the **Tools** menu. A new utility window opens, offering a number of options for configuring the current selection—in this case, the button we just dragged in (Figure 6.4). The field at the top of the inspector is labeled Title: and shows the current value, **Button**. Edit this to say `Add`, and press **Tab** or **Enter** to complete the edit. The button is now named **Add**.

Leave the Inspector window open. It will change to keep up with the currently selected element in the Nib, and we'll be needing it later.

Repeat the button-dragging twice more, for the **Remove** and **Compute** buttons. Name the new buttons accordingly.

Next, we add the table of data to the window. Click the fifth icon at the top of the palette to reveal Interface Builder's repertoire of data-display views. The table view (`NSTableView`) is in the lower-left of the Cocoa–Data palette. Drag this view into the window we're building. Lines will appear that allow you to place the table view a short distance below the buttons and just off the left edge of the window. Small blue knobs appear at the edges and corners of the view to let you resize it. Make the view wide



NSButton Info

Attributes

Title: Add

Alt. Title:

Icon:

Alt. Icon:

Sound:

Key Equiv: <no key>

Key Mod: ☐ ⌘ ☐ ⇧ ☐ ⌥

Type: Push Button

Behavior: Momentary Push In

Size: Regular Inset: 2

Tag: 0

Alignment:

Icon Pos:

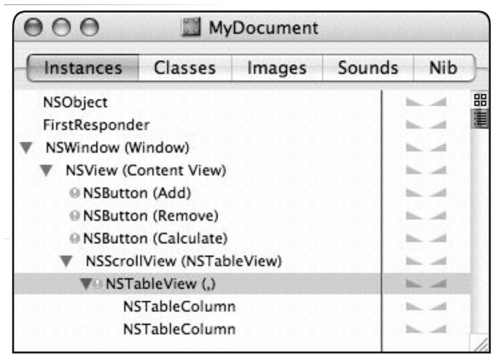
✓ Bordered
☐ Transparent
☐ Continuous
✓ Enabled
☐ Selected
☐ Hidden

**Figure 6.4** The Interface Builder Inspector, as a simple push button is selected. The default label for the button (**Button**) is replaced by **Add**.

enough to display two columns of numbers comfortably and deep enough to trigger a line at the bottom of the window.

What you've added to your document window is much more than simply an `NSTableView`. Look at the main Nib MyDocument window, which should be at the lower left of your screen if you haven't moved it. Make sure that the **Instances** tab is selected. At the top of the scroll bar at the right edge of window are two small buttons. The upper one, with four small boxes in it, is now highlighted; click the other one, with horizontal lines, to highlight it.

The Instances view changes to a hierarchical display of the Nib contents. **MyDocument** and **First Responder** come first, followed by an **NSWindow** with a disclosure triangle next to it. Clicking the disclosure triangle shows that a Cocoa window contains one **NSView**, the content view, which, because it contains other views, also has a disclosure triangle. If we open all the disclosure triangles we see, we end up with something like Figure 6.5. What we last dragged into the window was in fact an `NSScrollView`, containing an `NSTableView`, which in turn contained two `NSTableColumns`.



```
NSObject
FirstResponder
▼ NSWindow (Window)
  ▼ NSView (Content View)
      NSButton (Add)
      NSButton (Remove)
      NSButton (Calculate)
    ▼ NSScrollView (NSTableView)
      ▼ NSTableView (,)
          NSTableColumn
          NSTableColumn
```

**Figure 6.5** The hierarchical view of the MyDocument Nib in progress. You reach this view by selecting the **Instances** tab and then the list-view button just above the scroll bar at the right edge of the window. It can sometimes be easier to select views in this list than in the window display. The yellow caution badges on the screen indicate that "outlets"—links to other objects—in the flagged views have not yet been filled.

Select the header of the first column of the table. Using your mouse, click once to select the scroll view, double-click to get inside to the table view, double-click again to get down to the header view, and click once more to bring up the text field editor. Alternatively, simply keep clicking rapidly on the header until it turns white and a blinking insertion point appears. Type **`x`**. In the second header, type **`y`**. Putting the mouse between the headers will allow you to drag the boundary between them, so you can resize the columns to equal size.

The last element we'll put in the window is an `NSForm`, a simple array of labeled text fields that we'll use for the results of the regression. Find the form element in the third panel of the IB palette (Cocoa–Text), at bottom center. Drag it into the right half of the window you're building, under the **Compute** button. As supplied, the form has two big defects: It's too narrow, and it shows only two items. The width problem is easy to solve: Drag the handles on the sides of the form until they hit the spacing guidelines.

Dragging the handles on the top and bottom, however, gets you a taller form with two entries (**Undo** is your friend here). `NSForm` turns out to be a subclass of `NSMatrix`, a Cocoa class that manages an array of controls. You can add rows or columns to an `NSMatrix` in Interface Builder by dragging a resize handle *while holding the **Option** key down*. An **Option**-drag downward on the bottom handle of the form gets us our third row.

Click repeatedly on the labels in the form until they become editable, and change them to Slope:, Intercept:, and Correlation:. You'll probably have to resize the form when you're done.

## 6.3 Sizing

At this point, the layout of the window is almost done. Why "almost"? Pull down the **File** menu and select **Test Interface** (or press **command-R**). Your window now "goes live," using the components you put into it. There's nothing behind them, but you can click the buttons and work the other controls.

Now try resizing the window. The contents of the window ride the lower-left corner, sliding into and out of view as the window resizes (Figure 6.6). This is not what we want. Near the right end of your menu bar is an icon resembling an old-fashioned double-throw electrical switch. Click this icon to exit the interface-testing mode. None of the changes you made during testing are permanent.

Cocoa views can be set to resize or simply to stay put as their containers change size and shape. Click the **Add** button in the window we're constructing to select it. If

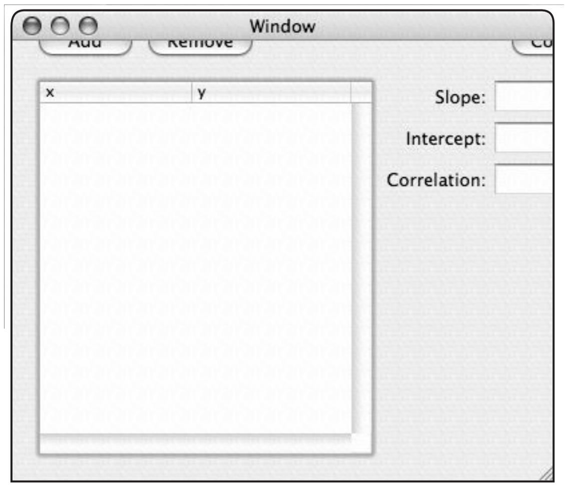
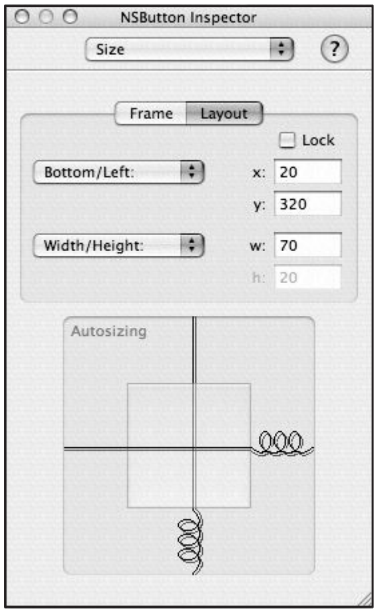


**Figure 6.6** As supplied, Cocoa controls do not automatically size and position themselves in a window as the window resizes. You have to specify sizing and positioning behavior yourself.



Bottom/Left: x: 20 y: 320
Width/Height: w: 70 h: 20

**Figure 6.7** The Size Inspector for a view that should stay put, relative to the top left of its enclosing view. The inside struts are all straight, meaning that it never resizes. The outside struts below and to the right are springy, meaning that they don't influence the view's placement.

the Inspector panel is not showing, select **Show Inspector** (or press **command-shift-I**) from the **Tools** menu. In the pop-up menu at the top of the Inspector window, select the third item, **Size**. (Note that you can bring the Inspector forward with the Size panel visible by pressing **command-3**.)

The Size panel (Figure 6.7) is dominated by a diagram showing the behavior of the selected view when its enclosing view is resized. The square inside the diagram represents the view itself. The various lines in the diagram switch between "struts" (straight lines) and "springs" (curling lines) when you click them. A view whose inner box contains rigid struts in both directions does not resize; if it has a spring in either direction or both, it can resize in that direction.

When the outer lines are struts, the view will try to maintain the same distance from the corresponding edge of the container. (If both lines are struts and the view isn't resizable, lower and left wins over upper and right, which is why the contents of our window "rode" the lower-left corner out of sight when you resized the window.)

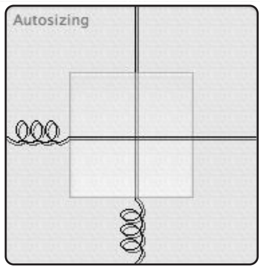


**Figure 6.8** This view maintains its position relative to the top *right* of its enclosing view. The outside struts to the top and right are rigid.

If an outer line is a spring and the view is not resizable in that direction, the view will ignore the movement of that edge of the container. If the view is resizable and an outer line is a spring, the view will resize to a degree proportional to its position in the window. Two side-by-side views that obey this rule will maintain their respective shares of the window when it is resized and will not run into each other.

We reexamine the views in the window with an eye to how they should behave when the window resizes. The buttons should never resize and should stay where they are relative to the nearest corner—top left for **Add** and **Remove**, top right for **Compute**. Select each in turn, making Size Inspector for the **Add** and **Remove** buttons look like the one in Figure 6.7 and for the **Compute** button, like Figure 6.8.

How do we want the form at the right of the window to behave? We certainly don't want it to shrink or stretch vertically with the window, but we wouldn't mind its growing if the window were to get wider. So the vertical internal strut should be rigid, and the horizontal strut should be springy. We want it to keep its position near the right edge of the window, so the right strut remains rigid. We also want it to stay an inch or so below the title bar, so the top strut is rigid. The bottom strut becomes a spring, allowing the form to float free of that edge. The left strut also becomes a spring, indicating that the form will expand to take a share of the window rather than maintain a rigid margin from the left edge. See Figure 6.9.

The data table should be freest of all, widening with the window and also growing vertically to show more points if the window grows. Both internal struts should be turned into springs. We anchor the view to the top left of the window by leaving those outer struts rigid; we also leave the bottom strut rigid. That way, when the window gets taller or shorter, the table will keep a constant distance of 20 pixels from the bottom

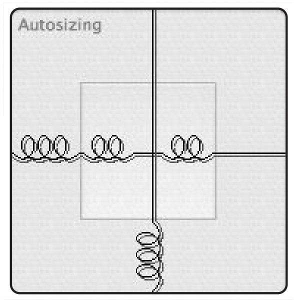


**Figure 6.9** The resizing specification for the form at the right side of the window. We don't want it to stretch vertically, so the inside vertical strut is straight. It would be nice if this view could take advantage of more room horizontally, so its inside horizontal strut is springy, allowing resizing in that direction. It is strictly bound to the top and the right side of the surrounding view. Being resizable horizontally and loosely bound to the left, this view will resize itself proportionately as the window resizes.

of the window. The right-horizontal strut is made springy, so the table view will grow horizontally only to maintain its share of the horizontal space. See Figure 6.10.

Now press **command-R** to try out the window. Now resizing does not shove views out the window. You may want to experiment with other sizing options and see their effects. Remember, you can exit the interface test mode by clicking the switch icon in the menu bar.

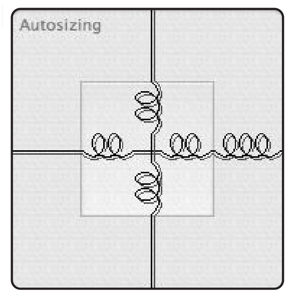
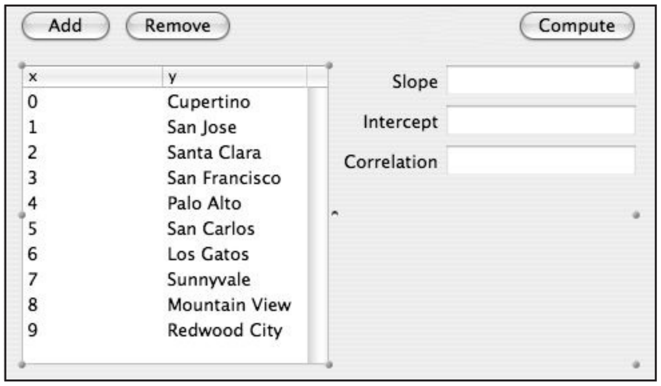


**Figure 6.10** The resizing specification for the scroll view enclosing the data table. It resizes in both directions along with the window. It is strictly bound to the top, left, and bottom edges of its enclosing view, so it will resize to maintain its present distance from those edges; it is loosely bound to the right, so it will take only a proportionate share of growth in that direction.



| x | y |
|---|---|
| 0 | Cupertino |
| 1 | San Jose |
| 2 | Santa Clara |
| 3 | San Francisco |
| 4 | Palo Alto |
| 5 | San Carlos |
| 6 | Los Gatos |
| 7 | Sunnyvale |
| 8 | Mountain View |
| 9 | Redwood City |

**Figure 6.11** Adding a split view. Selecting two side-by-side views, and using the menu command **Layout → Make subviews of → Split View** encloses the views in an `NSSplitView`, with the splitter between them.

## 6.4 A Split View

Our idea of how much space to allocate between the table of data points and the form containing the output statistics might not be the one that should control. The user may have other ideas. It would be better to put the two views inside a split view so the user can drag the border between them back and forth.

Interface Builder does not provide an `NSSplitView` or an `NSScrollView` in its palettes, except for the ones prebuilt around other views. To get a split view, select the views you want to be in the split view—in this case, the scroll view containing the data table and the `NSForm` for the results. Then select the menu item **Layout → Make subviews of → Split View** to wrap the views in a split view big enough to contain them and oriented so the split comes between them. See Figure 6.11.

Try out the new view by pressing **command-R**. The halves of the split view should resize as you drag the dimpled bar between them. The new split view comes with no automatic resizing, so when you return Interface Builder to its normal editing mode, you'll want to set the split view's internal struts to springs so it can resize with the window.

## 6.5 Summary

This chapter introduced Interface Builder, a tool no less important to Cocoa development than Xcode itself. We used IB as a straightforward tool for laying out windows and views. We saw how to set the many options for automatic sizing of embedded views and how to use Interface Builder's own simulation mode to verify that our layout and sizing choices work.

This does *not* end our work with Interface Builder. Because it is a constructor for networks of Cocoa objects, Interface Builder will have a big role to play as we move on to the controller layer of our design.