The Essential Guide to Flex 2 with ActionScript 3.0

Charles E. Brown

friendsof
DESIGNER TO DESIGNER™
an Apress® company

The Essential Guide to Flex 2 with ActionScript 3.0

ISBN-13 (pbk): 978-1-59059-733-0

ISBN-10 (pbk): 1-59059-733-8

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit www.apress.com.

The source code for this book is freely available to readers at www.friendsofed.com in the Downloads section.

Credits

Lead Editor
Chris Mills

Technical Reviewer
Sas Jacobs

Editorial Board
Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Matt Wade

Senior Project Manager
Beth Christmas

Copy Edit Manager
Nicole Flores

Copy Editor
Ami Knox

Assistant Production Director
Kari Brooks-Copony

Senior Production Editor
Laura Cheu

Compositor
Diana Van Winkle

Artist
April Milne

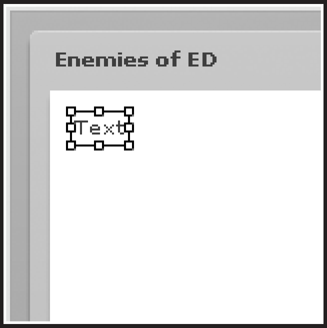
Proofreader
Liz Welch

Indexer
Toma Mulligan

Interior and Cover Designer
Kurt Krames

Manufacturing Director
Tom Debolski

7 STATES


Enemies of ED
Text

New State
Name: Contact
Based on: <Base state>
Set as start stat
OK

XML for Flash

Flex creates SWF files and, in the past, such files were always associated with Flash. But Flex applications do not get built like Flash applications. The timeline and many of the Flash animation tools you have come to know and depend on are missing.

Or are they?

As you will see in this chapter, states capture many of the features you thought you may have lost from Flash. However, states handle those features in a very different way. With states, you can dynamically change forms based upon incoming data or a user's response.

In this chapter, you are going to explore some traditional Flash concepts; but in a very new light. You will

- Explore the concept of a state.
- Review a brief history of web design.
- Understand click properties.
- Create rollover effects.
- Use components with states.

Before you begin, let me make one point clear: Flex is not an animation program.

One of the questions I am most asked in my seminars is, "Will Flex be replacing Flash?" The answer is an unqualified *no*!

Flash started its life nearly 10 years ago as a powerful animation program. When versions MX and MX 2004 came along, we saw a move from an animation program to a web design program. As a result, many felt that Flash had become a hodge-podge of features that didn't always work together gracefully, and that Flash was losing its original purpose of providing powerful animation capabilities.

To address these issues, Macromedia (now Adobe) developed Flex to take up the web design duties of Flash with the hope of refocusing Flash on animation.

As you will see in this chapter, states give Flex many of the dynamic form capabilities seen in Flash MX and MX 2004.

Understanding states

States allow you to create dynamic forms that can respond to data or user events. For instance, based on a user response, form components may appear or disappear.

Before you dig into states, it will help for you to have a general understanding of web design in XHTML and Flash first, so I'll go over this briefly here.

Let's step away from the concepts of Flash for a moment.

Traditional non-Flash web design usually consists of many individual XHTML pages, arranged in some sort of hierarchy, tied together by some sort of common navigation system. The navigation system, at its most basic level, is a series of hyperlinks that moves users from page to page.

While this system works most of the time, it is very inefficient. Each time the user clicks a hyperlink, a request has to be sent to a web server, and the page has to be located by that server, sent over the Internet to the caller, and loaded into the browser. In some cases, a "Page Not Found" error occurs.

Flash addressed many of these issues by internalizing a complete website into one SWF file; in some very complex situations, two or three SWF files might be used and incorporated various ways into the main SWF file. In earlier versions of Flash, clicking a link in the navigation system moved the user from page to page by moving to a different point on the timeline or possibly a different scene. In Flash MX and Flash MX 2004, you could dynamically change the interface right in the same frame. In later versions, rather than using the timeline or scenes, pages were created using multiple SWF files that could be encased within the library of the main SWF file or loaded dynamically.

All of this improved the efficiency of web navigation in that each request did not need to go back to the server and be returned over the Internet. Once the main SWF file was loaded into the Flash Player, everything needed for that website to be fully operational would be loaded with it.

Flex suddenly changed things again, most notably by eliminating the now familiar timeline. But if the timeline is eliminated, how does one jump from page to page within an application?

One way is to use the navigation containers you saw in the previous chapters. Another way is by employing states. Moving between states is similar to jumping to different points on a timeline. You'll get a better understanding of this as you work through the examples in this chapter. In the process of building these examples, you will have a chance to review some previously discussed concepts.

Before you start, download the files, if you haven't done so already, for Chapter 7. These can be found on the code download page for this book at `www.friendsofed.com`.

The first example will be a simple one: you will make another panel, or panels, appear by clicking a hyperlink.

1. Delete previous projects and create a new Flex project using the downloads for Chapter 7 and start a new main MXML file. I am going to call this one Chapter7_States. However, use any name you might want. Just make the location wherever you unzipped the downloaded files.

2. Switch to Design view and drag a `Panel` container to the upper-left corner of the stage (see Figure 7-1).

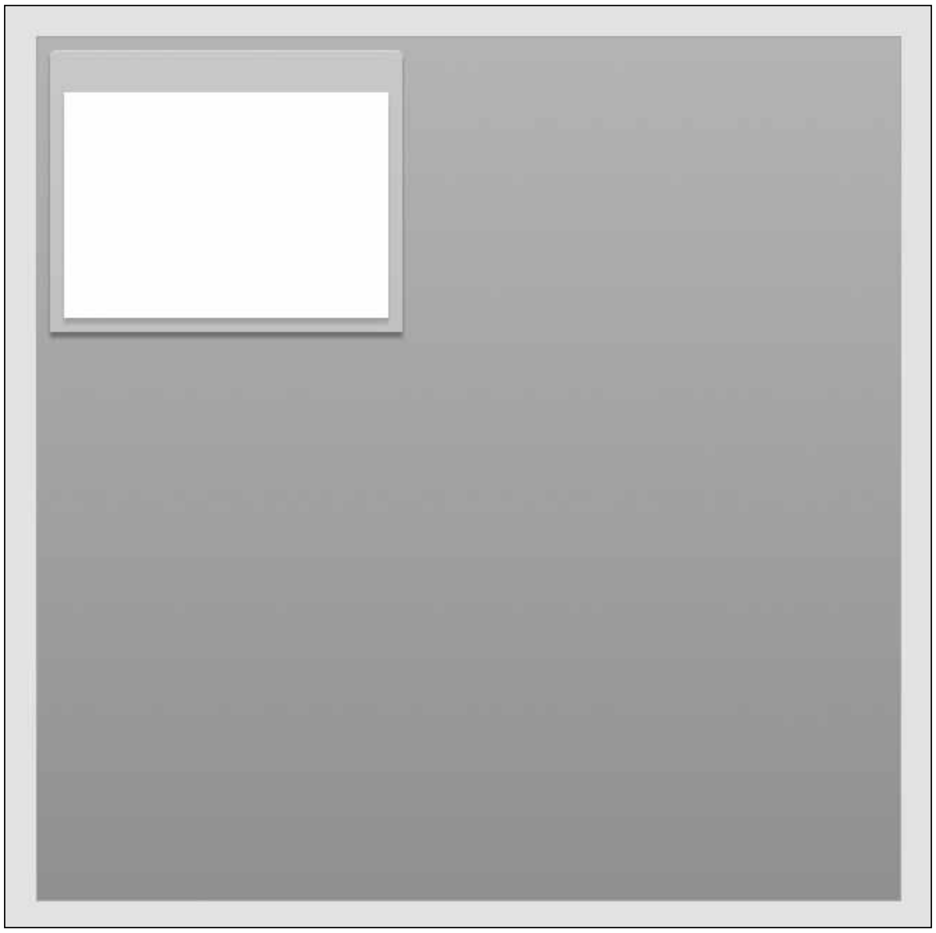


Figure 7-1. Placement of the Panel container

3. Double-click in the header of the `Panel` container (the gray area). This is another way to enter a header, in addition to using either code or the Flex Properties panel.
4. Type in a header. (I will have a little fun and use Enemies of ED.)
5. Press Enter. Your screen should look similar to what you see in Figure 7-2.

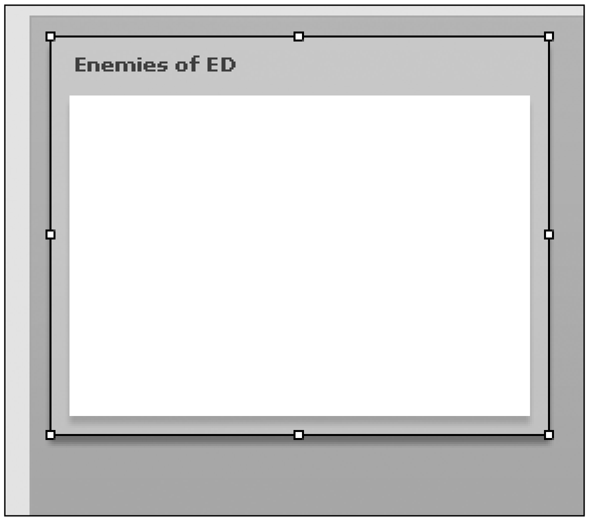


Figure 7-2. The Panel container with a header

6. Now you need to add some text to the body of the container. However, you can't type directly in the container, so drag a Text control into the body, as shown in Figure 7-3. The exact placement is not important.

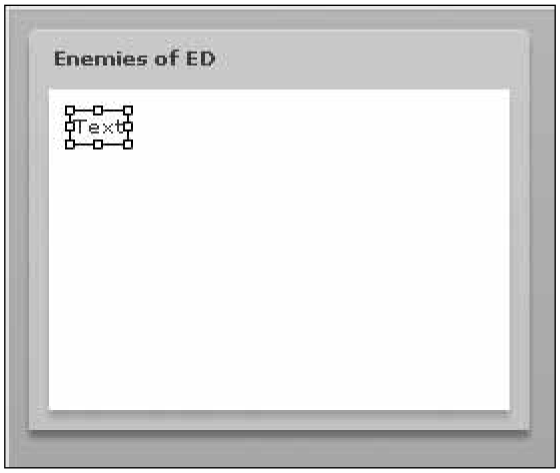


Figure 7-3. The Text control added to the Panel container

7. If necessary, highlight the word *Text* and type the following:

   We would like to hear from you. Please click on the link below to find out how to contact us.

> *If you find it easier, you can also add the text using the* Text *field in the Flex Properties panel.*

8. Press Enter, and you should end up with a Text control that extends beyond the Panel container, as shown in Figure 7-4.

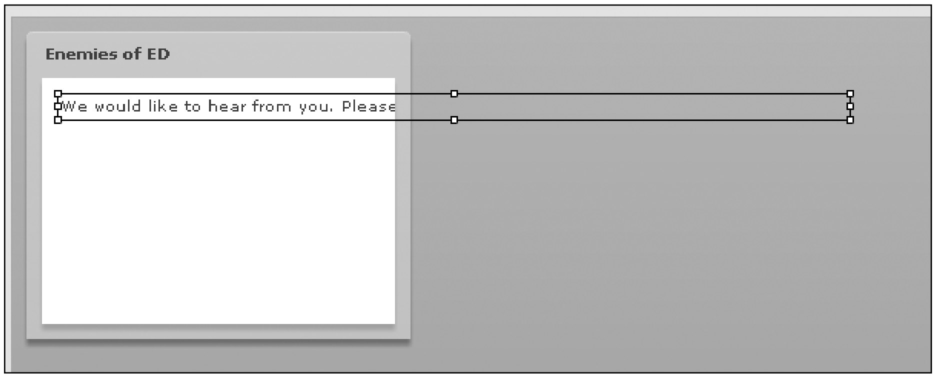


Figure 7-4. The initial state of the Text control

9. This is easily fixable—use the right-middle graphic handle to drag the right edge of the control back into the white area of the Panel container so that your screen resembles Figure 7-5.

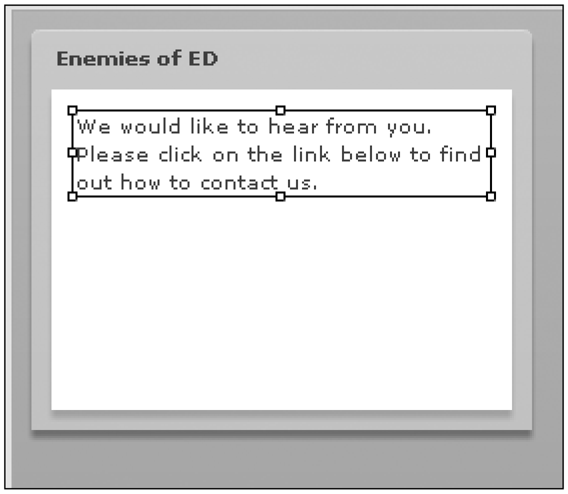
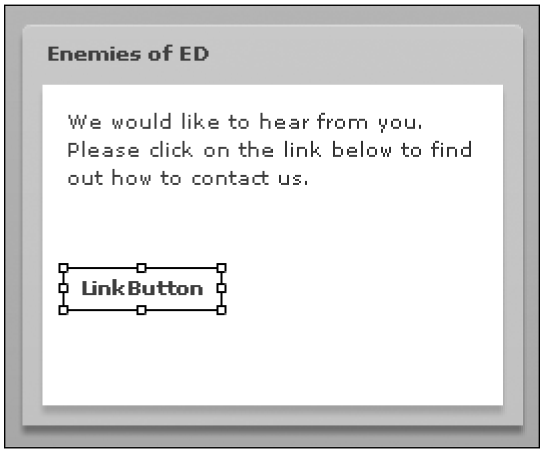

Enemies of ED
We would like to hear from you.
Please click on the link below to find
out how to contact us.

Figure 7-5. The adjusted Text control

You want to make a new panel appear when an event happens. In order to accomplish this, what you want to do is create an event to trigger the new state. This is a good chance to look at a control you haven't seen before: `LinkButton`.

The `LinkButton` control is, in my opinion, a rather unfortunate name, as the result is not a button at all. Instead, it is closer to a hyperlink in XHTML.

10. Drag a copy of the `LinkButton` control and place it under the text you just created (see Figure 7-6).



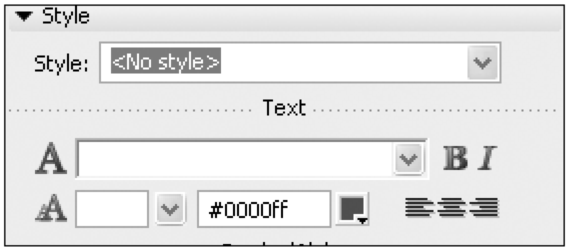
We would like to hear from you. Please click on the link below to find out how to contact us.

Figure 7-6. The LinkButton control added to the Panel container

11. Once again, you can either double-click the control or use the Text field in the Flex Properties panel to change the text. Type the text Click Here to Email Us.

12. Press Enter to lock in the changes.

13. Like all hyperlinks, you may want to change the color of the text to identify it as being a hyperlink. This can be easily done in the Flex Properties panel and the text color field, as shown in Figure 7-7.

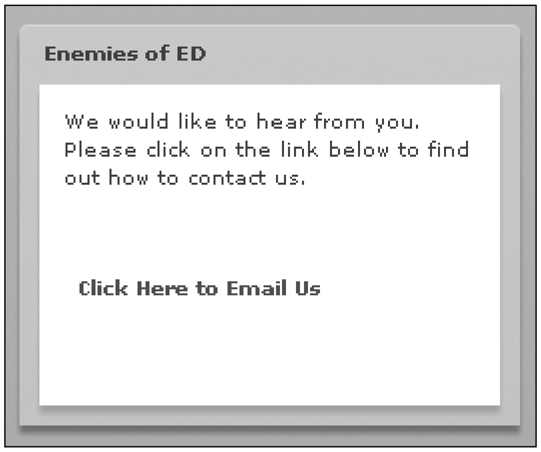


Style: <No style>

#0000ff

Figure 7-7. Setting the text color

14. Go ahead and test the application, and look at the `LinkButton` control (see Figure 7-8). The background color changes when you roll over the text, so you can easily see how this looks and feels more like a hyperlink than a button.



Enemies of ED

We would like to hear from you. Please click on the link below to find out how to contact us.

Click Here to Email Us

Figure 7-8. The finished Panel container

15. Go ahead and close the browser and return to Flex Builder.

Changing the state

Now the next trick is to use the `LinkButton` control to change the state of the application. Notice that in the upper-right corner, above the Flex Properties panel, there is a panel called States (see Figure 7-9).

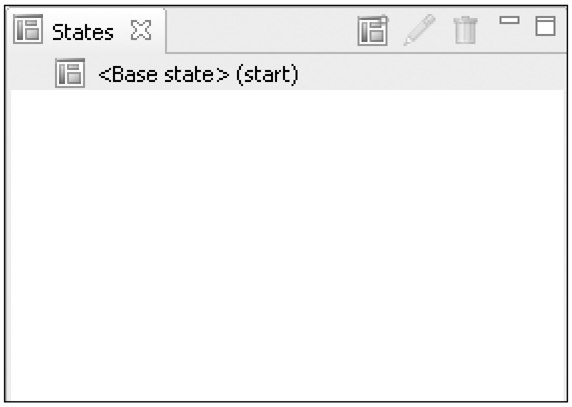


Figure 7-9. The States panel

If your States *panel is not visible, you can turn it on by selecting* Windows ➤ States.

All Flex applications start in a base, or start, state. This is the default state. In other words, what you see in Flex Builder is what you get.

But now you are going to add an additional state.

1. Either right-click <Base state> in the States panel or click the New State button in the upper-right corner of the panel.
2. Select New State.
3. Name your state. You can see in Figure 7-10 that I called my new state `Contact`.

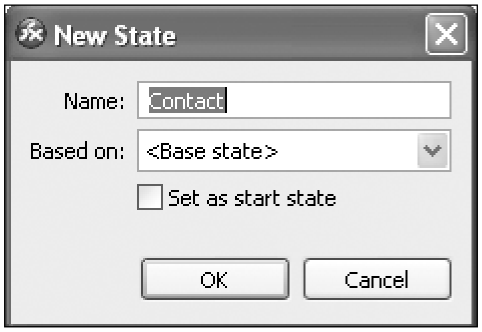

New State
Name: Contact
Based on: <Base state>
Set as start state

Figure 7-10. The New State dialog box

4. The Based on list is where you specify what state you want to build the new state over. Since you have no other states built, leave this set as <Base state>.
5. You can also make this new state the default or start state. For now, leave the Set as start state option unchecked.
6. Click OK. You should now see your new state appear in the States panel, as in Figure 7-11.

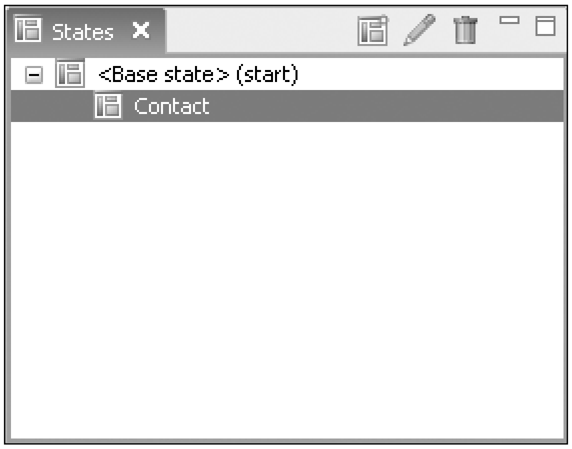
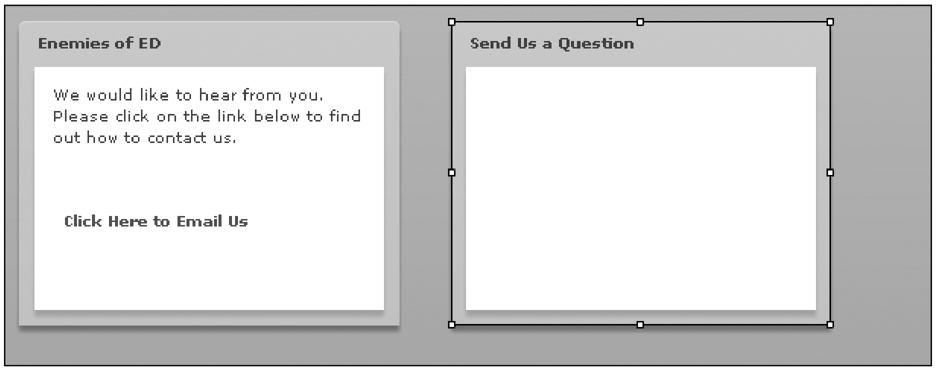

States
<Base state> (start)
Contact

Figure 7-11. The new state added to the States panel

Everything looks exactly the same on the new state. But now you are going to change that.

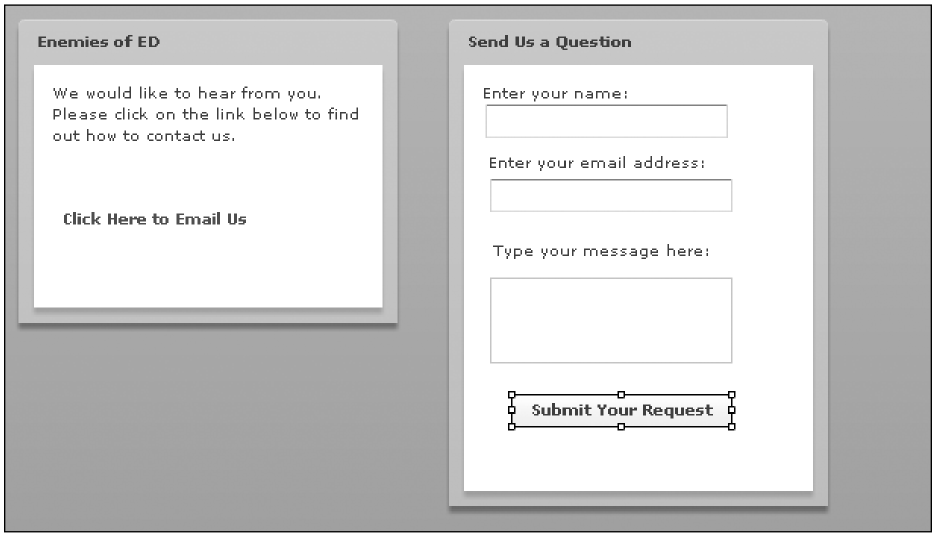
7. Make sure that `Contact` is selected in the States panel.
8. Drag another panel onto the stage, place it to the right of the existing panel, and as shown in Figure 7-12, give it the heading of Send Us a Question.



We would like to hear from you. Please click on the link below to find out how to contact us.

Figure 7-12. Adding a second panel in the Contact state

Let's build a simple e-mail form in this new `Panel` container as shown in Figure 7-13. Since you will not be actually e-mailing it now, the details of the form are not important.



We would like to hear from you. Please click on the link below to find out how to contact us.

Figure 7-13. The completed contact form

9. Using Figure 7-13 as your guide, build a sample form using a combination of `Label`, `TextInput`, `TextArea`, and `Button` controls.
10. Resize the `Panel` container to accommodate the form you just created.

Let's see a little magic now.

11. In the States panel, click the <Base state> tag. The Panel container you just created disappears. If you click back on the Contact tag, your new Panel container returns.

Welcome to states!

As I said earlier in this chapter, this is similar to moving to a new point on the Flash timeline or using ActionScript to rebuild the interface.

You now have to add some code so that the LinkButton and Button controls can change state.

12. Return to the base state if you are not there already. Just the initial Panel container should be visible.
13. Click the LinkButton control.
14. In the Flex Properties panel, click the Category View button (see Figure 7-14).

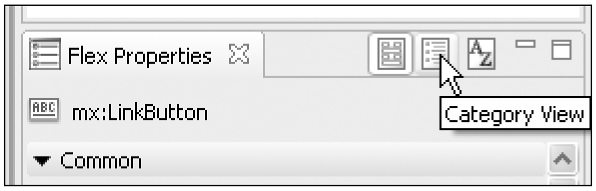


Figure 7-14. The Category View button

15. Under the Events category, select the Value column to the right of the click event.
16. In the Value box, type the following code:

```
currentState='Contact'
```

Notice that Contact is enclosed in single quotes.

17. Press Enter.

As discussed in earlier chapters, this creates a string within a string, and ActionScript needs this inner string to work properly here. A quick analysis of the code will show the reason why.

```
<mx:LinkButton x="10" y="91" label="Click Here to Email Us"➥
  color="#0000ff" click="currentState='Contact'"/>
```

You can see that currentState is the main string for the buttonDown event. The state the event is selecting, Contact, is the inner string.

18. Give your code a test by saving and running the application. When you click the LinkButton control of the initial panel, the Contact panel should appear. This can create a pretty cool way of going from page to page.
19. You are not finished yet. Close the browser and return to your work.
20. Click the Contact tag in the States panel so you can see the Contact panel.
21. Click the Button control.

22. This time, in the `buttonDown` event, type the following code into the Value column:

```
currentState=' '
```

23. Press Enter.
24. Save and test the application. When you click the `LinkButton` control, the second panel should turn on. When you click the `Button` control on that panel, it should just return back to the original panel only (base state).

As you can see, you have not lost the `gotoAndStop` functionality of the Flash timeline. Instead, you are addressing it differently.

Let's take this concept yet one step further.

25. With the `Contact` state selected, right-click the States panel and select New State.

Notice that this time you are asked whether you want to base your new state on the `Contact` state. Flex will always ask whether you want to build on whatever state you have selected.

26. Name your new state Thankyou, as shown in Figure 7-15.

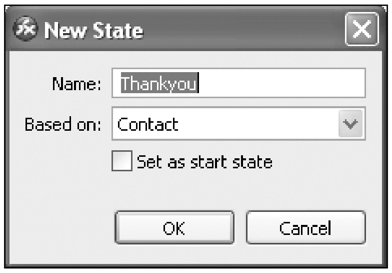


Figure 7-15. The New State dialog box for a state built over the Contact state

27. Click OK. Your States panel should now resemble Figure 7-16.

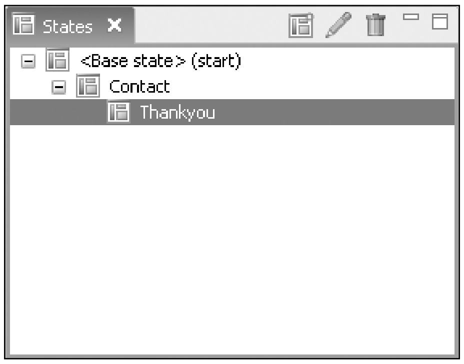
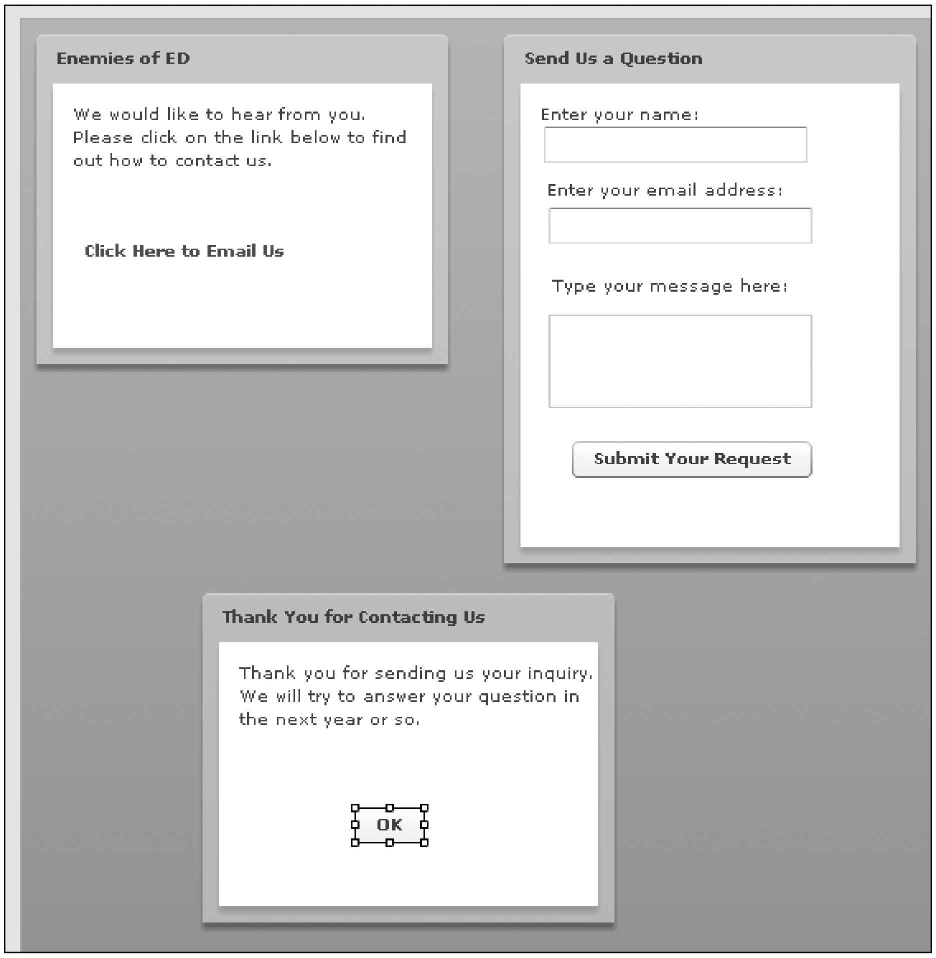


Figure 7-16. The States panel with a hierarchy of states

Notice that your new state, since it is built over the Contact state, is indented in the States panel. This makes it easy to see the hierarchy that is created by various states, as well as which states are built upon what others.

28. With the Thankyou state selected, bring a third Panel container onto the stage.

29. Give this Panel container a header of Thank You. Add some text to thank the user for submitting a question and add either a Button or LinkButton control (see Figure 7-17).



Enemies of ED

We would like to hear from you. Please click on the link below to find out how to contact us.

Send Us a Question

Enter your name:

Enter your email address:

Type your message here:

Thank You for Contacting Us

Thank you for sending us your inquiry. We will try to answer your question in the next year or so.

Figure 7-17. The finished states of the project

Now you need to make an adjustment to your application's functionality.

30. Click the Button control (or whatever control you used) in the new Panel container.

31. Set the buttonDown event to the following:

```
currentState = ' '
```

This will return the user to the default state.

32. Once you have completed that, click the Button control for the second state (Submit Your Request).

33. Change the currentState from ' ' to the following:

```
currentState='Thankyou'.
```

Needless to say, there are any number of variations you can try using these same ideas. For instance, you could have based the Thankyou state on the base state. By doing that, the e-mail form would become invisible when the Submit Your Request button is clicked.

It might be worth spending a bit of time looking at the code behind what you just did in Design view.

States and code

Throughout most of this book, I have been a strong advocate for using Source view over Design view. In my opinion, this gives you maximum control in creating your applications. However, when creating states, Design view may be the better choice, because the code can be daunting for a beginner. However, there are a couple of interesting concepts going on that you haven't seen yet. Take a look at the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"➥
  layout="absolute">
   <mx:states>
      <mx:State name="Contact">
         <mx:AddChild position="lastChild">
            <mx:Panel x="294" y="10" width="250" height="321"➥
              layout="absolute" title="Send Us a Question">
               <mx:Label x="10" y="10" text="Enter your name:"/>
               <mx:TextInput x="14" y="26"/>
               <mx:Label x="14" y="56"➥
                 text="Enter your email address:"/>
               <mx:TextInput x="17" y="75"/>
               <mx:Label x="17" y="114"➥
                  text="Type your message here:"/>
               <mx:TextArea x="17" y="140" height="57"/>
               <mx:Button x="31" y="217" label="Submit Your Request"➥
                 buttonDown="currentState='Thankyou'" id="button1"/>
            </mx:Panel>
         </mx:AddChild>
      </mx:State>
      <mx:State name="Thankyou" basedOn="Contact">
         <mx:AddChild position="lastChild">
```

```
            <mx:Panel x="111" y="349" width="250" height="200"➥
              layout="absolute" title="Thank You for Contacting Us">
               <mx:Text x="10" y="10" text="Thank you for sending us➥
                 your inquiry. We will try to answer your question in➥
                the next year or so." width="220"/>
               <mx:Button x="82" y="100" label="OK"➥
                 buttonDown="currentState=''"/>
            </mx:Panel>
         </mx:AddChild>
      </mx:State>
   </mx:states>
   <mx:Panel x="10" y="10" width="250" height="200" layout="absolute"➥
     title="Enemies of ED">
      <mx:Text x="10" y="10" text="We would like to hear from you.➥
        Please click on the link below to find out how to ➥
        contact us." width="210"/>
      <mx:LinkButton x="10" y="91" label="Click Here to Email Us"➥
        color="#0000ff" buttonDown="currentState='Contact'"/>
   </mx:Panel>
</mx:Application>
```

The UIComponents class is part of the mx:core package. Recall from earlier discussions that anything in the mx:core package is automatically available to any Flex application without having to import the package.

One of the properties of the class UIComponents is states. The states property is of type Array. But what is it an array of?

In this case, it is an array of containers.

Notice that the two states beyond the base state, Contact and Thankyou, are wrapped within an <mx:states> tag. Then, each state is wrapped in an <mx:State> tag. This last tag automatically puts the state into a container called a **child container**. The outer <mx:states> tag then creates an array of the <mx:State> containers contained within it.

As soon as you click the component that calls the currentState handler, the AddChild class calls the <mx:states> tag to find the appropriate <mx:State> container in the array. The AddChild class has a property called position, which has a default value of lastChild. This just means to put the new container wherever you indicated during the design.

When the child container is no longer needed, <mx:states> calls a class called RemoveChild. This class has the appropriate methods for toggling off the child container.

As I said at the outset, this can be a bit daunting for a novice programmer. However, to help you get your toes a bit wet, let’s try a small exercise involving rollover effects.

Rollovers and states

In traditional XHTML design, JavaScript can be used to create rollover effects. As soon as a mouse rolls over text or an image, an event handler in JavaScript catches the event, and the code instructs the browser to swap out one image for another (in the case of hyperlinks, the color of the text may change). This process, as easy and commonplace as it is by today's web standards, requires a lot of overhead in the resources used.

Flash handled the process much more efficiently by compiling, and compressing, all the necessary graphics and code into a single SWF file. Again, like the previous example, much of this was handled with the timeline. However, the timeline is no longer present in Flex.

Once again, states come to the rescue. Let's try a simple example. This time, rather than use Design view as before, you will create this example through code in the Source view.

1. Open the `Chapter7_Rollover.mxml` file found in the `Chapter7_Project` folder you should have downloaded from `www.friendsofed.com`. Figure 7-18 shows the contents of this file.

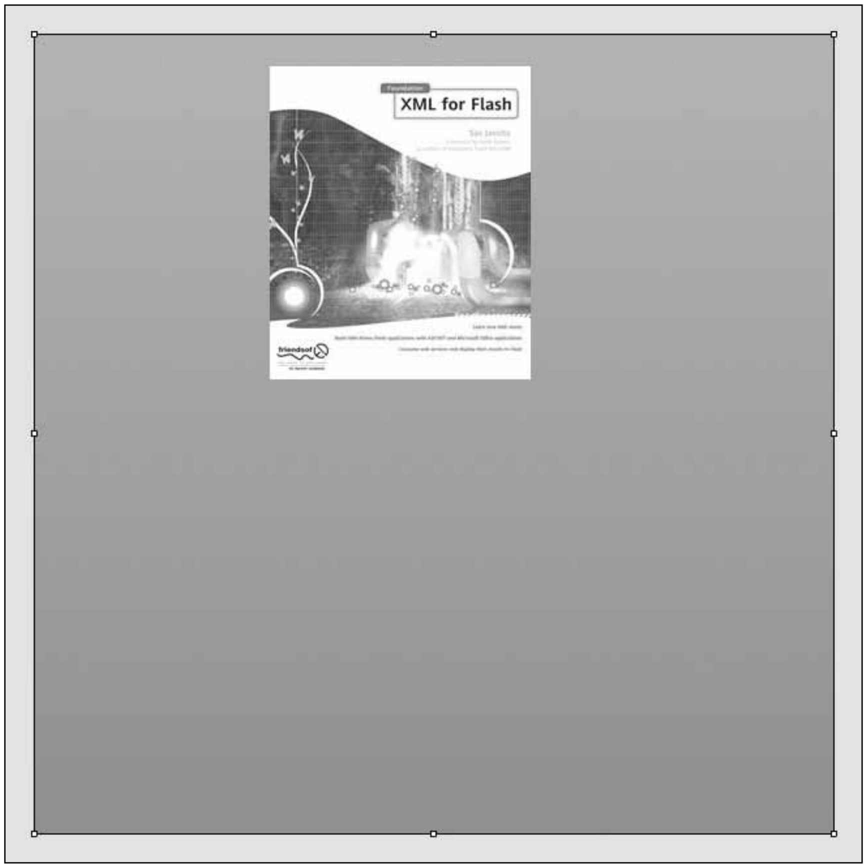


Figure 7-18. Initial state of the project file

As you can see, this is a just an Image control referencing an image in the assets folder of the project. What you want to do is create a rollover effect so that the book's details will appear when the mouse rolls over the cover image.

2. If you are not there, go into Source view, where you should see the following code. You will be working in code for this exercise.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"➥
  layout="absolute">
   <mx:Image x="181" y="25" source="assets/jacobs.jpg"/>
</mx:Application>
```

Recall that the <mx:states> tag contains the additional states that will be built over the base state. Also, recall that it creates an array of containers to hold these states.

3. Add the following code in bold above the Image control:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"➥
  layout="absolute">
<mx:states>

</mx:states>
   <mx:Image x="181" y="25" source="assets/jacobs.jpg"/>

</mx:Application>
```

Each new state you want to add must be enclosed in an <mx:State> container.

4. You must give the state a name so that it can be properly identified by ActionScript.

```
<mx:states>
   <mx:State name="bookDetails">

   </mx:State>
</mx:states>
```

5. Just to prove a point, return to Design view and look at the States panel.

The state you just created with the <mx:State> tag should be listed in the panel (see Figure 7-19). Each subsequent state added will be listed.

Again, referring to the last example, the container is added with the <mx:AddChild> class. If you really dig into the mechanics, AddChild is yet another container in that it will take its contents, which you will build in a moment, and attach it as a container within the State container you just built (bookDetails). This class

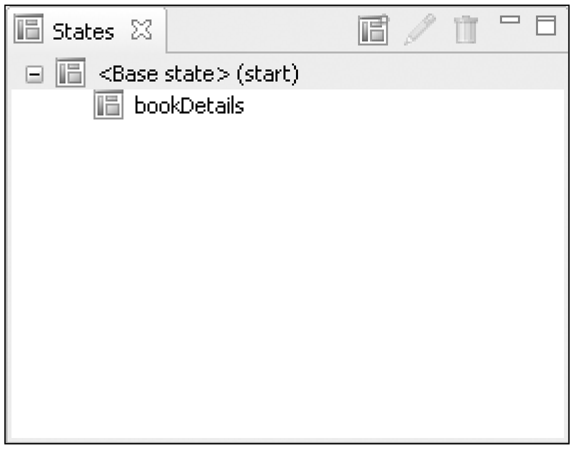


Figure 7-19. The bookDetails state listed

also decides the position of this subcontainer. The default position, as mentioned in the last section, is `lastChild`. This means the position of the `bookDetails` container, which, if you think about it, is a child of the base state (I warned you this was a bit confusing).

```
<mx:states>
   <mx:State name="bookDetails">
      <mx:AddChild position="lastChild">

      </mx:AddChild>
   </mx:State>
</mx:states>
```

You can now add your contents directly into the `AddChild` container.

In this case, you are going to keep it simple and add a `Text` control. You can add any text you want. However, if you want to use the text I show in this exercise, find the `Rollover.txt` file located with your exercise files.

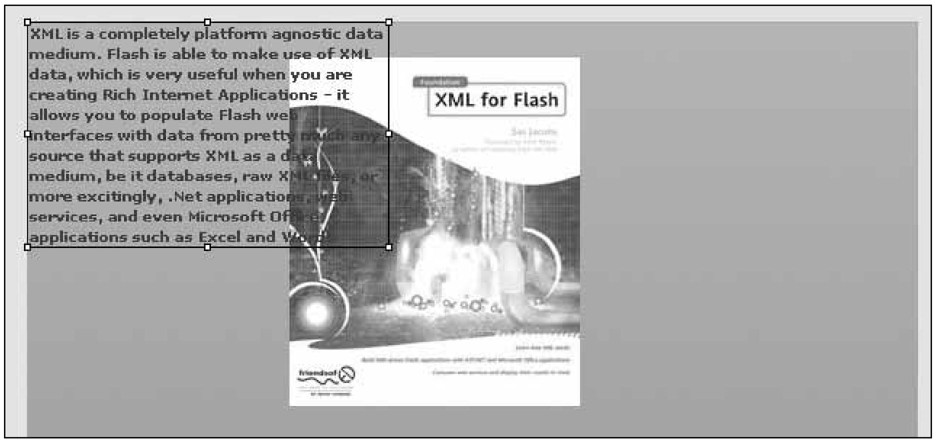
6. Add the `Text` control as follows:

```
<mx:State name="bookDetails">
   <mx:AddChild position="lastChild">
      <mx:Text width="250" fontWeight="bold" text=""
   </mx:AddChild>
</mx:State>
```

In this code example, you have not added the text yet; all you did was give the control a width and an attribute to make the text bold.

7. In the quotes of the `text` attribute, either type some text or paste in the text from the `Rollover.txt` file. Don't forget to close the `Text` control with `/>`.

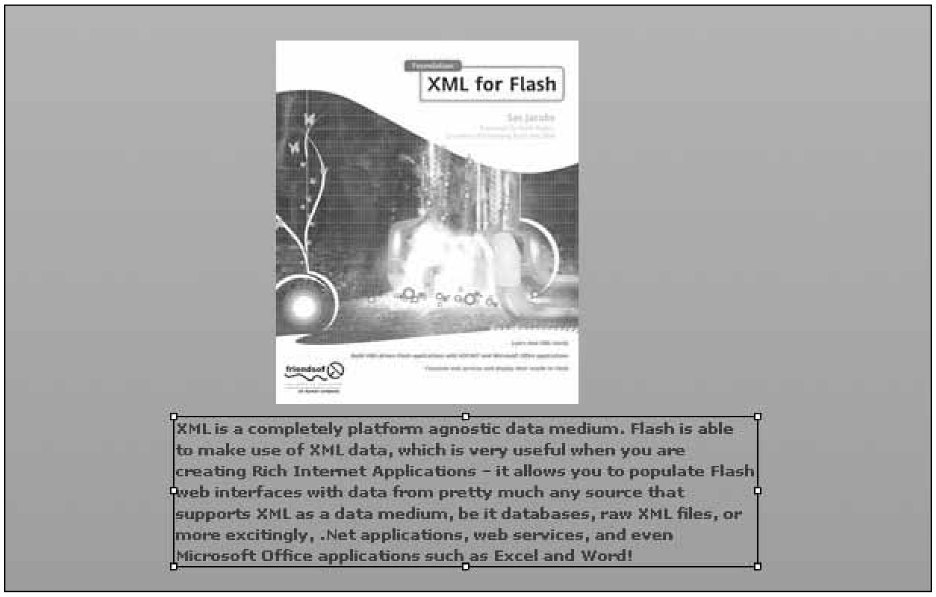
8. Just to see what you have done so far, go over to Design view and click the `bookDetails` state in the States panel. Your screen should look similar to Figure 7-20.



XML is a completely platform agnostic data medium. Flash is able to make use of XML data, which is very useful when you are creating Rich Internet Applications – it allows you to populate Flash web interfaces with data from pretty much any source that supports XML as a data medium, be it databases, raw XML files, or more excitingly, .Net applications, web services, and even Microsoft Office applications such as Excel and Word.

Figure 7-20. Initial placement of the child container

9. As you may have noticed, the placement may be less than ideal. This is not a huge problem. As with any container, just click and drag this one to the position you want. You can also adjust the width if you want. Reposition the container so that your screen looks like Figure 7-21.



XML is a completely platform agnostic data medium. Flash is able to make use of XML data, which is very useful when you are creating Rich Internet Applications – it allows you to populate Flash web interfaces with data from pretty much any source that supports XML as a data medium, be it databases, raw XML files, or more excitingly, .Net applications, web services, and even Microsoft Office applications such as Excel and Word!

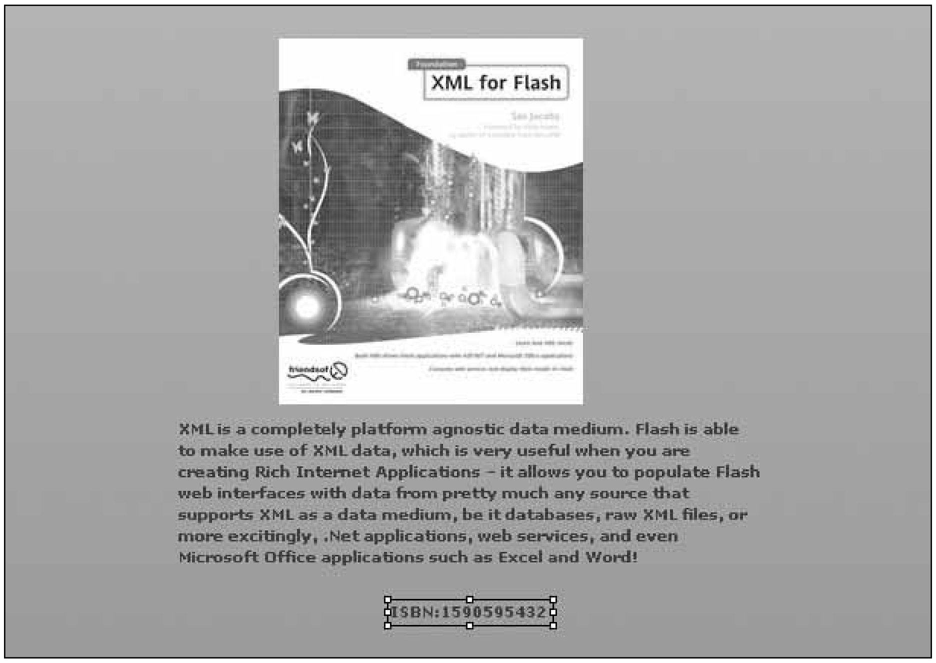
Figure 7-21. The repositioned child container

10. Now add another AddChild directly after the previous one.

```
<mx:State name="bookDetails">
   <mx:AddChild position="lastChild">
      <mx:Text width="387" fontWeight="bold" text="XML is a➥
        completely platform agnostic data medium. Flash is able to➥
        make use of XML data, which is very useful when you are➥
        creating Rich Internet Applications - it allows you to➥
        populate Flash web interfaces with data from pretty much any➥
        source that supports XML as a data medium, be it databases,➥
        raw XML files, or more excitingly, .Net applications, web ➥
        services, and even Microsoft Office applications such as ➥
        Excel and Word!" x="113" y="273"/>
   </mx:AddChild>
   <mx:AddChild position="lastChild">
      <mx:Text width="110" fontWeight="bold" text="ISBN:1590595432" />
   </mx:AddChild>
</mx:State>
```

11. As you did before, go back to Design view and position the container as you like (see Figure 7-22).

The AddChild *class has the attribute of* relativeTo *so that you can position the container relative to another component in your application. If you use that, you can set your* position *attribute to before or after that component. You will get a chance to see this in action in a bit.*



XML is a completely platform agnostic data medium. Flash is able to make use of XML data, which is very useful when you are creating Rich Internet Applications – it allows you to populate Flash web interfaces with data from pretty much any source that supports XML as a data medium, be it databases, raw XML files, or more excitingly, .Net applications, web services, and even Microsoft Office applications such as Excel and Word!

ISBN:1590595432

Figure 7-22. The second AddChild container positioned properly

Hopefully, you are now beginning to see how the AddChild class adds subcontainers within the State container.

You may be thinking that you could have added these Text controls directly into the State container. However, I strongly suggest that you don't do that. You will be sacrificing some interesting design possibilities that you will learn about later as you progress through the book.

It is now time to give your application some functionality.

12. Begin by giving your Image control an ID.

```
<mx:Image x="181" y="25" source="assets/jacobs.jpg" id="bookCover"/>
```

This is a good habit to get into whenever you work with ActionScript.

13. Recall that in your first exercise you assigned the currentState handler to the event buttonDown. You will do a slight variation here and assign this handler to a rollOver event for the image.

```
<mx:Image x="181" y="25" source="assets/jacobs.jpg" id="bookCover"➥
  rollOver="currentState='bookDetails'"/>
```

Please remember to enclose bookDetails *in single quotes as discussed earlier.*

14. Save and test the application. As soon as you roll over the image, your AddChild containers and their contents should appear.

You now have one slight problem. When you roll off of the image, the containers remain.

15. Remedy this problem by adding a second event to the Image control as follows:

```
<mx:Image x="181" y="25" source="assets/jacobs.jpg" id="bookCover"➥
rollOver="currentState='bookDetails'" rollOut="currentState=''"/>
```

Again, use single quotes after currentState.

16. Save and run your application now. You should see the text disappear after you roll the mouse away from the image.

Let's try a small variation in order to see an interesting programming tool in Flex Builder 2.

17. Delete the rollOut event in the Image control.

18. Right below the <mx:State> tag, start the tag shown here in bold:

```
<mx:State name="bookDetails" id="bookDetails">
   <mx:SetEventHandler
```

The SetHandler class allows you to define events outside of the component creating or dispatching the event. While this is a relatively simple example, you will be using this in increasingly complex situations—for instance, assigning multiple events and transitions to a state.

19. The first attribute you want to set is target. This is the component that will be dispatching the event. In this case, it will be the Image control with the id attribute of bookCover.

```
<mx:SetEventHandler target="{bookCover}"
```

20. The next attribute you need to set is name. This attribute should contain the event you want to trigger in the state. In this particular example, you want a rollOut event.

```
<mx:SetEventHandler target="{bookCover}" name="rollOut"
```

21. The final attribute you want to set is handler, which indicates what you want to happen when the event occurs. In this case, you specify currentState=' '. After this attribute, you can close out the tag.

```
<mx:SetEventHandler target="{bookCover}" name="rollOut"➥
  handler="currentState=''" />
```

22. Test your code now. It should work exactly the same way when you roll your mouse out.

When I give Flex seminars, I show the power of the SetEventHandler using an example similar to this. And, invariably, I am asked whether the rollOver event can be set up the same way. I have the attendees set it up and, when they go to test it, nothing happens.

The reason can be understood with a bit of logic. The SetEventHandler class is invoked within the State container. This means that it is not active until the State it is in is active. In this case, the State is not active until the rollOver event occurs. So you would be trying to call something that programmatically does not exist yet. It is a bit of cyclical thinking.

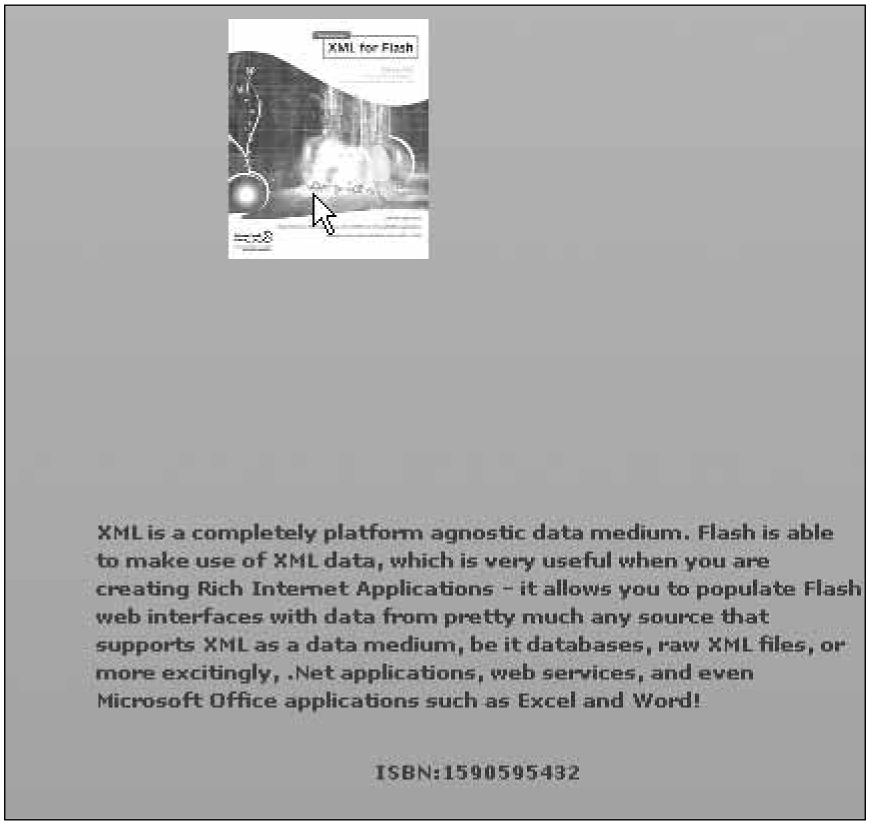
Along with SetEventHandler, there is the SetProperty event. The syntax for this event is very similar in that the target attribute will be the component whose properties you want to change. Likewise, name is the property you want to change.

In this example, you want to reduce the size of the image by 50% when the state is activated. This will require that you add two SetProperty instances: one for the scaleX property and one for the scaleY property. Where things change a bit is that you need to set a value attribute in place of the handler attribute used in SetEventHandler.

23. Make the following changes in bold to your code:

```
<mx:SetEventHandler target="{bookCover}" name="rollOut"➥
  handler="currentState=''" />
<mx:SetProperty target="{bookCover}" name="scaleX" value=".50" />
<mx:SetProperty target="{bookCover}" name="scaleY" value=".50" />
```

24. Run the application, and you should see results similar to those in Figure 7-23.



XML is a completely platform agnostic data medium. Flash is able to make use of XML data, which is very useful when you are creating Rich Internet Applications – it allows you to populate Flash web interfaces with data from pretty much any source that supports XML as a data medium, be it databases, raw XML files, or more excitingly, .Net applications, web services, and even Microsoft Office applications such as Excel and Word!

ISBN:1590595432

Figure 7-23. The effect with the mouse rolled over the picture

In all of these examples, the states seem to be making a rather ungraceful entrance and exit. Let's see how you could smooth things up a little bit.

Transitions

Once again, to address the loss of the Flash timeline in Flex, Flex has a corresponding feature associated with states: **transitions**.

A transition is a way to gracefully turn a state on or off. There is no way to set transitions up in Design view, and, as you are about to see, they require a bit of programming. The subject of transitions could take up an entire book; though a detailed discussion of transitions is outside the scope of this book, the following example will hopefully give you a good introduction.

Building transitions is actually a three-step process:

1. Build the container.
2. Create the states.
3. Program the transitions.

You have already done the first two steps in the previous exercises. So, in this exercise, you are going to work with a slightly more complex scenario. You will give the container some animation.

Building the container

You'll start this exercise by building the container you will eventually animate.

1. Start a new MXML file in your project and give it a name of your choice.

You are going to build a container to hold the information for the XML book you have been working with. This part will be pretty straightforward.

2. Change your MXML code to the following:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"➥
  layout="absolute">
   <mx:Panel title="Book" id="book" horizontalScrollPolicy="off"➥
     verticalScrollPolicy="off">
      <mx:Form id="bookForm">
        <mx:FormItem label="Foundation XML for Flash"➥
          fontWeight="bold" />
        <mx:FormItem label="Sas Jacobs" fontStyle="italic" />
      </mx:Form>
   </mx:Panel>
</mx:Application>
```

Your initial Panel container should appear as shown in Figure 7-24.

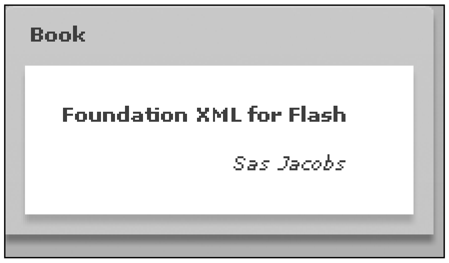


Figure 7-24. The initial Panel container

Next you need to add a LinkButton and Label control to the Panel container. As you saw in earlier chapters, the easiest way to do that is with a ControlBar. The purpose of the ControlBar is to provide a container for adding any controls that you might need.

You want to be certain that there is a space between the LinkButton and Label controls. In order to do this, you are going to call on a little-known class in Flex called Spacer.

In the past, HTML designers used to use spacers, which were usually a 1-pixel–by–1-pixel transparent GIF image, for a variety of needs, including preventing empty table cells from collapsing. In Flex, Spacer is an entire class whose main purpose is to allocate space in a parent container. In this case, the parent container is ControlBar.

In situations where the size of that parent container could be variable, the Spacer class has a number of properties for setting height, width, maximum height and width, minimum height and width, percent height and width, and so on. It would be well worth your time to study the documentation for this class. I am finding that it is coming in handy for a variety of situations.

3. Add the following code to the Panel container you just created:

```
<mx:Panel title="Book" id="book" horizontalScrollPolicy="off"➥
  verticalScrollPolicy="off">
      <mx:Form id="bookForm">
        <mx:FormItem label="Foundation XML for Flash"➥
          fontWeight="bold" />
        <mx:FormItem label="Sas Jacobs" fontStyle="italic" />
      </mx:Form>
      <mx:ControlBar>
        <mx:LinkButton label="Book Details" id="bookLink" />
        <mx:Spacer width="100%" id="spacer1"/>
        <mx:Label text="Book Title" id="title"/>
    </mx:ControlBar>
   </mx:Panel>
```

Your Panel container should now look like Figure 7-25.

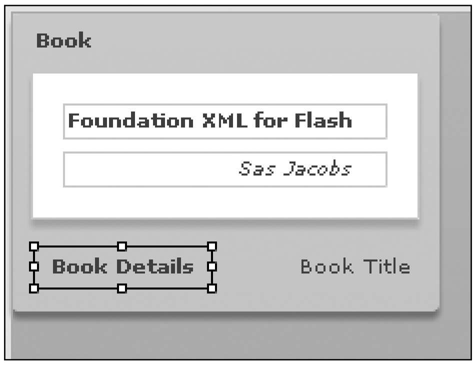


Figure 7-25. The completed Panel container

Now that the container is completed, your next step is to build the state for it.

Building the states

Recall from earlier in the chapter that states must be enclosed within the `<mx:states>` tags that create an array of containers. Each of those containers represent a different state and must be enclosed in `<ms:State>` tags.

Most of the following code example is similar to the previous states you built.

1. Place the following code before the `Panel` code you created earlier.

In this case, code placement is not critical. I am suggesting the placement so it will be easier to follow along with the book.

```
<mx:states>
        <mx:State name="bookDetails" basedOn="">
            <mx:AddChild relativeTo="{bookForm}" position="lastChild"➥
              creationPolicy="all" >
                <mx:FormItem id="isbn" label="ISBN: 1590595432" />
            </mx:AddChild>
            <mx:SetProperty target="{book}" name="title"➥
              value="Book Details"/>
            <mx:SetProperty target="{title}" name="text"➥
              value="Book Details"/>
            <mx:RemoveChild target="{bookLink}"/>
            <mx:AddChild relativeTo="{spacer1}" position="before">
                <mx:LinkButton label="Collapse Book Details"➥
                  click="currentState=''"/>
            </mx:AddChild>
        </mx:State>
    </mx:states>
```

After the opening `<mx:states>` you create the state container, named `bookDetails`, using the `<mx:State>` tag.

As mentioned previously, the `AddChild` class adds a new container, and each container can contain any content that you might need. However, here you are using it a bit differently from the earlier examples.

In this example, you are adding the `relativeTo` attribute to the `AddChild` class to add the container to the `<mx:Form id="bookForm">` container you created earlier. You are also telling it to position this container, containing form items, as the last child in the form.

The `creationPolicy` attribute decides when the child container is created.

Notice I said "created" and not "added." When it is created, it is just held in memory until called.

The creationPolicy attribute has three possible values. The default is auto. An auto value means that the child is created when the state is activated. The all value, which you use here, means that the child is created when the application is started. When doing transitions, you might find that caching the child containers will facilitate smoother transitions since the pieces will be in place already. The none value means that the child will not be created until a method, createInstance(), is called to specifically create it. This can be handy in certain advanced programming scenarios.

You are using the all value here to help make a smoother transition.

Like any container, once the child container is created, you can put whatever content you want into it. For the sake of simplicity in this exercise, you just put a label into it showing the ISBN number of the book.

After the AddChild container is created, you use the SetProperty class discussed earlier in the chapter. Notice that, in this case, you are using two instances of the class to change the title value of the Panel container, book, and the text attribute of the Label control on the ControlBar, title. Remember, these actions will not occur until the state is activated.

The next few lines are where things start to become a bit different from before.

The <mx:RemoveChild> can literally remove any component or container from the user interface. In this case, you are telling it to remove the LinkButton control, bookLink, on the ControlBar.

Once the initial LinkButton is removed, you put a whole new one in its place. Notice that you position the new control before the Spacer, spacer1, using a combination of the relativeTo and position attributes. This raises an interesting programming issue: could you have used the SetProperty and SetEventHandler classes to do the same thing?

The answer to that is yes. In this case, either technique would have worked. The decision as to which technique to use is largely a matter of programming style and needs. As you progress through the book, and as you learn new techniques, you may want to go through previous exercises and try your newfound knowledge.

If you switch to Design view and activate the bookDetails state, your UI should look something like Figure 7-26.

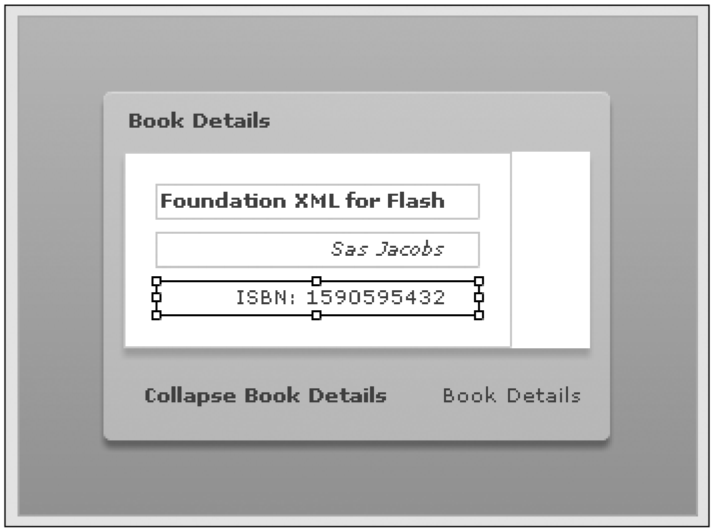


Figure 7-26. The design of the bookDetails state

Notice the change in the panel's title and the label's text. Also, there is an additional control in the form.

2. You have one more small thing to do. You need to tell the initial `LinkButton` control, located in the `ControlBar`, to switch to the `bookDetails` state when clicked.

   ```
   <mx:LinkButton label="Book Details" id="bookLink"➥
   click="currentState='bookDetails'" />
   ```

   Again, do not forget to use single quotes around the name of the state.

At this point, your code should look as follows:

```
<mx:states>
        <mx:State name="bookDetails" basedOn="">
            <mx:AddChild relativeTo="{bookForm}" position="lastChild"➥
              creationPolicy="all" >
                <mx:FormItem id="isbn" label="ISBN: 1590595432" />
            </mx:AddChild>
            <mx:SetProperty target="{book}" name="title"➥
              value="Book Details"/>
            <mx:SetProperty target="{title}" name="text"➥
              value="Book Details"/>
            <mx:RemoveChild target="{bookLink}"/>
            <mx:AddChild relativeTo="{spacer1}" position="before">
                <mx:LinkButton label="Collapse Book Details"➥
                 click="currentState=''"/>
            </mx:AddChild>
        </mx:State>
    </mx:states>

    <mx:Panel title="Book" id="book" horizontalScrollPolicy="off"➥
      verticalScrollPolicy="off">
    <mx:Form id="bookForm">
        <mx:FormItem label="Foundation XML for Flash"➥
          fontWeight="bold" />
        <mx:FormItem label="Sas Jacobs" fontStyle="italic" />
    </mx:Form>


    <mx:ControlBar>

        <mx:LinkButton label="Book Details" id="bookLink"➥
          click="currentState='bookDetails'" />

        <mx:Spacer width="100%" id="spacer1"/>
        <mx:Label text="Book Title" id="title"/>

    </mx:ControlBar>

    </mx:Panel>
```

3. Go ahead and test the application. You should be able to change states as you did in earlier exercises.

Your next step is to start creating a transition so your state enters and exits gracefully.

Creating transitions

In many respects, the syntax for creating transitions is similar to the syntax for states. You are going to use a tag, `<mx:transitions>`, to create an array of transitions. You can have as many transitional effects as you want. In addition, you can make a decision whether you want them to play sequentially or parallel. You are going to add several to help make going from state to state a bit smoother.

I will give you all the code for the transition first and then discuss it line-by-line.

1. For this example, place the following code above the state code:

```
<mx:transitions>
       <mx:Transition fromState="*" toState="*">
             <mx:Parallel targets="{[book, bookLink, title, isbn]}">
                <mx:Resize duration="500" easingFunction=➥
                 "Bounce.easeOut"/>
                <mx:Sequence target="{isbn}">
                    <mx:Blur duration="200" blurYFrom="0.0"➥
                      blurYTo="20.0" />
                    <mx:Blur duration="200" blurYFrom="20.0"➥
                      blurYTo="0.0" />
                </mx:Sequence>
            </mx:Parallel>
        </mx:Transition>
    </mx:transitions>
```

Just as the `<mx:states>` tag creates an array of states, the `<mx:transitions>` tag creates an array of transitions within it.

And, just as the `<mx:State>` creates a new state, `<mx:Transition>` creates a new transition. However, the `fromState` and `toState` attributes create an interesting programming possibility. Let's create a hypothetical situation here and say that your application has four states: `stateA`, `stateB`, `stateC`, and `stateD`. You could specify to use this particular transition only when going from `stateB` to `stateC` as follows:

```
<mx:Transition fromState="stateB" toState="stateC">
```

By using the asterisk, you are telling Flex to use this transition from any state to any state.

With transitions, you can choose whether they perform all together, in **parallel**, or one after the other, in **sequence**.

In this example, you choose to have parallel performance by selecting the `Parallel` class. But now you have to tell Flex what components you want to run in parallel. You do that with the `targets` attribute.

```
<mx:Parallel targets="{[book, bookLink, title, isbn]}">
```

The syntax is important here. The Parallel class is setting up its own array or components. Thus the [ ], which is array notation. So components book, bookLink, title, and isbn will all transition at the same time.

Within that Parallel container, you now need to specify the actions these components will perform. There are a large number of classes you could call here. But the one I decided to demonstrate is the Resize class.

```
<mx:Resize duration="500" easingFunction="Bounce.easeOut"/>
```

The first attribute you add is duration. This will decide how long the resizing transition will take measured in milliseconds. So 500 milliseconds translates to 5 seconds.

The easingFunction attribute varies the speed of the animation and goes back to a feature of Flash. The best analogy is that of a ball. If you ever follow the calculus of throwing a ball in the air, gravity will cause it to slow as it rises and accelerate as it descends. This variation of velocity is called **easing**. For a great discussion on this topic, go to www.ericd.net/chapter7.pdf.

Rather than specify a value directly, you are going to let the value of the easingFunction be controlled by another class: Bounce.

The Bounce class does just as it says: it causes the easing to bounce like a ball. The easeOut property causes the bounce to begin quickly and then slow down. So, in this case, the resize transition will last 5 seconds and end with a bounce that will begin quickly and then start to slow down.

If you were to look in the documentation for the Bounce *class, you would not see the* easeOut *property at first. You need to expand the* inherited *properties to see it.*

There is one quirk to using the Bounce class that you need to know. The Bounce class is part of the mx.effects.easing package. This package must be imported within a Script tag in order for you to use the Bounce class.

2. Right under the opening Application tag put the following Script tag:

```
<mx:Script>
        <![CDATA[
            import mx.effects.easing.Bounce;
        ]]>

    </mx:Script>
```

While all of this is going on, you run a second animation in the isbn label. You use the Sequence class instead of Parallel.

```
<mx:Sequence target="{isbn}">
```

An old trick of animators is to create a sense of motion by blurring an object and then bringing it back into focus. The Blur class does just that by creating a graphics effect called a Gaussian blur.

```
<mx:Sequence target="{isbn}">
      <mx:Blur duration="200" blurYFrom="0.0" blurYTo="20.0" />
       <mx:Blur duration="200" blurYFrom="20.0" blurYTo="0.0" />
</mx:Sequence>
```

The amount of blur you apply ranges from 0.0 to 255.0. In addition, you can apply blur either along the x- or the y-axis.

Looking at the preceding code, it should be fairly obvious why you did this sequentially. You first change the blur of the control from 0.0 to 20.0 over a time of 2 seconds. The second call to the Blur class reverses the process going from 20.0 to 0.0 over 2 seconds.

3. Now that you have the transitions all set, take a quick review of all of your code:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml" verticalAlign="middle"
    width="340" height="250"
    viewSourceURL="src/DefiningStateTransitions/index.html">
    <mx:Script>
        <![CDATA[
            import mx.effects.easing.Bounce;
        ]]>

    </mx:Script>

    <mx:transitions>
       <mx:Transition fromState="*" toState="*">
             <mx:Parallel targets="{[book, bookLink, title, isbn]}">
                <mx:Resize duration="500"➥
                  easingFunction="Bounce.easeOut"/>
                <mx:Sequence target="{isbn}">
                    <mx:Blur duration="200"➥
                      blurYFrom="0.0" blurYTo="20.0" />
                    <mx:Blur duration="200"➥
                      BlurYFrom="20.0" blurYTo="0.0" />
                </mx:Sequence>
            </mx:Parallel>
        </mx:Transition>
    </mx:transitions>
    <mx:states>
        <mx:State name="bookDetails" basedOn="">
            <mx:AddChild relativeTo="{bookForm}"➥
              position="lastChild" creationPolicy="all" >
                <mx:FormItem id="isbn" label="ISBN: 1590595432" />
                </mx:AddChild>
                <mx:SetProperty target="{book}" name="title"➥
                  value="Book Details"/>
                <mx:SetProperty target="{title}" name="text"➥
                  value="Book Details"/>
                <mx:RemoveChild target="{bookLink}"/>
                <mx:AddChild relativeTo="{spacer1}" position="before">
                    <mx:LinkButton label="Collapse Book Details"➥
                      click="currentState=''"/>
                </mx:AddChild>
            </mx:State>
        </mx:states>

        <mx:Panel title="Book" id="book" horizontalScrollPolicy="off"➥
          verticalScrollPolicy="off">
        <mx:Form id="bookForm">
            <mx:FormItem label="Foundation XML for Flash"➥
              fontWeight="bold" />
            <mx:FormItem label="Sas Jacobs" fontStyle="italic" />
        </mx:Form>


        <mx:ControlBar>

            <mx:LinkButton label="Book Details" id="bookLink"➥
              click="currentState='bookDetails'" />

            <mx:Spacer width="100%" id="spacer1"/>
            <mx:Label text="Book Title" id="title"/>

        </mx:ControlBar>

        </mx:Panel>
    </mx:Application>
```

If all looks well, go ahead and give the code a test drive.

When you click the LinkButton control, you should see all the changes of the new state taking place. When you return to the initial state, the changes reverse.

While these transitions are fun and, once you get the idea, easy to program, I strongly recommend that you plan them out before you start to program. In very complex situations, they can get quite involved.

And you thought you lost the timeline?

Summary

I started off this chapter by saying that you could regard states as a replacement for the Flash timeline. I think after going through these pages you will agree with me. If you're interested in reading more about animation techniques achieved without a timeline, check out *Foundation ActionScript 3.0 Animation: Making Things Move!* by Keith Peters (friends of ED, 2007).

You are now ready to put all of the knowledge you've gained so far to work by creating a project from scratch in the next chapter.