

Importing Flash CS3 Assets into Flex

Adobe Flash CS3 Professional creates SWF files compatible with Adobe Flash Player 9. You can import assets from Flash CS3 Professional to use in your Adobe Flex applications.

Contents

Installing the Adobe Flex Component Kit for Flash CS3	779
About importing SWF 9 assets into Flex	780
Importing static SWF 9 assets	780
Importing dynamic SWF 9 assets	782
Using Flash components in a Flex application	788

Installing the Adobe Flex Component Kit for Flash CS3

This section describes the system requirements and installation instructions for the Flex Component Kit for Flash CS3.

System requirements

Your system must meet the following requirements to install the Flex Component Kit for Flash CS3:

- Flex 2.01 with Updater 1 and the necessary JAR files for the Flex Component Kit for Flash CS3
- Flash CS3 Professional
- Adobe Extension Manager for Flash CS3

Installation procedure

The Flex Component Kit for Flash CS3 is a single ZIP file that contains two directories and the FlexIntegration.mxp file.

Install the Flex Component Kit for Flash CS3

1. Download the Flex Component Kit for Flash CS3 from Adobe Labs.
2. Follow the installation instructions on Adobe Labs to install the Flex Component Kit for Flash CS3.
3. Double-click the FlexIntegration.mxp file to install the necessary files in Flash CS3.

This file installs the SWC file that contains the UIMovieClip class, and adds the Make Flex Component option to the Command menu. For more information, see [“Importing dynamic SWF 9 assets” on page 782](#).

About importing SWF 9 assets into Flex

You can import SWF 9 assets created by Adobe Flash CS3 Professional into Flex applications. The way you import these assets depends on the type of asset and where you use the asset in your Flex application.

You typically import two types of SWF 9 assets:

Static assets Assets used for simple artwork or skins that do not contain any ActionScript 3.0 code. You import them in the same way that you import SWF 8 assets. For more information, see [“Importing static SWF 9 assets” on page 780](#).

Dynamic assets Assets that correspond to Flex components and contain ActionScript 3.0 code. These components are designed to work with Flex features such as view states and transitions, skinning, and tool tips. To use dynamic assets in a Flex application, export the assets to a SWC file, and then link the SWC file to your Flex application. For more information, see [“Importing dynamic SWF 9 assets” on page 782](#).

Importing static SWF 9 assets

The "Embedding Assets" chapter in the *Flex 2 Developer's Guide* describes how to import SWF 8 files into a Flex application. You use the same process to import static SWF 9 assets as you do with SWF 8 assets, where static assets are simple artwork or skins that do not contain any ActionScript 3.0 code.

The following example shows how to import static assets created in Flash CS3:

```
<?xml version="1.0"?>
<!-- embedSWF9/EmbedSimpleSWF9Movie..xml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" height="1000" >

    <mx:Script>
        <![CDATA[

                // Embed the SWF file and the symbols from the SWF file. -->
                [Embed(source="../assets/circlesquare.swf")]
                [Bindable]
                public var LogoCls:Class;

                // Embed the BlueSquare symbol. -->
                [Embed(source="../assets/circlesquare.swf", symbol="BlueSquare")]
                [Bindable]
                public var LogoClsBlueSquare:Class;

                // Embed the GreenCircle symbol. -->
                [Embed(source="../assets/circlesquare.swf", symbol="GreenCircle")]
                [Bindable]
                public var LogoClsGreenCircle:Class;
            ]]>
    </mx:Script>

    <!-- Load the SWF file using the Image control. -->
    <mx:Image source="{LogoCls}"/>

    <!-- Load the SWF file as the icon for a Button control. -->
    <mx:Button icon="{LogoCls}" height="100" width="200"/>

    <!-- Load the BlueSquare symbol as the icon for a Button control. -->
    <mx:Button icon="{LogoClsBlueSquare}" height="100" width="100"/>

    <!-- Load the GreenCircle symbol as the icon for a Button control. -->
    <mx:Button icon="{LogoClsGreenCircle}" height="100" width="100"/>

    <!-- Use the SWF symbols to skin a Button control. -->
    <mx:Button id="b1" label="Click Me"
        upSkin="@Embed(source='../assets/circlesquare.swf',
            symbol='BlueSquare')"
        overSkin="@Embed(source='../assets/circlesquare.swf',
            symbol='GreenCircle')"
        downSkin="@Embed(source='../assets/circlesquare.swf',
            symbol='BlueSquare')"/>
</mx:Application>
```

Importing dynamic SWF 9 assets

Dynamic SWF 9 assets are Flex components that you create in Flash CS3, and then import into your Flex application. These assets are Flex components, so you can use them in the same way as you would a component shipped with Flex, or a custom component that you created as an MXML file or as an ActionScript class.

To create a Flash component, you define a symbol in your FLA file, and configure the symbol as a subclass of the `mx.flash.UIMovieClip` class. To integrate the Flash components into your Flex application, you publish your FLA file as a SWC file.

For example, you define a FLA file with four symbols that represent the following four skins of the Button class:

- Button disabled skin
- Button down skin
- Button over skin
- Button up skin

Each symbol corresponds to a different subclass of the `mx.flash.UIMovieClip` class. When you publish your FLA file as a SWC file, the SWC file contains a class definition for each symbol. You can then reference the classes from your Flex application.

Use the following process to import dynamic SWF 9 assets is:

1. Install the Flex Component Kit for Flash CS3. For more information, see [“Installing the Adobe Flex Component Kit for Flash CS3” on page 779](#).
2. Create symbols for your dynamic assets in the FLA file. For more information, see [“Creating the symbol for your component” on page 783](#).
3. Run `Commands > Make Flex Component` to convert your symbol to a subclass of the `UIMovieClip` class and to configure the Flash CS3 publishing settings for use with Flex. For more information, see [“Actions performed by the Make Flex Component command” on page 785](#).
4. Publish your FLA file as a SWC file.
5. Reference the class name of your symbols in your Flex application as you would for any class. For more information, see [“Compiling your Flex application” on page 786](#).
6. Include the SWC file in your `library-path` when you compile your Flex application. For more information, see [“Compiling your Flex application” on page 786](#).

Class hierarchy of UIMovieClip

Components created in Flash for use in Flex are subclasses of the `mx.flash.UIMovieClip` class, where `UIMovieClip` is a subclass of the `flash.display.MovieClip` class. Shown below is the definition of the `UIMovieClip` class:

```
class UIMovieClip extends MovieClip
    implements IDeferredInstantiationUIComponent, IToolTipManagerClient
```

`UIMovieClip` implements the interfaces necessary for a Flash component to be used like any other Flex component. Therefore, you can use a subclass of `UIMovieClip` can be used as a child of a Flex container or as a skin, and it can respond to events, define view states and transitions, and work with effects in the same way as any Flex component.

When you install the Flex Component Kit for Flash CS3, you also install the `UIMovieClip.swc` file. This SWC file contains all of the information necessary to use the `UIMovieClip` class in Flash.

For more information on the `mx.flash.UIMovieClip` class and its interfaces, see *Adobe Flex 2 Language Reference*.

About SWC files

You deploy your Flash components in a SWC file. A SWC file is an archive file for Flex components and other assets. SWC files contain a SWF file and a `catalog.xml` file. SWC files make it easy to exchange components and other assets among Flex developers. You exchange only a single file, rather than the MXML or ActionScript files and images and other resource files.

The SWF file in a SWC file is compiled, which means that the code is loaded efficiently and it is hidden from casual view. Also, compiling a component as a SWC file can make namespace allocation an easier process.

For more information on SWC files, see *Building and Deploying Flex 2 Applications*.

Creating the symbol for your component

To create a component in Flash, create a symbol in your FLA file that corresponds to the component. Then, use the Make Flex Component command to convert the symbol into a Flex component.

Create a symbol in Flash:

1. Ensure that you installed the Flex Component Kit for Flash CS3. For installation instructions, see [“Installing the Adobe Flex Component Kit for Flash CS3” on page 779](#).

2. Select File > New to open the New Document dialog box.
3. Select Flash File (ActionScript 3.0) as the file type.
4. Create a symbol.
 - When you create a symbol, Flash prompts you to specify the symbol name. By default, the symbol name becomes the class name of your Flex component. Also, in Flex Builder, your component's class name is displayed in code hints.

However, the Make Flex Component command removes all nonalphanumeric characters (including spaces) from the class name and, if the first character is a number, it prefixes the class name with an underscore. Therefore, it is recommended that you use only uppercase and lowercase letters in the symbol name, and do not use spaces.
 - Set the registration point to the upper-left corner of the symbol.

If you have Flash content that requires a different registration point, you can wrap the symbol in another symbol with an upper-left registration point. For more information, see [“Positioning a Flash component in a Flex container” on page 788](#).
5. Select the symbol in the Library pane.
6. Select Commands > Make Flex Component.

This command converts the symbol into a subclass of the UIMovieClip class, and configures the publishing settings of your FLA file for use with Flex. For more information on the actions performed by this command, see [“Actions performed by the Make Flex Component command” on page 785](#).

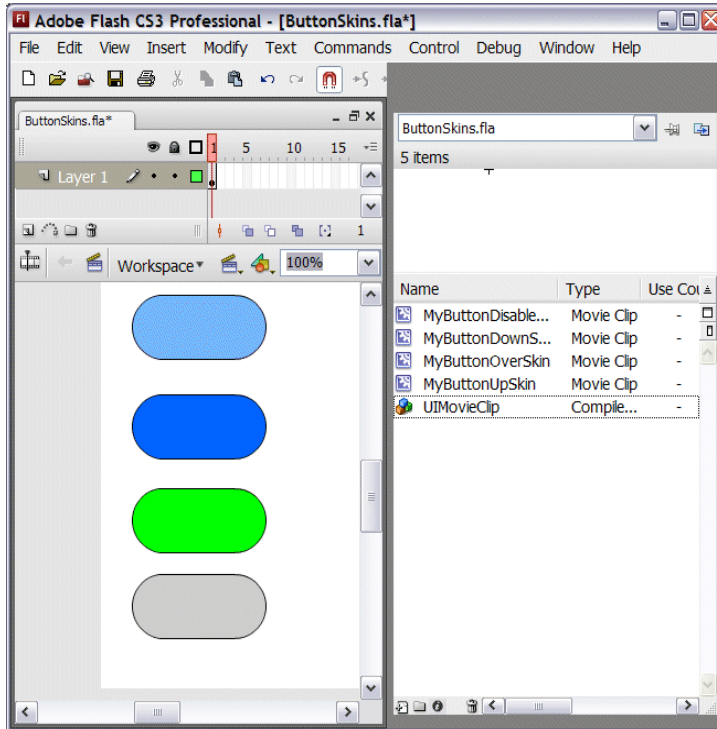
The Make Flex Component prompts you to set the frame rate of the FLA file to match the default Flex frame rate, which is 24 frames per second. Although you can set the frame rate to a different value, setting it below 24 frames per second can make the Flex application appear sluggishness.

After you run this command, UIMovieClip appears in the Library pane of your Flash application.
7. Select File > Publish to create the SWC file.

In the following example, you define four symbols that correspond to the following four skins of the Button class:

- Button disabled skin: MyButtonDisabledSkin symbol
- Button down skin: MyButtonDownSkin symbol
- Button over skin: MyButtonOverSkin symbol
- Button up skin: MyButtonUpSkin symbol

The following image shows these symbol definitions for the skins in Flash CS3 after running the Make Flex Component command:



Actions performed by the Make Flex Component command

The Make Flex Component command performs the following actions to convert a Flash symbol into a Flex component:

- Removes all nonalphanumeric characters (including spaces) from the class name and, if the first character is a number, it prefixes the class name with an underscore. Therefore, it is recommended that you use only uppercase and lowercase letters in the symbol name, and do not use spaces.
- Sets the base class of the symbol to the `mx.flash.UIMovieClip` class. In Flash, you can right click on the symbol name in the Library pane to open the Linkage Properties dialog box to see this settings.
- Configures the following properties for the symbol:
 - Sets the Export for ActionScript option
 - Sets the Export in First Frame option

- Sets the Class name to be the same as the symbol name
 - Configures the following publishing settings for the FLA file:
 - Sets the Version to Flash Player 9
 - Set the ActionScript version to ActionScript 3.0
 - Selects the Permit Debugging option to let you debug your component in Flex
 - Selects the Export SWC option
- You can see these settings by using the Select File > Publishing Settings command to open the Publishing Settings dialog box, and then selecting the Flash tab.
- Sets the Frame Rate of the FLA file to match the default Flex frame rate, which is 24 frames per second. Although you can set the Flex frame rate to a different value, setting it below 24 frames per second can make the application appear sluggishness.

NOTE

Instead of using the Make Flex Component command, you can manually configure a symbol to be used as a Flex component by making all of the settings described in this section. However, before you can reference the UIMovieClip class, you must open the Components pane and drag the UIMovieClip into the Library pane.

Compiling your Flex application

After you create the SWC file that contains your Flash components, you can use the components in your Flex application. For example, the following application uses the four skin components created in the previous section in a Flex application:

```
<?xml version="1.0"?>
<!-- skins/EmbedSWF9ButtonSkins.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" height="1500"
  xmlns:myComps="*">

  <mx:Button label="Click Me"
    upSkin="MyButtonUpSkin"
    overSkin="MyButtonOverSkin"
    downSkin="MyButtonDownSkin"
    disabledSkin="MyButtonDisabledSkin"/>

  <myComps:MyButtonUpSkin/>
  <myComps:MyButtonOverSkin/>
  <myComps:MyButtonDownSkin/>
  <myComps:MyButtonDisabledSkin/>

</mx:Application>
```

In this example, you use the Flash components to skin the Button control and use the components as stand-alone Flex components.

To use a Flash component as stand-alone Flex component, the main application file must include a namespace definition. The namespace tells Flex where to look for the file that implements the component. By default, Flex creates the class that corresponds to your Flash component in the default package, corresponding to a namespace of "*".

If you define a package name for your Flash component, you must specify the namespace accordingly. For example, suppose you define the package name for your Flash component as the following example shows:

```
package flashComponents {  
    // Component definition  
}
```

You specify the namespace as "flashComponents.*".

Compile your application in Flex Builder

1. Open the Project Properties dialog box.
2. Select Flex Build Path
3. Select the Library Path tab.
4. Click the Add SWC button.
5. In the Add SWC dialog box, click the Browse button.
6. Select your SWC file.

The component now appears in Flex Builder code hints.

7. Add an XML namespace for the component. For example:

```
xmlns:myComps="*"
```

8. Compile your application.

Compile your application by using the Flex mxmhc compiler

- Include the SWC file in the compilation by using the `library-path` option, as the following example shows:

```
mxmhc --strict=true --show-actionscript-warnings=true --use-network=true  
      --library-path+=./SWF9SWC --file-specs EmbedSWF9ButtonSkins.mxml
```

In this example, the SWC file is in the SWF9SWC directory.

Using Flash components in a Flex application

You use components that you create in Flash anywhere in a Flex application that you use any other type of Flex component. Therefore, the components can be children of a container, can dispatch and respond to events, work with effects, and perform just about any other action that a standard Flex component can perform.

Positioning a Flash component in a Flex container

Flex containers position child components by setting the x and y coordinates of the component's upper-left corner within the layout area of container. For components created in Flash, containers uses the registration point of the component to position it. Therefore, you should set the registration point of your Flash component to its upper-left corner. If you have Flash component that requires a different registration point, wrap the component in a new symbol with a registration point in the upper-left corner.

Using SWF 9 files as skins

To use a Flash component as a Flex skin, create your symbol as a subclass of `UIMovieClip`, publish it in a SWC file with other Flash components, and then use the class name as a skin. This procedure is very much like using a programmatic skin, as described in the Using Skins topic of the *Flex 2 Developer's Guide*.

For an example that uses Flash components as skins, see [“Compiling your Flex application” on page 786](#).

Adding custom events

The `UIMovieClip` class supports many of the same events that you use with any Flex component, such as `mouseover`, `move`, `initialize`, and others. You can also add your own custom events to a Flash component for use in your Flex application. To add custom events, create an ActionScript file for your class definition, as the following example shows:

```
package {  
  
    // EventBlueSquare.as  
  
    import mx.flash.UIMovieClip;  
    import flash.events.Event;  
  
    [Event(name="blueSquareSelected", type="flash.events.Event")]  
  
    public class EventBlueSquare extends UIMovieClip  
    {  
        public function EventBlueSquare() {  
            super();  
            addEventListener('mouseDown', dispatchMyEvent);  
        }  
  
        protected function dispatchMyEvent(event:Event):void {  
            dispatchEvent(new Event("blueSquareSelected"));  
        }  
    }  
}
```

After you create this file, you must create a symbol named `EventBlueSquare` in your FLA file, and then use the `Make Flex Component` command to convert that symbol to a Flex component.

You can use your custom event from a Flex application, as the following example shows:

```
<?xml version="1.0"?>
<!-- embedSWF9/EmbedSWF9Event.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:myComps="*">

    <mx:Script>
        <![CDATA[

                public function handleBlueSquareSelected():void {
                    myTA.text='blueSquareSelected event occured.';
                }
            ]]>
    </mx:Script>

    <myComps:EventBlueSquare id="myBS"
        blueSquareSelected="handleBlueSquareSelected();"/>

    <mx:TextArea id="myTA"/>
</mx:Application>
```

For more information on defining custom events, see Chapter 4, “Creating Custom Events,” in *Creating and Extending Flex 2 Components*.

Adding tool tips to Flash components

The `UIMovieClip` class implements the `IToolTipManagerClient` interface, which includes the `toolTip` property. If you set the `toolTip` property in your Flash component, the tool tip appears in your Flex application when the mouse moves over the component.

For example, to add a tool tip to the `EventBlueSquare` component defined in the section “Adding custom events” on page 789, you modify the constructor to set the `toolTip` property, as the following example shows:

```
public function EventBlueSquare() {
    super();
    addEventListener('mouseDown', dispatchMyEvent);
    toolTip = 'blue square component';
}
```

You can also set the tool tip in MXML when you use the component in a Flex application, as the following example shows:

```
<?xml version="1.0"?>
<!-- embedSWF9/EmbedSWF9ToolTip.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:myComps="*">

    <myComps:EventBlueSquare id="myBS"
        tooltip="MXML Tooltip"/>
</mx:Application>
```

For more information on tool tips, see Chapter 23, “Using ToolTips,” in *Flex 2 Developer’s Guide*.

Controlling the size of a Flash component

By default, the measured size of a Flash component at run time matches its actual size. Any run-time changes to the component’s size are recognized by Flex, and Flex updates the layout of your application with the component’s new size.

You should be aware of some important resizing considerations:

- Typically, the Flash `width` and `height` properties determine the size of the component in Flex.
- Shape tweens are only resized on key frames.
- Masked content is reported at its full size, not its visual size.

There are several situations when you might have to add functionality to your Flash component to control its size:

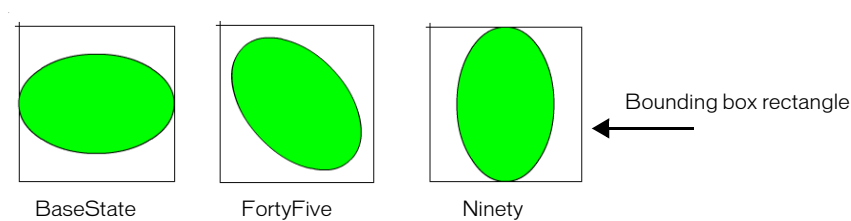
1. The component resizes itself at run time because of an animation, tween, or other reason. You can add functionality to the Flash component to prevent Flex from having to update the component’s size. For more information, see [“Adding a bounding box to a Flash component” on page 792](#).
2. Flex resizes the component at run time and you require the component to perform an action based on its new size. For example, you might have a component that can modify its appearance as it gets larger or smaller. For more information, see [“Overriding the `setActualSize\(\)` method” on page 793](#).

Adding a bounding box to a Flash component

You might add animations or other effects to your Flash component that change its size at run time. By default, Flex sets the size of a Flash component at run time to its actual size. Therefore, when your component changes size, Flex must update the layout of your application; otherwise, the new size of your component might cause it to overlap other components.

One way to control the size of a Flash component is to add a bounding box to the component. The bounding box is a MovieClip instance that Flex uses to size the Flash component at run time. Within the area of the bounding box, the Flash component can perform animations or modify its visual characteristics. But, as long as the area of the bounding box does not change, Flex does not resize the component.

The following figure shows a Flash component named `StatesGreenOval` with three view states:



The rectangle surrounding the green oval is the MovieClip that corresponds to the bounding box. As the green oval rotates, it stays within the area defined by the bounding box, so Flex does not attempt to resize it.

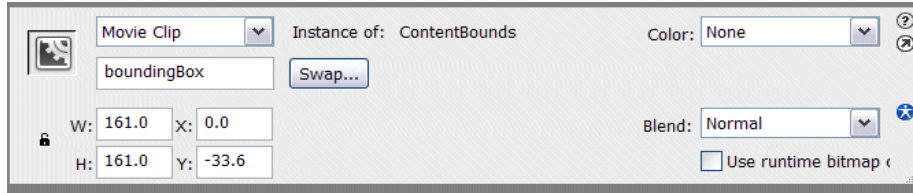
The `UIMovieClip` class defines a property named `boundingBoxName` specifies the bounding box MovieClip for the Flash component. The default value of the `boundingBoxName` property is `boundingBox`. Therefore, you typically specify `boundingBox` as the instance name of the bounding box MovieClip. If you use a different instance name for the MovieClip, ensure that you set the `boundingBoxName` property to that name.

The Make Flex Component command sets the `visible` property to `false` for the MovieClip that represents the bounding box so that it does not appear in your Flex application.

Add a bounding box to a Flash component

1. Add a new layer to the symbol definition of your component.
2. Use the Rectangle tool to draw a bounding box rectangle.
3. Set the size of the rectangle.
4. Convert the rectangle to a symbol of type MovieClip.

5. In the Properties Window for the MovieClip, specify `boundingBox` for the instance name of the MovieClip, as the following figure shows:



Overriding the `setActualSize()` method

If you want your component to be able to recognize when Flex resizes it at run time, you override the component's `setActualSize()` method. Flex calls the component's `setActualSize()` method at run time, passing to it the height and width that Flex has allocated for the component.

In the following example, you define a Flash component named `DotPattern` that lays out a grid of instances of the `YellowDot` MovieClip. The number of rows and columns of `YellowDot` instances in the component is determined by the run-time size of the `DotPattern` component.

```
package {
    import mx.flash.UIMovieClip;

    public dynamic class DotPattern extends UIMovieClip
    {
        private const SIZE:Number = 36; // Size of column/row
        private const GAP:Number = 4;

        public function DotPattern():void
        {
        }

        // Set the default width of the component, in pixels,
        // to be wide enough for 2 columns plus the gap between them.
        override public function get measuredWidth():Number
        {
            return SIZE + GAP + SIZE;
        }

        // Set the default height of the component, in pixels,
        // to be tall enough for 2 columns plus the gap between them.
        override public function get measuredHeight():Number
        {
            return SIZE + GAP + SIZE;
        }

        // Set the number of visible dots at run time based on the size
        // Flex passes to the component at runtime.
        override public function
            setActualSize(newWidth:Number, newHeight:Number):void
        {
            if (newWidth != _width || newHeight != _height)
            {
                _width = newWidth;
                _height = newHeight;

                // Clear out our existing children
                for (var i:int = 0; i < numChildren; i++)
                {
                    removeChildAt(0);
                }

                // Figure out how many rows/columns to create.
                var numRows:int = Math.floor((newHeight) / (SIZE + GAP));
                var numCols:int = Math.floor((newWidth) / (SIZE + GAP));
            }
        }
    }
}
```

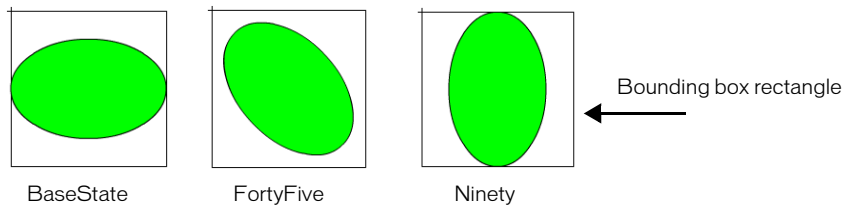

Adding view states

Each view state of a Flash component corresponds to a specific key frame and frame label. For example, to create a component with three view states, you define three key frames and add a frame label to each key frame.

NOTE

Ensure that you add the key frames and frame labels to the symbol definition in Flash, not to the scene.

The following figure shows a Flash component named `StatesGreenOval` with three view states:



NOTE

The green oval includes a bounding box that Flex uses to control the run-time size of the component. For more information, see [“Adding a bounding box to a Flash component” on page 792](#).

In the Flex application, you change the view state of the component by setting its `currentState` property to the frame label, as the following example shows:

```
<?xml version="1.0"?>
<!-- embedSWF9/EmbedSWF9ViewStates.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" height="1500"
  xmlns:myComps="*">

  <mx:Button label="BaseState" click="myOval.currentState='BaseState';"/>
  <mx:Button label="FortyFive" click="myOval.currentState='FortyFive';"/>
  <mx:Button label="Ninety" click="myOval.currentState='Ninety';"/>

  <myComps:StatesGreenOval id="myOval"/>

</mx:Application>
```

Adding transitions

A Flash component can define an animation that plays during a view state change, where the animation corresponds to a Flex transition. For example, if the change of view state alters the color, size, or other visual aspect of the component, the transition animates that change.

To define an animation in a Flash component that corresponds to a Flex transition, you create two key frames and associated frame labels. The first key frame marks the beginning of the animation, and the second key frame marks the end. The frame labels must adhere to the following naming convention:

- Beginning key frame: *currentViewState-destinationViewState:start*
- Ending key frame: *currentViewState-destinationViewState:end*

For example, if the name of the current view state is `BaseState`, and the name of the destination view state is `Ninety`, the frame labels for the associated key frames of the transition must be:

```
BaseState-Ninety:start  
BaseState-Ninety:end
```

When you change the view state of the Flash component from `BaseState` to `Ninety`, Flex automatically looks for frame labels that define the beginning and end of the transition and, if found, plays the associated animation. When you switch back from the view state `Ninety` to `BaseState`, Flex automatically looks for frame labels in the form:

```
Ninety-BaseState:start  
Ninety-BaseState:end
```

If found, Flex automatically plays the animation. If these labels are not found, Flex plays the animation for the `BaseState` to `Ninety` transition in reverse.

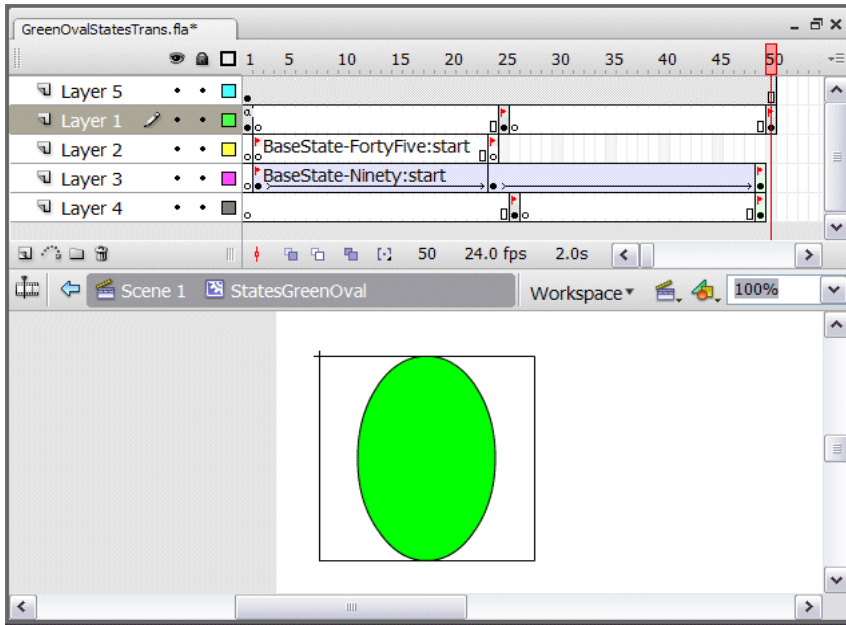
You can use the wildcard symbol, "*", when specifying the frame label, where the wildcard symbol corresponds to any view state. For example, the frame label `BaseState-*:start` specifies a transition from the `BaseState` view state to any other view state. The frame label `*-BaseState:start` specifies a transition from any view state to `BaseState`.

Flex searches for frame labels that match the view state change in the following order:

- *currentViewState-destinationViewState* (no wildcards)
- *currentViewState-destinationViewState* (reversed)
- **-destinationViewState*
- **-destinationViewState* (reversed)
- *currentViewState-**
- *currentViewState-** (reversed)
- **-**

This list shows that Flex searches first for frame labels with no wildcards, both forward and reverse, before checking for frame labels that use wildcards. Then, Flex searches for a frame label that uses a wildcard for the current view state, both forward and reverse, followed by a search for a frame label using a wildcard for the destination view state. Lastly, Flex searches for a frame label using wildcards for both the current and destination view states.

The following image shows a green oval in Flash CS3:



In this example, the layers define the following information:

Layer 1 Specifies three key frames at frames 1, 25, and 50 that define the view states for the green oval. The oval is either at an angle of 0, 45, or 90 degrees. The figure shows the 90 degree rotation.

Layer 2 Defines the following two key frames and frame labels for the transition from the BaseState view state to the FortyFive view state:

BaseState-FortyFive:start

BaseState-FortyFive:end (not shown in the image)

Layer 3 Defines two key frames, frame labels, and a motion tween for the transition from the BaseState view state to the Ninety view state. The motion tween defines the animation that plays when you change view states. The frame labels are:

BaseState-Ninety:start

BaseState-Ninety:end (not shown in the image)

Layer 4 Defines two key frames and frame labels for the transition from the FortyFive view state to the Ninety view state:

FortyFive-Ninety:start

FortyFive-Ninety:end (not shown in the image)

The following Flex application uses the transition when changing view state:

```
<?xml version="1.0"?>
<!-- embedSWF9/EmbedSWF9ViewStatesTrans.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" height="1500"
  xmlns:myComps="*">

  <mx:Button label="BaseState" click="myOval.currentState='BaseState';"/>
  <mx:Button label="FortyFive" click="myOval.currentState='FortyFive';"/>
  <mx:Button label="Ninety" click="myOval.currentState='Ninety';"/>

  <myComps:StatesGreenOval id="myOval" width="161" height="161"/>

</mx:Application>
```

Using tweens in a transition

The previous example uses a motion tween to define the animation used by all the transitions for the component. By default, Flex plays the tween when it loads the Flash component. To prevent Flex from playing the tween when the application loads, add the `stop()` statement to the Actions panel of Frame 1 of Layer 1.

