Using
ACTIONSCRIPT® 3.0 Components

Legal notices

Contents

Chapter 1: Introduction

Adobe® Flash® CS5 Professional is the standard authoring tool for producing high-impact web experiences. Components are the building blocks for the rich Internet applications that provide these experiences. A *component* is a movie clip with parameters that allow you to customize the component either during authoring in Flash or at run time with Adobe® ActionScript® methods, properties, and events. Components are designed to allow developers to reuse and share code, and to encapsulate complex functionality that designers can use and customize without using ActionScript.

Components allow you to easily and quickly build robust applications with a consistent appearance and behavior. This manual describes how to build applications with Adobe ActionScript 3.0 components. The *Adobe® ActionScript® 3.0 Language and Components Reference* describes each component's application programming interface (API).

You can use components created by Adobe®, download components created by other developers, or create your own components.

Intended audience

This manual is for developers who are building Flash applications and want to use components to speed development. You should already be familiar with developing applications in Flash and writing ActionScript.

If you are less experienced with writing ActionScript, you can add components to a document, set their parameters in the Property inspector or Component inspector, and use the Behaviors panel to handle their events. For example, you could attach a Go To Web Page behavior to a Button component that opens a URL in a web browser when the button is clicked without writing any ActionScript code.

If you are a programmer who wants to create more robust applications, you can create components dynamically, use ActionScript to set properties and call methods at run time, and use the event listener model to handle events.

For more information, see "Working with Components" on page 16.

System requirements

Flash components do not have any system requirements beyond the system requirements for Flash.

Any SWF file that uses Flash CS3 or later components must be viewed with Adobe® Flash® Player 9.0.28.0 or later, and must be published for ActionScript 3.0 (you can set this through File > Publish Settings, in the Flash tab).

About the documentation

This document explains the details of using components to develop Flash applications. It assumes that you have general knowledge of Flash and ActionScript 3.0. Specific documentation about Flash and related products is available separately.

This document is available as a PDF file and as online help. To view the online help, start Flash and select Help > Flash Help > Using Adobe ActionScript 3.0 Components.

For information about Flash, see the following documents:

- *Using Flash*
- *ActionScript 3.0 Developer's Guide*
- *Adobe ActionScript 3.0 Reference for the Adobe Flash Platform*

Typographical conventions

The following typographical conventions are used in this manual:

- *Italic font* indicates a value that should be replaced (for example, in a folder path).
- `Code font` indicates ActionScript code, including method and property names.
- *Code font italic* indicates a code item that should be replaced (for example, an ActionScript parameter).
- **Bold font** indicates a value that you enter.

Terms used in this manual

The following terms are used in this manual:

at run time When the code is running in Flash Player.

while authoring While you are working in the Flash authoring environment.

Additional resources

In addition to the content in these manuals, Adobe provides regularly updated articles, design ideas, and examples at the Adobe Developer Center and the Adobe Design Center.

You can find additional component samples at www.adobe.com/go/learn_fl_samples.

Adobe Developer Center

The Adobe Developer Center is your resource for up-to-the-minute information on ActionScript, articles about real-world application development, and information about important emerging issues. View the Developer Center at www.adobe.com/go/flash_devcenter.

Adobe Design Center

Learn the latest in digital design and motion graphics. Browse work by leading artists, discover new design trends, and hone your skills with tutorials, key workflows, and advanced techniques. Check back twice a month for fresh tutorials and articles, and inspirational gallery pieces. View the Design Center at www.adobe.com/go/fl_designcenter.

Chapter 2: About ActionScript 3.0 Components

Adobe® Flash® Professional CS5 components are movie clips with parameters that allow you to modify their appearance and behavior. A component can be a simple user interface control, such as a RadioButton or a CheckBox, or it can contain content, such as a List or DataGrid.

Components allow you to easily and quickly build robust Flash applications with consistent behavior and appearance. Rather than creating custom buttons, combo boxes, and lists, you can use the Flash components that implement these controls. Simply drag them from the Components panel into your application document. You can also easily customize the look and feel of these components to suit your application design.

While you can do all of this without an advanced understanding of ActionScript, you can also use ActionScript 3.0 to modify a component's behavior or implement new behavior. Each component has a unique set of ActionScript methods, properties, and events that make up its *application programming interface* (API). The API allows you to create and manipulate components while the application is running.

The API also allows you to create new, custom components of your own. You can download components built by members of the Flash community on the Adobe Exchange at www.adobe.com/go/flash_exchange.

The ActionScript 3.0 component architecture includes classes on which all components are based, skins and styles that allow you to customize appearance, an event-handling model, focus management, an accessibility interface, and more.

***Note:** Adobe Flash CS5 includes ActionScript 2.0 components as well as ActionScript 3.0 components. You cannot mix these two sets of components. You must use one set or the other for a given application. Flash CS5 presents either ActionScript 2.0 components or ActionScript 3.0 components based on whether you open an ActionScript 2.0 or an ActionScript 3.0 file. When you create a new Flash document, you must specify either Flash File (ActionScript 3.0) or Flash File (ActionScript 2.0). When you open an existing document, Flash examines the Publish Settings to determine which set of components to use. For information about ActionScript 2.0 components, see Using Adobe® ActionScript® 2.0 Components.*

For a complete list of the Flash ActionScript 3.0 components, see "Component types" on page 4.

Benefits of using components

Components enable you to separate the process of designing your application from the process of coding. They allow developers to create functionality that designers can use in applications. Developers can encapsulate frequently used functionality into components and designers can customize the size, location, and behavior of components by changing their parameters. They can also change the appearance of a component by editing its graphical elements, or skins.

Components share core functionality such as styles, skins, and focus management. When you add the first component to an application, this core functionality accounts for approximately 20 kilobytes of the size. When you add other components, that initial memory allocation is shared by the added components, reducing the growth in the size of your application.

This section outlines some of the benefits of the ActionScript 3.0 components.

The power of ActionScript 3.0 provides a powerful, object-oriented programming language that is an important step in the evolution of Flash Player capabilities. The language is designed for building rich Internet applications on a reusable code base. ActionScript 3.0 is based on ECMAScript, the international standardized language for scripting, and is compliant with the ECMAScript (ECMA-262) edition 3 language specification. For a thorough introduction to ActionScript 3.0, see *ActionScript 3.0 Developer's Guide*. For reference information on the language, see the ActionScript 3.0 Reference for the Adobe Flash Platform.

FLA-based User Interface components provide easy access to skins for easy customizing while authoring. These components also provide styles, including skin styles, that allow you to customize aspects of the components appearance and load skins at run time. For more information, see "Customizing the UI Components" on page 95 and the ActionScript 3.0 Reference for the Adobe Flash Platform.

New FVLPlayback component adds FLVPlaybackCaptioning component along with full screen support, improved live preview, skins that allow you to add color and alpha settings, and improved FLV download and layout features.

The Property inspector and Component inspector allow you to change component parameters while authoring in Flash. For more information, see "Working with component files" on page 18 and "Set parameters and properties" on page 20.

New collection dialog box for the ComboBox, List, and TileList components allows you to populate their `dataProvider` property through the user interface. For more information, see "Create a DataProvider" on page 28.

The ActionScript 3.0 event model allows your application to listen for events and invoke event handlers to respond. For more information, see "ActionScript 3.0 event handling model" on page 8 and "Handling events" on page 23.

Manager classes provide an easy way to handle focus and manage styles in an application. For more information, see the ActionScript 3.0 Reference for the Adobe Flash Platform.

The UIComponent base class provides core methods, properties, and events to components that extend it. All of the ActionScript 3.0 user interface components inherit from the UIComponent class. For more information see the UIComponent class in the ActionScript 3.0 Reference for the Adobe Flash Platform.

Use of a SWC in the UI FLA-based components provide ActionScript definitions as an asset inside the component's Timeline to speed compilation.

An easily extendable class hierarchy using ActionScript 3.0 allows you to create unique namespaces, import classes as needed, and subclass easily to extend components.

For more information, see the ActionScript 3.0 Reference for the Adobe Flash Platform.

Note: *Flash CS5 supports both FLA-based and SWC-based components. For more information, see "Component architecture" on page 16.*

Component types

You install the Flash components when you install Flash CS5.

ActionScript 3.0 components include the following user interface (UI) components:

Button	List	TextArea
CheckBox	NumericStepper	TextInput
ColorPicker	RadioButton	TileList

ComboBox	ProgressBar	UILoader
DataGrid	ScrollPane	UIScrollBar
Label	Slider	

In addition to the user interface components, the Flash ActionScript 3.0 components include the following components and supporting classes:

- FLVPlayback component (fl.video.FLVPlayback), which is a SWC-based component.

  The FLVPlayback component lets you readily include a video player in your Flash application to play progressive streaming video over HTTP, from an Adobe® Flash® Video Streaming Service (FVSS), or from Adobe's Macromedia® Flash® Media Server (FMS). For more information, see "Using the FLVPlayback Component" on page 131.

- The FLVPlayback Custom UI components, which are FLA-based and work with both the ActionScript 2.0 and ActionScript 3.0 versions of the FLVPlayback component. For more information, see "Using the FLVPlayback Component" on page 131.

- The FLVPlayback Captioning component, which provides closed captioning for FLVPlayback. See "Using the FLVPlayback Captioning Component" on page 167.

  For a complete list of the ActionScript 3.0 components and their supporting classes, see the ActionScript 3.0 Reference for the Adobe Flash Platform.

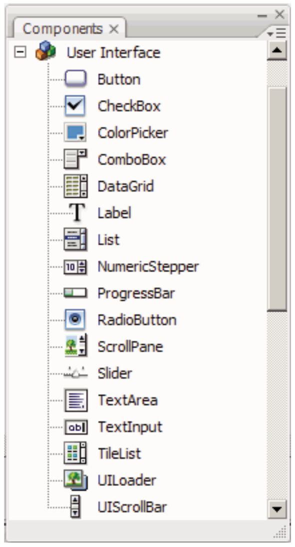
View Flash components:

You can view the Flash ActionScript 3.0 components in the Components panel by following these steps.

1 Start Flash.

2 Create a new Flash file (ActionScript 3.0) or open an existing Flash document in which the Publish Settings specify ActionScript 3.0.

3 Select Window > Components to open the Components panel, if it isn't already open.



Components panel with User Interface components

You can also download additional components from the Adobe Exchange at www.adobe.com/go/flash_exchange. To install components downloaded from the Exchange, download and install the Adobe® Extension Manager at www.adobe.com/go/exchange. Click the Adobe Exchange Home link and look for the Extension Manager link.

Any component can appear in the Components panel in Flash. Follow these steps to install components on either a Windows® or Macintosh® computer.

Install components on a Windows-based or a Macintosh computer:

1 Quit Flash.

2 Place the SWC or FLA file containing the component in the following folder on your hard disk:

- In Windows:

  C:\Program Files\Adobe\Adobe Flash CS5*language*\Configuration\Components

- On the Macintosh:

  Macintosh HD:Applications:Adobe Flash CS5:Configuration:Components

3 Start Flash.

4 Select Window > Components to view the component in the Components panel if it isn't already open.

For more information about component files, see "Working with component files" on page 18

Add to and delete from a document

When you drag a FLA-based component from the Components panel to the Stage, Flash imports an editable movie clip to the library. When you drag a SWC-based component to the Stage, Flash imports a compiled clip to the library. After a component has been imported to the library, you can drag instances of it to the Stage from either the Library panel or the Components panel.

Add components during authoring

You can add a component to a document by dragging it from the Components panel. You can set properties for each instance of a component in the Property inspector or in the Parameters tab in the Component inspector.

1 Select Window > Components.

2 Either double-click the component in the Components panel or drag the component to the Stage.

3 Select the component on the Stage.

4 If the Property inspector is not visible, select Window > Properties > Properties.

5 In the Property inspector, enter an instance name for the component instance.

6 Select Window > Component inspector and select the Parameters tab to specify parameters for the instance.

For more information, see "Set parameters and properties" on page 20.

7 Change the size of the component as desired by editing the values for the width (W:) and height (H:).

For more information on sizing specific component types, see "Customizing the UI Components" on page 95.

8 Select Control > Test Movie or press Control+Enter to compile the document and see the results of your settings.

You can also change the color and text formatting of a component by setting style properties for it or customize its appearance by editing the component's skins. For more information on these topics, see "Customizing the UI Components" on page 95.

If you drag a component to the Stage during authoring, you can refer to the component by using its instance name (for example, `myButton`).

Add components at run time with ActionScript

To add a component to a document at run time with ActionScript, the component must first be in the application's library (Window > Library) when the SWF file is compiled. To add a component to the library, drag the component from the Components panel into the Library panel. For more information on the library, see "The library" on page 21.

You must also import the component's class file to make its API available to your application. Component class files are installed in *packages* that contain one or more classes. To import a component class, use the `import` statement and specify the package name and class name. You would import the Button class, for example, with the following `import` statement:

```
import fl.controls.Button;
```

For information on what package a component is in, see the ActionScript 3.0 Reference for the Adobe Flash Platform. For information about the location of component source files, see "Working with component files" on page 18.

To create an instance of the component, you must invoke the component's ActionScript constructor method. For example, the following statement creates an instance of a Button called `aButton`:

```
var aButton:Button = new Button();
```

The final step is to call the static `addChild()` method to add the component instance to the Stage or application container. For example, the following statement adds the `aButton` instance:

```
addChild(aButton);
```

At this point, you can use the component's API to dynamically specify the component's size and position on the Stage, listen for events, and set properties to modify its behavior. For more information on the API for a particular component, see the ActionScript 3.0 Reference for the Adobe Flash Platform.

For more information on the `addChild()` method, see "Work with the display list" on page 24.

Delete a component

To delete a component instance from the Stage while authoring, simply select it and press the Delete key. This will remove the instance from the Stage but does not remove the component from your application.

To delete a component from your Flash document after you've placed it on the Stage or in the library, you must delete it and its associated assets from the library. It isn't enough to delete the component from the Stage. If you don't remove it from the library, it will be included in your application when you compile it.

1. In the Library panel, select the symbol for the component.
2. Click the Delete button at the bottom of the Library panel, or select Delete from the Library panel menu.

   Repeat these steps to delete any assets associated with the component.

   For information on how to remove a component from its container while your application is running, see "Remove a component from the display list" on page 25.

Find the version of the component

Flash ActionScript 3.0 components have a version property that you can display if you need to provide it to Adobe Technical Support or you need to know what version of the component you are using.

Display the version number for a user interface component:

1 Create a new Flash file (ActionScript 3.0) document.

2 Drag the component to the Stage and give it an instance name. For example, drag a ComboBox to the Stage and call it **aCb**.

3 Press the **F9** key or select Window > Actions to open the Actions panel.

4 Click Frame 1 on the main Timeline and add the following code to the Actions panel:

```
trace(aCb.version);
```

The version number, similar to the one in the following illustration, should appear in the Output panel.

For the FLVPlayback and FLVPlaybackCaptioning components, you must refer to the class name rather than the instance name because the version number is stored in a class constant.

Display the version number for the FLVPlayback and FLVPlaybackCaptioning components:

1 Create a new Flash file (ActionScript 3.0) document.

2 Drag the FLVPlayback and FLVPlaybackCaptioning components into the Library panel.

3 Press the **F9** key or select Window > Actions to open the Actions panel.

4 Click Frame 1 on the main Timeline and add the following code to the Actions panel.

```
import fl.video.*;
trace("FLVPlayback.VERSION: " + FLVPlayback.VERSION);
trace("FLVPLaybackCaptioning.VERSION: " + FLVPlaybackCaptioning.VERSION);
```

The version numbers will appear in the Output panel.

ActionScript 3.0 event handling model

ActionScript 3.0 introduces a single event handling model that replaces the different event handling mechanisms that existed in previous versions of ActionScript. The new event model is based on the Document Object Model (DOM) Level 3 Events Specification.

For developers with experience using the ActionScript 2.0 `addListener()` method, it may be helpful to point out the differences between the ActionScript 2.0 event listener model and the ActionScript 3.0 event model. The following list describes a few of the major differences between the two event models:

- To add event listeners in ActionScript 2.0, you use `addListener()` in some cases and `addEventListener()` in others, whereas in ActionScript 3.0 you use `addEventListener()` in all cases.
- There is no event flow in ActionScript 2.0, which means that the `addListener()` method can be called only on the object that broadcasts the event, whereas in ActionScript 3.0 the `addEventListener()` method can be called on any object that is part of the event flow.
- In ActionScript 2.0, event listeners can be either functions, methods, or objects, whereas in ActionScript 3.0, only functions or methods can be event listeners.

- The `on(`*event*`)` syntax is no longer supported in ActionScript 3.0, so you cannot attach ActionScript event code to a movie clip. You can only use `addEventListener()` to add an event listener.

  The following example, which listens for a `MouseEvent.CLICK` event on a Button component called `aButton`, illustrates the basic ActionScript 3.0 event handling model:

  ```
  aButton.addEventListener(MouseEvent.CLICK, clickHandler);
  function clickHandler(event:MouseEvent):void {
  trace("clickHandler detected an event of type: " + event.type);
  trace("the event occurred on: " + event.target.name);
  }
  ```

  For more information on ActionScript 3.0 event handling, see *Programming ActionScript 3.0*. For more information on ActionScript 3.0 event handling for components, see "Handling events" on page 23.

A simple application

This section takes you through the steps to create a simple ActionScript 3.0 application using Flash components and the Flash authoring tool. The example is provided both as a FLA file with the ActionScript code included on the Timeline and also as an external ActionScript class file with a FLA file that contains only the components in the library. In general, you will want to develop larger application using external class files so that you can share code between classes and applications and to make your applications easier to maintain. For more information on programming with ActionScript 3.0, see *Programming ActionScript 3.0*.

Design of the application

Our first example of an ActionScript component application is a variation of the standard "Hello World" application, so its design is fairly simple:

- The application will be called Greetings.
- It uses a TextArea to display a greeting that is initially Hello World.
- It uses a ColorPicker that allows you to change the color of the text.
- It uses three RadioButtons that allow you to set the size of the text to small, larger or largest.
- It uses a ComboBox that allows you to select a different greeting from a drop-down list.
- The application uses components from the Components panel and also creates application elements through ActionScript code.

  With that definition in place, you can start building the application.

Create the Greetings application

The following steps create the Greetings application using the Flash authoring tool to create a FLA file, place components on the Stage, and add ActionScript code to the Timeline.

Create the Greetings application in a FLA file:

1. Select File > New.
2. In the New Document dialog box, select Flash File (ActionScript 3.0), and click OK.

   A new Flash window opens.
3. Select File > Save, name the Flash file **Greetings.fla**, and click the Save button.

4 In the Flash Components panel, select a TextArea component and drag it to the Stage.

5 In the Properties window, with the TextArea selected on the Stage, type **aTa** for the instance name, and enter the following information:

- Enter **230** for the W value (width).
- Enter **44** for the H value (height).
- Enter **165** for the X value (horizontal position).
- Enter **57** for the Y value (vertical position).
- Enter **Hello World!** for the text parameter, on the Parameters tab.

6 Drag a ColorPicker component to the Stage, place it to the left of the TextArea and give it an instance name of **txtCp.** Enter the following information in the Property inspector:

- Enter **96** for the X value.
- Enter **72** for the Y value.

7 Drag three RadioButton components to the Stage, one at a time and give them instance names of **smallRb**, **largerRb**, and **largestRb**. Enter the following information for them in the Property inspector:

- Enter **100** for the W value and **22** for the H value for each of them.
- Enter **155** for the X value.
- Enter **120** for the Y value for smallRb, **148** for largerRb, and **175** for largestRb.
- Enter **fontRbGrp** for the groupName parameter for each of them.
- Enter labels for them on the Parameters tab of **Small**, **Larger**, **Largest**.

8 Drag a ComboBox to the Stage and give it an instance name of **msgCb**. Enter the following information for it in the Property inspector:

- Enter **130** for the W value.
- Enter **265** for the X value.
- Enter **120** for the Y value.
- On the Parameters tab, enter **Greetings** for the prompt parameter.
- Double-click the text field for the dataProvider parameter to open the Values dialog box.
- Click the plus sign and replace the label value with **Hello World!**
- Repeat the preceding step to add the label values **Have a nice day!** and **Top of the Morning!**
- Click OK to close the Values dialog box.

9 Save the file.

10 If it is not already open, open the Actions panel by hitting **F9** or selecting Actions from the Window menu. Click Frame 1 on the main Timeline and enter the following code in the Actions panel:

```
import flash.events.Event;
import fl.events.ComponentEvent;
import fl.events.ColorPickerEvent;
import fl.controls.RadioButtonGroup;

var rbGrp:RadioButtonGroup = RadioButtonGroup.getGroup("fontRbGrp");
rbGrp.addEventListener(MouseEvent.CLICK, rbHandler);
txtCp.addEventListener(ColorPickerEvent.CHANGE,cpHandler);
msgCb.addEventListener(Event.CHANGE, cbHandler);
```

The first three lines import the event classes that the application uses. An event occurs when a user interacts with one of the components. The next five lines register event handlers for the events that the application wants to listen for. A `click` event occurs for a RadioButton when a user clicks on it. A `change` event occurs when a user selects a different color in the ColorPicker. A `change` event occurs on the ComboBox when a user chooses a different greeting from the drop-down list.

The fourth line imports the RadioButtonGroup class so that the application can assign an event listener to the group of RadioButtons, rather than assigning the listener to each button individually.

11 Add the following line of code to the Actions panel to create the `tf` TextFormat object, which the application uses to change the `size` and `color` style properties of the text in the TextArea.

```
var tf:TextFormat = new TextFormat();
```

12 Add the following code to create the `rbHandler` event handling function. This function handles a `click` event when a user clicks on one of the RadioButton components.

```
function rbHandler(event:MouseEvent):void {
    switch(event.target.selection.name) {
        case "smallRb":
            tf.size = 14;
            break;
        case "largerRb":
            tf.size = 18;
            break;
        case "largestRb":
            tf.size = 24;
            break;
        }
        aTa.setStyle("textFormat", tf);
}
```

This function uses a `switch` statement to examine the `target` property of the `event` object to determine which RadioButton triggered the event. The `currentTarget` property contains the name of the object that triggered the event. Depending on which RadioButton the user clicked, the application changes the size of the text in the TextArea to 14, 18, or 24 points.

13 Add the following code to implement the `cpHandler()` function, which handles a change to the value in the ColorPicker:

```
function cpHandler(event:ColorPickerEvent):void {
    tf.color = event.target.selectedColor;
    aTa.setStyle("textFormat", tf);
}
```

This function sets the `color` property of the `tf` TextFormat object to the color selected in the ColorPicker and then calls `setStyle()` to apply it to the text in the `aTa` TextArea instance.

14 Add the following code to implement the `cbHandler()` function, which handles a change to the selection in the ComboBox:

```
function cbHandler(event:Event):void {
    aTa.text = event.target.selectedItem.label;
}
```

This function simply replaces the text in the TextArea with the selected text in the ComboBox, `event.target.selectedItem.label`.

15 Select Control > Test Movie or press Control+Enter to compile the code and test the Greetings application.

The following section shows you how to build the same application with an external ActionScript class and a FLA file that has only the required components in the library.

Create the Greetings2 application with an external class file:

1 Select File > New.

2 In the New Document dialog box, select Flash File (ActionScript 3.0), and click OK.

   A new Flash window opens.

3 Select File > Save, name the Flash file **Greetings2.fla**, and click the Save button.

4 Drag each of the following components from the Components panel to the library:

   - ColorPicker
   - ComboBox
   - RadioButton
   - TextArea

   The compiled SWF file will use each of these assets, so you need to add them to the library. Drag the components to the bottom of the Library panel. As you add these components to the library, other assets (such as List, TextInput, and UIScrollBox) are added automatically.

5 In the Properties window, for the Document Class, type **Greetings2**.

   If Flash displays a warning, saying that "a definition for the document class could not be found," ignore it. You will define the Greetings2 class in the following steps. This class defines the main functionality for the application.

6 Save the Greetings2.fla file.

7 Select File > New.

8 In the New Document dialog box, select ActionScript File, and click OK.

   A new script window opens.

9 Add the following code into the script window:

```
package {
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import flash.text.TextFormat;
    import fl.events.ComponentEvent;
    import fl.events.ColorPickerEvent;
    import fl.controls.ColorPicker;
    import fl.controls.ComboBox;
    import fl.controls.RadioButtonGroup;
    import fl.controls.RadioButton;
    import fl.controls.TextArea;
    public class Greetings2 extends Sprite {
        private var aTa:TextArea;
        private var msgCb:ComboBox;
        private var smallRb:RadioButton;
        private var largerRb:RadioButton;
        private var largestRb:RadioButton;
        private var rbGrp:RadioButtonGroup;
        private var txtCp:ColorPicker;
        private var tf:TextFormat = new TextFormat();
        public function Greetings2() {
```

The script defines an ActionScript 3.0 class, named Greetings2. The script does the following:

- It imports classes that we will use in the file. Normally you would add these `import` statements as you reference different classes in the code, but for the sake of brevity, this example imports them all in this one step.
- It declares variables that represent the different types of component objects that we will add to the code. Another variable creates the `tf` TextFormat object.
- It defines a constructor function, `Greetings2()`, for the class. We will add lines to this function, and add other methods to the class in the following steps.

10 Select File > Save, name the file **Greetings2.as**, and click the Save button.

11 Add the following lines of code to the `Greeting2()` function:

```
    createUI();
    setUpHandlers();
}
```

The function should now look like the following:

```
public function Greetings2() {
    createUI();
    setUpHandlers();
}
```

12 Add the following lines of code after the closing brace of the `Greeting2()` method:

```
private function createUI() {
    bldTxtArea();
    bldColorPicker();
    bldComboBox();
    bldRadioButtons();
}
private function bldTxtArea() {
    aTa = new TextArea();
    aTa.setSize(230, 44);
    aTa.text = "Hello World!";
    aTa.move(165, 57);
    addChild(aTa);
}
private function bldColorPicker() {
    txtCp = new ColorPicker();
    txtCp.move(96, 72);
    addChild(txtCp);
}
private function bldComboBox() {
    msgCb = new ComboBox();
    msgCb.width = 130;
    msgCb.move(265, 120);
    msgCb.prompt = "Greetings";
    msgCb.addItem({data:"Hello.", label:"English"});
    msgCb.addItem({data:"Bonjour.", label:"Français"});
    msgCb.addItem({data:"¡Hola!", label:"Español"});
    addChild(msgCb);
}
private function bldRadioButtons() {
    rbGrp = new RadioButtonGroup("fontRbGrp");
    smallRb = new RadioButton();
    smallRb.setSize(100, 22);
```

```
        smallRb.move(155, 120);
        smallRb.group = rbGrp; //"fontRbGrp";
        smallRb.label = "Small";
        smallRb.name = "smallRb";
        addChild(smallRb);
        largerRb = new RadioButton();
        largerRb.setSize(100, 22);
        largerRb.move(155, 148);
        largerRb.group = rbGrp;
        largerRb.label = "Larger";
        largerRb.name = "largerRb";
        addChild(largerRb);
        largestRb = new RadioButton();
        largestRb.setSize(100, 22);
        largestRb.move(155, 175);
        largestRb.group = rbGrp;
        largestRb.label = "Largest";
        largestRb.name = "largestRb";
        addChild(largestRb);
    }
```

These lines do the following:

- Instantiate the components used in the application.
- Set each component's size, position, and properties.
- Add each component to the Stage, using the `addChild()` method.

13 After the closing brace of the `bldRadioButtons()` method, add the following code for the `setUpHandlers()` method:

```
private function setUpHandlers():void {
    rbGrp.addEventListener(MouseEvent.CLICK, rbHandler);
    txtCp.addEventListener(ColorPickerEvent.CHANGE,cpHandler);
    msgCb.addEventListener(Event.CHANGE, cbHandler);
}
private function rbHandler(event:MouseEvent):void {
    switch(event.target.selection.name) {
        case "smallRb":
            tf.size = 14;
            break;
        case "largerRb":
            tf.size = 18;
            break;
        case "largestRb":
            tf.size = 24;
            break;
    }
    aTa.setStyle("textFormat", tf);
}
private function cpHandler(event:ColorPickerEvent):void {
    tf.color = event.target.selectedColor;
    aTa.setStyle("textFormat", tf);
}
private function cbHandler(event:Event):void {
    aTa.text = event.target.selectedItem.data;
}
}
}
```

These functions define event listeners for the components.

14 Select File > Save to save the file.

15 Select Control > Test Movie or press Control+Enter to compile the code and test the Greetings2 application.

Develop and run subsequent examples

Having developed and run the Greetings application, you should have the basic knowledge you need to run the other code examples presented in this book. The relevant ActionScript 3.0 code in each example will be highlighted and discussed and you should be able to cut and paste each of the examples in this book into a FLA file, compile and run it.

Chapter 3: Working with Components

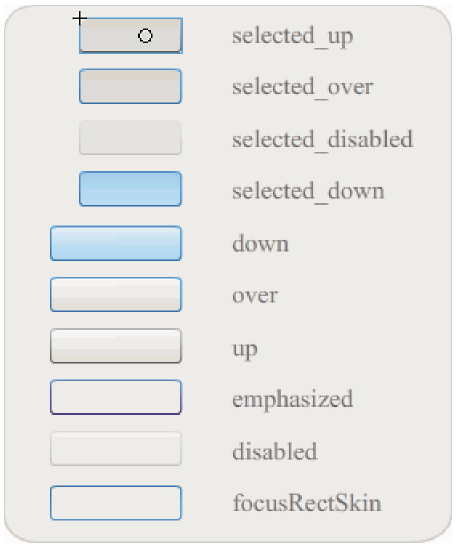
Component architecture

Adobe® ActionScript® 3.0 components are supported by Adobe® Flash Player version 9.0.28.0 and later. These components are not compatible with components built prior to Flash CS4. For information on using Adobe® ActionScript® 2.0 components, see *Using Adobe® ActionScript® 2.0 Components* and the *Adobe® ActionScript® 2.0 Components Language Reference*.

The Adobe ActionScript 3.0 User Interface (UI) components are implemented as FLA-based components but Flash CS5 supports both SWC and FLA-based components. The FLVPlayback and FLVPlaybackCaptioning components are SWC-based components, for example. You can place either type of component in the Components folder to have it appear in the Components panel. These two types of components are built differently so they are described separately here.

ActionScript 3.0 FLA-based components

The ActionScript 3.0 User Interface components are FLA-based (.fla) files with built-in skins that you can access for editing by double-clicking the component on the Stage. The component's skins and other assets are placed on Frame 2 of the Timeline. When you double-click the component, Flash automatically jumps to Frame 2 and opens a palette of the component's skins. The following illustration shows the palette with skins that display for the Button component.



Skins for the Button component

For more information on component skins and customizing components, see "Customizing the UI Components" on page 95 and "Customize the FLVPlayback component" on page 149.

To speed up compilation for your applications and to avoid conflicts with your ActionScript 3.0 settings, the Flash CS5 FLA-based UI components also contain a SWC that contains the component's already compiled ActionScript code. The ComponentShim SWC is placed on Stage on Frame 2 in every User Interface component to make available the precompiled definitions. To be available for ActionScript, a component must either be on Stage or be in the library with the Export In First Frame option selected in its Linkage properties. To create a component using ActionScript, you also must import the class with an `import` statement to access it. For information on the `import` statement, see the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

SWC-based Components

SWC-based components have a FLA file and an ActionScript class file, too, but they have been compiled and exported as a SWC. A SWC file is a package of precompiled Flash symbols and ActionScript code that allows you to avoid recompiling symbols and code that will not change.

The FLVPlayback and FLVPlaybackCaptioning components are SWC-based components. They have external rather than built-in skins. The FLVPlayback component has a default skin that you can change by selecting one from a collection of predesigned skins, by customizing controls from the UI controls in the Components panel (BackButton, BufferingBar, and so on), or by creating a custom skin. For more information, see "Customize the FLVPlayback component" on page 149.

In Flash, you can convert a movie clip to a compiled clip as follows:

Compile a movie clip

- Right-click (Windows) or Control-click (Macintosh) the movie clip in the Library panel, and then select Convert To Compiled Clip.

  The compiled clip behaves just like the movie clip from which it was compiled, but compiled clips appear and publish much faster than ordinary movie clips. Compiled clips can't be edited, but their properties can appear in the Property inspector and the Component inspector.

  SWC components contain a compiled clip, the component's pre-compiled ActionScript definitions, and other files that describe the component. If you create your own component, you can export it as a SWC file to distribute it.

Export a SWC file

- Select the movie clip in the Library panel and right-click (Windows) or Control-click (Macintosh), and then select Export SWC File.

  ***Note:** The format of a Flash CS4 or later SWC file is compatible with the Flex SWC format so that SWC files can be exchanged between the two products, but not necessarily without modifications.*

For information on creating SWC-based components, see www.adobe.com/go/learn_fl_creating_components.

The ActionScript 3.0 Components API

Each ActionScript 3.0 component is built on an ActionScript 3.0 class that is located in a package folder and has a name of the format fl.*packagename.classname*. The Button component, for example, is an instance of the Button class and has a package name of `fl.controls.Button`. You must reference the package name when you import a component class in your application. You would import the Button class with the following statement:

```
import fl.controls.Button;
```

For more information about the location of component class files, see "Working with component files" on page 18.

A component's class defines the methods, properties, events, and styles that enable you to interact with it in your application. The ActionScript 3.0 UI components are subclasses of the Sprite and UIComponent classes and inherit properties, methods, and events from them. The Sprite class is the basic display list building block and is similar to a MovieClip but does not have a Timeline. The UIComponent class is the base class for all visual components, both interactive and non-interactive. The inheritance path of each component, as well as its properties, methods, events, and styles are described in the *Adobe ActionScript 3.0 Reference for the Adobe Flash Platform*.

All ActionScript 3.0 components use the ActionScript 3.0 event handling model. For more information on event handling, see "Handling events" on page 23 and *Programming ActionScript 3.0*.

Working with component files

This section explains where component files are stored, where to find the ActionScript source files, and how to add and remove components from the Components panel.

Where component files are stored

Flash components are stored in the application-level Configuration folder.

Note: *For information about these folders, see "Configuration folders installed with Flash" in Using Flash.*

Components are installed in the following locations:

- Windows:

  C:\Program Files\Adobe\Adobe Flash CS5*language*\Configuration\Components

- Mac OS X:

  Macintosh HD:Applications:Adobe Flash CS5:Configuration:Components

  Within the Components folder, the User Interface (UI) components are in the User Interface.fla file and the FLVPlayback (FLVPlaybackAS3.swc) and FLVPlaybackCaptioning components are in the Video folder.

You can also store components in the following user-based locations:

- Windows:

  C:\Documents and Settings*username*\Local Settings\Application Data\Adobe\Adobe Flash CS5\en\Configuration\Components

- Windows Vista:

  C:\Users\username\Local Settings\Application Data\Adobe\Adobe Flash CS5\en\Configuration\Components

- Windows 7:

  C:\Users\username\AppData\Local\Adobe\Flash CS5\en\Configuration\Components

  Note: *In Windows, the Application Data folder (Windows 2000, XP, Vista) and AppData folder (Windows7) is hidden by default. To show hidden folders and files, select My Computer to open Windows Explorer, select Tools > Folder Options and then select the View tab. Under the View tab, select the Show hidden files and folders radio button.*

- Mac OS X:

  Macintosh HD:Users:<username>:Library:Application Support:Adobe:Flash CS5:Configuration:Components

Where component source files are stored

The ActionScript (.as) class files (or *source files*) for components are installed in the following application folders for Windows 2000, XP, Vista, and 7:

User Interface components C:\Program Files\Adobe\Adobe Flash CS5\en\Configuration\Component Source\ActionScript 3.0\User Interface\fl

FLVPlayback C:\Program Files\Adobe\Adobe Flash CS5\en\Configuration\Component Source\ActionScript 3.0\FLVPlayback\fl\video

FLVPlaybackCaptioning C:\Program Files\Adobe\Adobe Flash CS5\en\Configuration\Component Source\ActionScript 3.0\FLVPlaybackCaptioning\fl\video

For Mac OS X, the component source files are located here:

User Interface components Macintosh HD:Applications:Adobe Flash CS5:Configuration:Component Source:ActionScript 3.0:User Interface:fl

FLVPlayback Macintosh HD:Applications:Adobe Flash CS5:Configuration:Component Source:ActionScript 3.0:FLVPlayback:fl:video

FLVPlaybackCaptioning Macintosh HD:Applications:Adobe Flash CS5:Configuration:Component Source:ActionScript 3.0:FLVPlaybackCaptioning:fl:video

Component source files and Classpath

Because the ActionScript 3.0 components have their code compiled in, you should not specify the location of the ActionScript class files in your Classpath variable. If you do include their location in the Classpath, it will increase the time required to compile your applications. However, if Flash finds component class files in your Classpath setting, the class file will always take precedence over the component's compiled-in code.

One time that you might wish to add the location of the component source files to your Classpath setting is when you're debugging an application with components. For more information see "Debug component applications" on page 20.

Modify the component files

If you update, add, or remove SWC-based components or add new FLA-based components to Flash, you must reload them to the Components panel to make them available. You can reload the components either by restarting Flash or by selecting Reload from the Components panel menu. This will cause Flash to pick up any components that you've added to the Components folder.

Reload components in the Components panel while Flash is running:

- Select Reload from the Components panel menu.

Remove a component from the Components panel:

- Remove the FLA, SWC, or MXP file from the Components folder and either restart Flash or select Reload from the Components panel menu. An MXP file is a component file that has been downloaded from the Adobe Exchange.

  You can remove and replace SWC-based components while Flash is running, and reloading will reflect the changes, but if you change or delete FLA-based components, the changes are not reflected until you terminate and restart Flash. You can, however, add FLA-based components and load them with the Reload command.

  Adobe recommends that you first make a copy of any Flash component file (.fla or .as) that you are going to alter. Then you can restore it, if necessary.

Debug component applications

The ActionScript 3.0 components contain all their source code to reduce compilation time when you compile your application. The Flash debugger, however, cannot inspect code inside compiled clips. Therefore, if you want to debug your application down into the components' source code, you must add the component source files to your Classpath setting.

The location of the component package folders is relative to the location of the source files for the component type. To reference all of the ActionScript 3.0 source files for all UI components, add the following location to your Classpath for the User Interface packages:

- $(AppConfig)/Component Source/ActionScript 3.0/User Interface

Note: *This will override the compiled-in code for all UI components and increase compilation time for your application. If you have changed a component's source file for any reason, that component might exhibit different behavior as a result.*

To set the Classpath, select Preferences from the Edit menu and then select ActionScript from the Category list and click the ActionScript 3.0 Settings button. To add a new entry, click the plus above the window that displays the current settings.

The `$(AppConfig)` variable refers to the Flash CS5 Configuration folder in the location where you installed Flash CS5. Typically, the path looks like this:

- Windows: C:\Program Files\Adobe\Adobe Flash CS5\language\Configuration\
- Mac OS X: Macintosh HD:Applications:Adobe Flash CS5:Configuration

Note: *If you must change a component source file, Adobe strongly recommends that you copy the original source file to a different location and add that location to your Classpath.*

For more information about the location of component source files, see "Where component source files are stored" on page 19.

Set parameters and properties

Each component has parameters that you can set to change its appearance and behavior. A parameter is a property of the component's class and appears in the Property inspector and the Component inspector. The most commonly used properties appear as authoring parameters; others you must set with ActionScript. All parameters that can be set during authoring can also be set with ActionScript. Setting a parameter with ActionScript overrides any value set during authoring.

Most ActionScript 3.0 User Interface components inherit properties and methods from the UIComponent class as well as from a base class. For example, the Button and CheckBox classes inherit properties from both the UIComponent class and the BaseButton class. A component's inherited properties, as well as its own class properties, are available for you to access. For example, the ProgressBar component inherits the `ProgressBar.enabled` property from UIComponent but also has its own `ProgressBar.percentComplete` property. You can access both of these properties to interact with an instance of the ProgressBar component. For more information on a component's properties, see its class entry in the ActionScript 3.0 Reference for the Adobe Flash Platform.

You can set parameters for a component instance using either the Property inspector or the Component inspector.

Enter an instance name for a component in the Property inspector:

1 Select Window > Properties > Properties.

2 Select an instance of a component on the Stage.

3 Enter a name for the component instance in the box that says <Instance Name>, located beneath the drop-down list that says Movie Clip. Or click the Parameters tab and enter the name in the box below the word *Component*. Enter values for any parameters that you want to set.

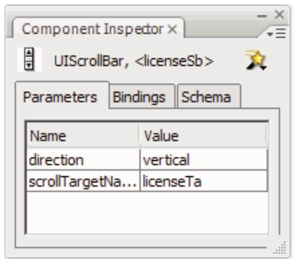
   It's a good idea to add a suffix to the instance name to indicate what kind of component it is; this makes it easier to read your ActionScript code. For example, the instance name **licenseSb** identifies that a component is a scroll bar that scrolls a license agreement in the **licenseTa** text area.

Enter parameters for a component instance in the Component inspector:

1 Select Window > Component Inspector.

2 Select an instance of a component on the Stage.

3 Click the Parameters tab and enter values for any of the listed parameters.



Name	Value
direction	vertical
scrollTargetNa...	licenseTa

Component parameters in the Component inspector

Set component properties in ActionScript

In ActionScript, you use a dot (.) operator (dot syntax) to access properties or methods that belong to an object or instance on the Stage. A dot syntax expression begins with the name of the instance, followed by a dot, and it ends with the element you want to specify. For example, the following ActionScript code sets the `width` property of the CheckBox instance `aCh` to make it 50 pixels wide:

```
aCh.width = 50;
```

The following `if` statement checks to see if the user has selected the check box:

```
if (aCh.selected == true) {
    displayImg(redCar);
}
```

The library

When you first add a component to a document, Flash imports it as a movie clip into the Library panel. You can also drag a component from the Components panel directly to the Library panel and then add an instance of it to the Stage. In any case, you must add a component to the library before you can access its class elements.

If you add a component to the library and create an instance of it using ActionScript, you must first import its class with the `import` statement. In the `import` statement, you must specify both the component's package name and its class name. For example, the following statement imports the Button class:

```
import fl.controls.Button;
```

When you place a component in the library, Flash also imports a folder of its assets, which contain the skins for its different states. A component's *skins* comprise the collection of symbols that make up its graphical display in the application. A single skin is the graphical representation, or movie clip, that indicates a particular state for the component.

The contents of the Component Assets folder allow you to change the component's skins if you wish to do that. For more information, see "Customizing the UI Components" on page 95.

Once a component is in the library, you can add more instances of it to your document by dragging its icon to the Stage from either the Components panel or the Library panel.

Sizing components

Use the Free Transform tool or the `setSize()` method to resize component instances. You can call the `setSize()` method from any component instance (see `UIComponent.setSize()`) to resize it. The following code resizes an instance of the List component to 200 pixels wide and 300 pixels high:

```
aList.setSize(200, 300);
```

A component does not resize automatically to fit its label. If a component instance that has been added to a document is not large enough to display its label, the label text is clipped. You must resize the component to fit its label.

For more information about sizing components, see their individual entries in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

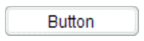
Live Preview

The Live Preview feature, enabled by default, lets you view components on the Stage as they will appear in the published Flash content; the components appear at their approximate size.

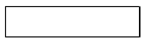
To turn Live Preview on or off:

- Select Control > Enable Live Preview. A check mark next to the option indicates that it is enabled.

The live preview reflects different parameters for different components. For information about which component parameters are reflected in the live preview, see each component entry in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.



A Button component with Live Preview enabled



A Button component with Live Preview disabled

Components in Live Preview are not functional. To test functionality, you must use the Control > Test Movie command.

Handling events

Every component broadcasts events when a user interacts with it. When a user clicks a Button, for example, it dispatches a `MouseEvent.CLICK` event and when a user selects an item in a List, the List dispatches an Event.`CHANGE` event. An event can also occur when something significant happens to a component such as when content finishes loading for a UILoader instance, generating an `Event.COMPLETE` event. To handle an event, you write ActionScript code that executes when the event occurs.

A component's events include the events of any class from which the component inherits. This means that all ActionScript 3.0 User Interface components inherit events from the UIComponent class because it is the base class for the ActionScript 3.0 User Interface components. To see the list of events a component broadcasts, see the Events section of the component's class entry in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

For a complete explanation of event handling in ActionScript 3.0, see the *ActionScript 3.0 Developer's Guide*.

About event listeners

The following key points apply to handling events for ActionScript 3.0 components:

- All events are broadcast by an instance of a component class. The component instance is the *broadcaster*.
- You register an event *listener* by calling the `addEventListener()` method for the component instance. For example, the following line of code adds a listener for the `MouseEvent.CLICK` event to the Button instance `aButton`:

  ```
  aButton.addEventListener(MouseEvent.CLICK, clickHandler);
  ```

  The second parameter of the `addEventListener()` method registers the name of the function, `clickHandler`, to be called when the event occurs. This function is also referred to as a *callbackfunction*.
- You can register multiple listeners to one component instance.

  ```
  aButton.addEventListener(MouseEvent.CLICK, clickHandler1);
  aButton.addEventListener(MouseEvent.CLICK, clickHandler2);
  ```
- You can register one listener to multiple component instances.

  ```
  aButton.addEventListener(MouseEvent.CLICK, clickHandler1);
  bButton.addEventListener(MouseEvent.CLICK, clickHandler1);
  ```
- The event handler function is passed to an event object that contains information about the event type and the instance that broadcast the event. For more information, see "About the event object" on page 23.
- The listener remains active until the application terminates or you explicitly remove it using the `removeEventListener()` method. For example, the following line of code removes the listener for the `MouseEvent.CLICK` event on `aButton`:

  ```
  aButton.removeEventListener(MouseEvent.CLICK, clickHandler);
  ```

About the event object

The event object inherits from the Event object class and has properties that contain information about the event that occurred, including the `target` and `type` properties, which provide essential information about the event:

Property	Description
`type`	A string indicating the type of the event.
`target`	A reference to the component instance broadcasting the event.

When an event has additional properties, they are listed in the event's class description in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

The event object is automatically generated and passed to the event handler function when an event occurs.

You can use the event object inside the function to access the name of the event that was broadcast or the instance name of the component that broadcast the event. From the instance name, you can access other component properties. For example, the following code uses the `target` property of the `evtObj` event object to access the `label` property of `aButton` and display it in the Output panel:

```
import fl.controls.Button;
import flash.events.MouseEvent;

var aButton:Button = new Button();
aButton.label = "Submit";
addChild(aButton);
aButton.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(evtObj:MouseEvent){
trace("The " + evtObj.target.label + " button was clicked");
}
```

Work with the display list

All ActionScript 3.0 components inherit from the DisplayObject class and, therefore, have access to its methods and properties to interact with the display list. The *display list* is the hierarchy of displayed objects and visual elements in an application. This hierarchy includes the following elements:

- The Stage, which is the top-level container
- Display objects, which include shapes, MovieClips, and text fields, among others
- Display object containers, which are special types of display objects that can contain child display objects.

The order of objects in the display list determines their depth in the parent container. An object's depth refers to its position from top to bottom or front to back on the Stage or in its display container. The order of depth is apparent when objects overlap but it exists even when they do not. Every object in the display list has a corresponding depth on the Stage. If you want to change an object's depth by placing it in front of or moving it behind other objects, you need to change its position in the display list. The default order of objects in the display list is the order in which they are placed on the Stage. Position 0 in the display list is the object at the bottom of the depth order.

Add a component to the display list

You can add an object to a DisplayObjectContainer object by calling the container's `addChild()` or `addChildAt()` method. In the case of the Stage, you can also add an object to its display list during authoring by creating it, or in the case of components, by dragging it to the Stage from the Components panel. To add an object to a container with ActionScript, first create an instance of it by invoking its constructor with the `new` operator and then call the `addChild()` or `addChildAt()` method to place it on the Stage and in the display list. The `addChild()` method places the object at the next position in the display list, while `addChildAt()` specifies the position at which to add the object. If you specify a position that is already occupied, the object at that position, and those at higher positions, move up by 1. The `numChildren` property of a DisplayObjectContainer object contains the number of display objects that it contains. You can retrieve an object from the display list by calling the `getChildAt()` method and specifying the position, or if you know the name of the object, by calling the `getChildByName()` method.

Note: *When you add a component with ActionScript, you must assign a name to it's name property if you want to access it by name in the display list.*

The following example displays the names and positions of three components in the display list. First, drag a NumericStepper, a Button, and a ComboBox to the Stage so that they overlap each other and give them instance names of **aNs**, **aButton**, and **aCb**. Then add the following code to the Actions panel on Frame 1 of the Timeline:

```
var i:int = 0;
while(i < numChildren) {
    trace(getChildAt(i).name + " is at position: " + i++);
}
```

You should see the following lines in the Output panel:

```
aNs is at position: 0
aButton is at position: 1
aCb is at position: 2
```

Move a component in the display list

You can change the position of an object in the display list, and its display depth, by calling the `addChildAt()` method and supplying the name of an object and the position where you want to place it as the method's parameters. For example, add the following code to the preceding example to place the NumericStepper on top and repeat the loop to display the components' new positions in the display list:

```
this.addChildAt(aNs, numChildren - 1);
i = 0;
while(i < numChildren) {
    trace(getChildAt(i).name + " is at position: " + i++);
}
```

You should see the following in the Output panel:

```
aNs is at position: 0
aButton is at position: 1
aCb is at position: 2
aButton is at position: 0
aCb is at position: 1
aNs is at position: 2
```

The NumericStepper should also appear in front of the other components on the screen.

Note that `numChildren` is the number of objects (from 1 to *n*) in the display list while the first position in the list is 0. So if there are three objects in the list, the index position of the third object is 2. This means that you can reference the last position in the display list, or the top object in terms of display depth, as `numChildren - 1`.

Remove a component from the display list

You can remove a component from a display object container and its display list with the `removeChild()` and `removeChildAt()` methods. The following example places three Button components in front of each other on the Stage and adds an event listener for each of them. When you click each Button, the event handler removes it from the display list and the Stage.

1. Create a new Flash file (ActionScript 3.0) document.
2. Drag a Button from the Components panel to the Library panel.
3. Open the Actions panel, select Frame 1 in the main Timeline and add the following code:

```
import fl.controls.Button;

var i:int = 0;
while(i++ < 3) {
    makeButton(i);
}
function removeButton(event:MouseEvent):void {
    removeChildAt(numChildren -1);
}
function makeButton(num) {
    var aButton:Button = new Button();
    aButton.name = "Button" + num;
    aButton.label = aButton.name;
    aButton.move(200, 200);
    addChild(aButton);
    aButton.addEventListener(MouseEvent.CLICK, removeButton);
}
```

For a complete explanation of the display list, see “Display programming” in *Programming ActionScript 3.0*.

Work with FocusManager

When a user presses the Tab key to navigate in a Flash application or clicks in an application, the FocusManager class determines which component receives input focus. You don’t need to add a FocusManager instance to an application or write any code to activate the FocusManager unless you are creating a component.

If a RadioButton object receives focus, the FocusManager examines that object and all objects with the same `groupName` value and sets focus on the object with the `selected` property set to `true`.

Each modal Window component contains an instance of the FocusManager, so the controls on that window become their own tab set. This prevents a user from inadvertently navigating to components in other windows by pressing the Tab key.

The FocusManager uses the depth level (or *z*-order) of elements in the container as the default navigation scheme or *tab loop*. A user typically navigates the tab loop by using the Tab key, with focus moving from the first component that has focus, to the last, and then back again to the first.The depth levels are set up primarily by the order in which components are dragged to the Stage; however, you can also use the Modify > Arrange > Bring To Front/Send To Back commands to determine the final *z*-order. For more information on depth levels, see “Work with the display list” on page 24.

You can call the `setFocus()` method to give focus to a component instance in an application. For example, the following example creates a FocusManager instance for the current container (`this`) and gives focus to the Button instance `aButton`.

```
var fm:FocusManager = new FocusManager(this);
fm.setFocus(aButton);
```

You can determine which component has focus by calling the `getFocus()` method and you can determine which component in the tab loop will receive focus next by calling the `getNextFocusManagerComponent()` method. In the following example, a CheckBox, a RadioButton, and a Button are on the Stage and each component has listeners for `MouseEvent.CLICK` and `FocusEvent.MOUSE_FOCUS_CHANGE` events. When the `MouseEvent.CLICK` event occurs, because the user clicked on the component, the `showFocus()` function calls the `getNextFocusManagerComponent()` method to determine which component in the tab loop would receive focus next. It then calls the `setFocus()` method to give focus to that component. When the `FocusEvent.MOUSE_FOCUS_CHANGE` event occurs, the `fc()` function displays the name of the component on which this event occurred. This event is triggered when the user clicks on a component other than the next one in the tab loop.

```
// This example assumes a CheckBox (aCh), a RadioButton (aRb) and a Button
// (aButton) have been placed on the Stage.

import fl.managers.FocusManager;
import flash.display.InteractiveObject;

var fm:FocusManager = new FocusManager(this);

aCh.addEventListener(MouseEvent.CLICK, showFocus);
aRb.addEventListener(MouseEvent.CLICK, showFocus);
aButton.addEventListener(MouseEvent.CLICK, showFocus);
aCh.addEventListener(FocusEvent.MOUSE_FOCUS_CHANGE, fc);
aRb.addEventListener(FocusEvent.MOUSE_FOCUS_CHANGE, fc);
aButton.addEventListener(FocusEvent.MOUSE_FOCUS_CHANGE, fc);

function showFocus(event:MouseEvent):void {
    var nextComponent:InteractiveObject = fm.getNextFocusManagerComponent();
    trace("Next component in tab loop is: " + nextComponent.name);
    fm.setFocus(nextComponent);
}

function fc(fe:FocusEvent):void {
    trace("Focus Change: " + fe.target.name);
}
```

To create a Button that receives focus when a user presses Enter (Windows) or Return (Macintosh), set the `FocusManager.defaultButton` property to the Button instance that you want to be the default Button, as in the following code:

```
import fl.managers.FocusManager;

var fm:FocusManager = new FocusManager(this);
fm.defaultButton = okButton;
```

The FocusManager class overrides the default Flash Player focus rectangle and draws a custom focus rectangle with rounded corners.

For more information about creating a focus scheme in a Flash application, see the `FocusManager class` in the *ActionScript 3.0 Reference for the Adobe Flash Platform*. To create a custom focus manager, you must create a class that implements the *IFocusManager* interface. For more information, see `IFocusManager` in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Work with List-based components

The List, DataGrid, and TileList components all inherit from the SelectableList base class. For this reason, these components are considered List-based components. A ComboBox consists of a text box and a List so it, too, is a List-based component.

A List is composed of rows. A DataGrid and a TileList are composed of rows that can be divided into multiple columns. The intersection of a row and a column is a cell. In a List, which is a single column of rows, each row is a cell. A cell has the following two important aspects:

- The data values that cells hold are called items. An *item* is an ActionScript object used for storing the units of information in a List. A List can be thought of as an array with each indexed space of the array being an item. In a List, an item is an object that typically has a `label` property that is displayed and a `data` property that is used for storing data. A *data provider* is a data model of the items in a List. A data provider allows you to populate a List-based component simply by assigning it to the component's `dataProvider` property.
- A cell can hold different types of data that range from text to images, MovieClips, or any class that you can create. For this reason, a cell must be drawn or rendered in a way that is appropriate for its content. Consequently, List-based components have a *cell renderer* to render its cells. In the case of the DataGrid, each column is a DataGridColumn object, which also has a `cellRenderer` property, so that each column can be rendered appropriately for its content.

  All List-based components have `cellRenderer` and `dataProvider` properties that you can set to load and render the cells of these components. For information on using these properties and working with List-based components, see "Work with a DataProvider" on page 28 and "Work with a CellRenderer" on page 36.

Work with a DataProvider

A DataProvider is a data source that you can use to supply data to the ComboBox, DataGrid, List, and TileList components. Each of these component classes has a `dataProvider` property to which you can assign a DataProvider object to populate the component's cells with data. Typically, a data provider is a collection of data such as an Array or XML object.

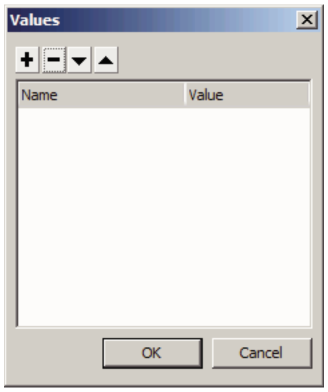
Create a DataProvider

For the ComboBox, List, and TileList components you can create a DataProvider using the `dataProvider` parameter in the authoring environment. The DataGrid component does not have a dataProvider parameter in the Property inspector because it can have several columns and its data provider is consequently more complex. You can also use ActionScript to create a DataProvider for these components, as well as for the DataGrid.

Use the dataProvider parameter

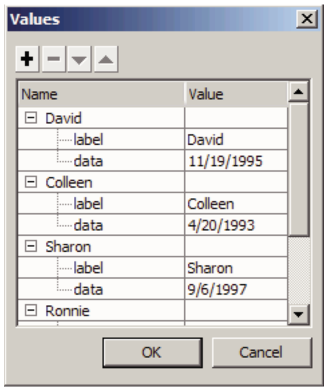
You can create a simple data provider for the ComboBox, List, and TileList components by clicking the dataProvider parameter on the Parameters tab of the Property inspector or the Component inspector.

If you double-click the value cell, which initially shows an empty Array, you will open the Values dialog box, which allows you to enter multiple label and data values to create the data provider.



Values dialog box for dataProvider

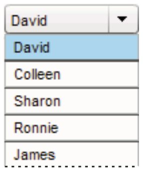
Click the plus sign to add an item to the dataProvider. Click the minus sign to delete an item. Click the up arrow to move a selected item up in the list or click the down arrow to move a selected item down in the list. The following illustration shows a Values dialog box that creates a list of children's names and their birthdays.



Name	Value
David	
label	David
data	11/19/1995
Colleen	
label	Colleen
data	4/20/1993
Sharon	
label	Sharon
data	9/6/1997
Ronnie	

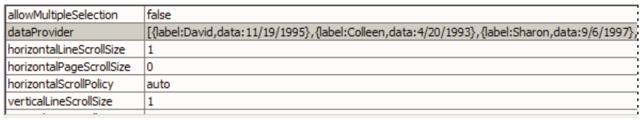
Values dialog box with data

The Array you create consists of pairs of label and value fields. The label fields are `label` and `data` and the value fields are the children's names and their birthdays. The label field identifies the content that appears in the List, which in this case is the names of the children. The resulting ComboBox looks like this:



The ComboBox populated by the DataProvider

When you finish adding data, click OK to close the dialog box. The Array in the dataProvider parameter is now populated with the items that you created.



allowMultipleSelection	false
dataProvider	[{label:David,data:11/19/1995},{label:Colleen,data:4/20/1993},{label:Sharon,data:9/6/1997},
horizontalLineScrollSize	1
horizontalPageScrollSize	0
horizontalScrollPolicy	auto
verticalLineScrollSize	1

dataProvider parameter with data

You can access the label and data values that you created by using ActionScript to access the component's `dataProvider` property.

Create a DataProvider using ActionScript

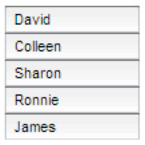
You can create a DataProvider by creating the data in an Array or XML object and providing the object as the `value` parameter to the DataProvider constructor.

***Note:** In ActionScript 3.0, you cannot assign an Array or XML object directly to a dataProvider property because the property is defined as a DataProvider object and can only receive an object of the DataProvider type.*

The following example populates a List component, which is a single column of rows, with the names of several children and their birthdays. The example defines the list in the `items` Array and supplies it as the parameter when it creates the DataProvider instance (`new DataProvider(items)`) and assigns it to the `dataProvider` property of the List component.

```
import fl.controls.List;
import fl.data.DataProvider;

var aList:List = new List();
var items:Array = [
{label:"David", data:"11/19/1995"},
{label:"Colleen", data:"4/20/1993"},
{label:"Sharon", data:"9/06/1997"},
{label:"Ronnie", data:"7/6/1993"},
{label:"James", data:"2/15/1994"},
];
aList.dataProvider = new DataProvider(items);
addChild(aList);
aList.move(150,150);
```

The Array consists of pairs of label and value fields. The label fields are `label` and `data` and the value fields are the children's names and their birthdays. The label field identifies the content that appears in the List, which in this case is the names of the children. The resulting List looks like this:

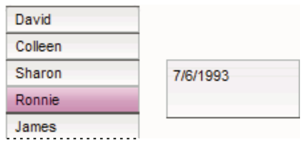


A List populated by a DataProvider

The value of the data field is available when the user selects an item in the list by clicking it and causing a `change` event. The following example adds a TextArea (`aTa`) and an event handler (`changeHandler`) to the preceding example to display the child's birthday when a user selects a name in the List.

```
import fl.controls.List;
import fl.controls.TextArea;
import flash.events.Event;
import fl.data.DataProvider;

var aList:List = new List();
var aTa:TextArea = new TextArea();
var items:Array = [
{label:"David", data:"1/19/1995"},
{label:"Colleen", data:"4/20/1993"},
{label:"Sharon", data:"9/06/1994"},
{label:"Ronnie", data:"7/6/1993"},
{label:"James", data:"2/15/1994"},
];
aList.dataProvider = new DataProvider(items);

addChild(aList);
addChild(aTa);

aList.move(150,150);
aTa.move(150, 260);

aList.addEventListener(Event.CHANGE, changeHandler);

function changeHandler(event:Event):void {
    aTa.text = event.target.selectedItem.data;
};
```

Now when a user selects a child's name in the List, the child's birthday displays in the TextArea as shown in the following illustration. This is accomplished by the `changeHandler()` function when it sets the `text` property of the TextArea (`aTa.text`) to the value of the data field in the selected item (`event.target.selectedItem.data`). The `event.target` property is the object that triggered the event, which in this case is the List.



Displaying the data field from a List's DataProvider

You can include data other than text in a DataProvider. The following example includes MovieClips in a DataProvider that supplies data to a TileList. It builds the DataProvider by calling `addItem()` to add each item after it creates the MovieClip, a colored box.

```
import fl.data.DataProvider;
import flash.display.DisplayObject;

var aBox:MovieClip = new MovieClip();
var i:uint = 0;
var colors:Array = new Array(0x00000, 0xFF0000, 0x0000CC, 0x00CC00, 0xFFFF00);
var colorNames:Array = new Array("Midnight", "Cranberry", "Sky", "Forest", "July");
var dp:DataProvider = new DataProvider();
for(i=0; i < colors.length; i++) {
    drawBox(aBox, colors[i]);// draw box w next color in array
    dp.addItem( {label:colorNames[i], source:aBox} );
}
aTl.dataProvider = dp;
aTl.columnWidth = 110;
aTl.rowHeight = 130;
aTl.setSize(280,150);
aTl.move(150, 150);
aTl.setStyle("contentPadding", 5);

function drawBox(box:MovieClip,color:uint):void {
            box.graphics.beginFill(color, 1.0);
            box.graphics.drawRect(0, 0, 100, 100);
            box.graphics.endFill();
```

You can also use XML data (instead of an array) to populate a DataProvider object. For example, the following code stores data in an XML object named `employeesXML`, and then passes that object as the value parameter of the `DataProvider()` constructor function:

```
import fl.controls.DataGrid;
import fl.data.DataProvider;

var aDg:DataGrid = new DataGrid();
addChild(aDg);

var employeesXML:XML =
    <employees>
        <employee Name="Edna" ID="22" />
        <employee Name="Stu" ID="23" />
    </employees>;

var myDP:DataProvider = new DataProvider(employeesXML);

aDg.columns = ["Name", "ID"];
aDg.dataProvider = myDP;
```

You can provide data as attributes of the XML data, as in the previous code, or as properties of the XML data, as in the following code:

```
var employeesXML:XML =
    <employees>
        <employee>
            <Name>Edna</Name>
            <ID>22</ID>
        </employee>
        <employee>
            <Name>Stu</Name>
            <ID>23</ID>
        </employee>
    </employees>;
```

The DataProvider also has a set of methods and properties that allow you to access and manipulate it. You can use the DataProvider API to add, remove, replace, sort, and merge items in a DataProvider.

Manipulate a DataProvider

You can add items to a DataProvider with the `addItem()` and `addItemAt()` methods. The following example adds items that a user enters into the text field of an editable ComboBox. It assumes a ComboBox has been dragged onto the Stage and given an instance name of `aCb`.

```
import fl.data.DataProvider;
import fl.events.ComponentEvent;

var items:Array = [
{label:"Roger"},
{label:"Carolyn"},
{label:"Darrell"},
{label:"Rebecca"},
{label:"Natalie"},
{label:"Mitchell"},
];
aCb.dataProvider = new DataProvider(items);

aCb.addEventListener(ComponentEvent.ENTER, newItemHandler);

function newItemHandler(event:ComponentEvent):void {
    var newRow:int = event.target.length + 1;
        event.target.addItemAt({label:event.target.selectedLabel},
        event.target.length);
}
```

You can also replace and remove items in a component through its DataProvider. The following example implements two separate List components, `listA` and `listB`, and provides a Button labeled Sync. When a user clicks the Button, the example uses the `replaceItemAt()` method to replace the items in `listB` with the items in `listA`. If `listA` is longer than `listB`, the example calls the `addItem()` method to add the extra items to `listB`. If `listB` is longer than `listA`, the example calls the `removeItemAt()` method to remove the extra items in `ListB`.

```
// Requires the List and Button components to be in the library

import fl.controls.List;
import fl.controls.Button;
import flash.events.Event;
import fl.data.DataProvider;

var listA:List = new List();
var listB:List = new List();
var syncButton:Button = new Button();
syncButton.label = "Sync";

var itemsA:Array = [
{label:"David"},
{label:"Colleen"},
{label:"Sharon"},
{label:"Ronnie"},
{label:"James"},
];
var itemsB:Array = [
{label:"Roger"},
{label:"Carolyn"},
{label:"Darrell"},
{label:"Rebecca"},
{label:"Natalie"},
{label:"Mitchell"},
];
listA.dataProvider = new DataProvider(itemsA);
listB.dataProvider = new DataProvider(itemsB);

addChild(listA);
addChild(listB);
addChild(syncButton);

listA.move(100, 100);
listB.move(250, 100);
syncButton.move(175, 220);

syncButton.addEventListener(MouseEvent.CLICK, syncHandler);

function syncHandler(event:MouseEvent):void {
    var i:uint = 0;
    if(listA.length > listB.length) { //if listA is longer, add items to B
        while(i < listB.length) {
            listB.dataProvider.replaceItemAt(listA.dataProvider.getItemAt(i), i);
            ++i;
        }
        while(i < listA.length) {
            listB.dataProvider.addItem(listA.dataProvider.getItemAt(i++));
        }
    } else if(listA.length == listB.length) { //if listA and listB are equal length
        while(i < listB.length) {
            listB.dataProvider.replaceItemAt(listA.dataProvider.getItemAt(i), i);
            ++i;
        }
    } else {        //if listB is longer, remove extra items from B
        while(i < listA.length) {
            listB.dataProvider.replaceItemAt(listA.dataProvider.getItemAt(i), i);
            ++i;
        }
        while(i < listB.length) {
            listB.dataProvider.removeItemAt(i++);
        }
    }
}
```

You can also merge with and sort a DataProvider using the `merge()`, `sort()`, and `sortOn()` methods. The following example populates two DataGrid instances (`aDg` and `bDg`) with partial rosters for two softball teams. It adds a Button with a label of Merge, and when the user clicks it, the event handler (`mrgHandler`) merges the roster for `bDg` with the roster for `aDg` and sorts the resulting DataGrid on the Name column.

```
import fl.data.DataProvider;
import fl.controls.DataGrid;
import fl.controls.Button;

var aDg:DataGrid = new DataGrid();
var bDg:DataGrid = new DataGrid();
var mrgButton:Button = new Button();
addChild(aDg);
addChild(bDg);
addChild(mrgButton);
bldRosterGrid(aDg);
bldRosterGrid(bDg);
var aRoster:Array = new Array();
var bRoster:Array = new Array();
aRoster = [
        {Name:"Wilma Carter", Bats:"R", Throws:"R", Year:"So", Home: "Redlands, CA"},
        {Name:"Sue Pennypacker", Bats:"L", Throws:"R", Year:"Fr", Home: "Athens, GA"},
        {Name:"Jill Smithfield", Bats:"R", Throws:"L", Year:"Sr", Home: "Spokane, WA"},
        {Name:"Shirley Goth", Bats:"R", Throws:"R", Year:"Sr", Home: "Carson, NV"}
];
bRoster = [
        {Name:"Angelina Davis", Bats:"R", Throws:"R", Year:"So", Home: "Odessa, TX"},
        {Name:"Maria Santiago", Bats:"L", Throws:"L", Year:"Sr", Home: "Tacoma, WA"},
        {Name:"Debbie Ferguson", Bats:"R", Throws:"R", Year: "Jr", Home: "Bend, OR"}
];
aDg.dataProvider = new DataProvider(aRoster);
bDg.dataProvider = new DataProvider(bRoster);
aDg.move(50,50);
aDg.rowCount = aDg.length;
```

```
bDg.move(50,200);
bDg.rowCount = bDg.length;
mrgButton.label = "Merge";
mrgButton.move(200, 315);
mrgButton.addEventListener(MouseEvent.CLICK, mrgHandler);

function bldRosterGrid(dg:DataGrid){
    dg.setSize(400, 300);
    dg.columns = ["Name", "Bats", "Throws", "Year", "Home"];
    dg.columns[0].width = 120;
    dg.columns[1].width = 50;
    dg.columns[2].width = 50;
    dg.columns[3].width = 40;
    dg.columns[4].width = 120;
};

function mrgHandler(event:MouseEvent):void {
    aDg.dataProvider.merge(bDg.dataProvider);
    aDg.dataProvider.sortOn("Name");
}
```

For more information, see the DataProvider class in the ActionScript 3.0 Reference for the Adobe Flash Platform.

Work with a CellRenderer

CellRenderer is a class that List-based components, such as List, DataGrid, TileList, and ComboBox, use to manipulate and display custom cell content for their rows. A customized cell can contain text, a prebuilt component, such as a CheckBox, or any display object class you can create. To render data using a custom CellRenderer, you can either extend the CellRenderer class or implement the ICellRenderer interface to create your own custom CellRenderer class.

The List, DataGrid, TileList, and ComboBox classes are subclasses of the SelectableList class. The SelectableList class includes a `cellRenderer` style. This style defines the display object that the component uses to render cells.

You can adjust the formatting of the styles used by the CellRenderer, by calling the `setRendererStyle()` method of the List object (see "Format cells" on page 36). Or you can define a custom class to use as the CellRenderer (see "Define a custom CellRenderer class" on page 37).

Format cells

The CellRenderer class includes a number of styles that let you control the format of the cell.

The following styles let you define the skins used for the different states of the cell (disabled, down, over, and up):

- `disabledSkin` and `selectedDisabledSkin`
- `downSkin` and `selectedDownSkin`
- `overSkin` and `selectedOverSkin`
- `upSkin` and `selectedUpSkin`

  The following styles apply to the text formatting:

- `disabledTextFormat`
- `textFormat`
- `textPadding`

You can set these styles by calling the `setRendererStyle()` method of the List object or by calling the `setStyle()` method of the CellRenderer object. You can get these styles by calling the `getRendererStyle()` method of the List object or by calling the `getStyle()` method of the CellRenderer object. You can also access an object that defines all renderer styles (as named properties of the object) via the `rendererStyles` property of the List object or the `getStyleDefinition()` method of the CellRenderer object.

You can call the `clearRendererStyle()` method to reset a style to its default value.

To get or set the height of the rows in the list, use the `rowHeight` property of the List object.

Define a custom CellRenderer class

Create a class that extends the CellRenderer class to define a custom CellRenderer

For example, the following code includes two classes. The ListSample class instantiates a List component, and it uses the other class, CustomRenderer, to define the cell renderer to use for the List component. The CustomRenderer class extends the CellRenderer class.

1 Select File > New.

2 In the New Document dialog box that is displayed, select Flash File (ActionScript 3.0), and then click OK.

3 Select Window > Components to display the Components panel.

4 In the Components panel, drag a List component to the Stage.

5 If Flash is not displaying the Property inspector, select Window > Properties > Properties.

6 With the List component selected, set the properties in the Property inspector:

- Instance Name: myList
- W (width): 200
- H (height): 300
- X: 20
- Y: 20

7 Select Frame 1 of Layer 1 in the Timeline, and select Window > Actions.

8 Type the following script in the Actions panel:

```
myList.setStyle("cellRenderer", CustomCellRenderer);
myList.addItem({label:"Burger -- $5.95"});
myList.addItem({label:"Fries -- $1.95"});
```

9 Select File > Save. Give the file a name and click the OK button.

10 Select File > New.

11 In the New Document dialog box that is displayed, select ActionScript File and then click the OK button.

12 In the script window, enter the following code to define the CustomCellRenderer class:

```
package {
    import fl.controls.listClasses.CellRenderer;
    import flash.text.TextFormat;
    import flash.filters.BevelFilter;
    public class CustomCellRenderer extends CellRenderer {
        public function CustomCellRenderer() {
            var format:TextFormat = new TextFormat("Verdana", 12);
            setStyle("textFormat", format);
            this.filters = [new BevelFilter()];
        }
    }
}
```

13 Select File > Save. Name the file CustomCellRenderer.as, put it in the same directory as the FLA file, and click the OK button.

14 Select Control > Test Movie.

Use a class that implements the ICellRenderer interface to define a custom CellRenderer

You can also define a CellRenderer using any class that inherits the DisplayObject class and implements the ICellRenderer interface. For example the following code defines two classes. The ListSample2 class adds a List object to the display list and defines its CellRenderer to use the CustomRenderer class. The CustomRenderer class extends the CheckBox class (which extends the DisplayObject class) and implements the ICellRenderer interface. Note that the CustomRenderer class defines getter and setter methods for the `data` and `listData` properties, defined in the ICellRenderer interface. Other properties and methods defined in the ICellRenderer interface (the `selected` property and the `setSize()` method) are already defined in the CheckBox class:

1 Select File > New.

2 In the New Document dialog box that is displayed, select Flash File (ActionScript 3.0), and then click OK.

3 Select Window > Components to display the Components panel.

4 In the Components panel, drag a List component to the Stage.

5 If Flash is not displaying the Property inspector, select Window > Properties > Properties.

6 With the List component selected, set the properties in the Property inspector:

- Instance Name: myList
- W (width): 100
- H (height): 300
- X: 20
- Y: 20

7 Select Frame 1 of Layer 1 in the Timeline, and select Window > Actions.

8 Type the following script in the Actions panel:

```
myList.setStyle("cellRenderer", CustomCellRenderer);
myList.addItem({name:"Burger", price:"$5.95"});
myList.addItem({name:"Fries", price:"$1.95"});
```

9 Select File > Save. Give the file a name and click the OK button.

10 Select File > New.

11 In the New Document dialog box that is displayed, select ActionScript File and then click the OK button.

12 In the script window, enter the following code to define the CustomCellRenderer class:

```
package
{
    import fl.controls.CheckBox;
    import fl.controls.listClasses.ICellRenderer;
    import fl.controls.listClasses.ListData;
    public class CustomCellRenderer extends CheckBox implements ICellRenderer {
        private var _listData:ListData;
        private var _data:Object;
        public function CustomCellRenderer() {
        }
        public function set data(d:Object):void {
            _data = d;
            label = d.label;
        }
        public function get data():Object {
            return _data;
        }
        public function set listData(ld:ListData):void {
            _listData = ld;
        }
        public function get listData():ListData {
            return _listData;
        }
    }
}
```

13 Select File > Save. Name the file CustomCellRenderer.as, put it in the same directory as the FLA file, and click the OK button.

14 Select Control > Test Movie.

Use a symbol to define a CellRenderer

You can also use a symbol in the library to define a CellRenderer. The symbol must be exported for ActionScript and the class name for the library symbol must have an associated class file that either implements the ICellRenderer interface or that extends the CellRenderer class (or one of its subclasses).

The following example defines a custom CellRenderer using a library symbol.

1 Select File > New.

2 In the New Document dialog box that is displayed, select Flash File (ActionScript 3.0), and then click OK.

3 Select Window > Components to display the Components panel.

4 In the Components panel, drag a List component to the Stage.

5 If Flash is not displaying the Property inspector, select Window > Properties > Properties.

6 With the List component selected, set the properties in the Property inspector:

- Instance Name: myList
- W (width): 100
- H (height): 400
- X: 20
- Y: 20

7 Click the Parameters panel, and then double-click the second column in the dataProvider row.

8 In the Values dialog box that is displayed, click the plus sign twice to add two data elements (with labels set to label0 and label1), and then click the OK button.

9 With the Text tool, draw a text field on the Stage.

10 With the text field selected, set the properties in the Property inspector:

- Text type: Dynamic Text
- Instance Name: textField
- W (width): 100
- Font size: 24
- X: 0
- Y: 0

11 With the text field selected, select Modify > Convert To Symbol.

12 In the Convert To Symbol dialog box, make the following settings and then click OK.

- Name: MyCellRenderer
- Type: MovieClip
- Export for ActionScript: Selected
- Export in First Frame: Selected
- Class: MyCellRenderer
- Base Class: flash.display.MovieClip

  If Flash displays an ActionScript Class Warning, click the OK button in the warning box.

13 Delete the instance of the new movie clip symbol from the Stage.

14 Select Frame 1 of Layer 1 in the Timeline, and select Window > Actions.

15 Type the following script in the Actions panel:

```
myList.setStyle("cellRenderer", MyCellRenderer);
```

16 Select File > Save. Give the file a name and click the OK button.

17 Select File > New.

18 In the New Document dialog box that is displayed, select ActionScript File and then click the OK button.

19 In the script window, enter the following code to define the MyCellRenderer class:

```
package {
    import flash.display.MovieClip;
    import flash.filters.GlowFilter;
    import flash.text.TextField;
    import fl.controls.listClasses.ICellRenderer;
    import fl.controls.listClasses.ListData;
    import flash.utils.setInterval;
    public class MyCellRenderer extends MovieClip implements ICellRenderer {
        private var _listData:ListData;
        private var _data:Object;
        private var _selected:Boolean;
        private var glowFilter:GlowFilter;
        public function MyCellRenderer() {
            glowFilter = new GlowFilter(0xFFFF00);
            setInterval(toggleFilter, 200);
        }
        public function set data(d:Object):void {
            _data = d;
            textField.text = d.label;
        }
        public function get data():Object {
            return _data;
        }
        public function set listData(ld:ListData):void {
            _listData = ld;
        }
        public function get listData():ListData {
            return _listData;
        }
        public function set selected(s:Boolean):void {
            _selected = s;
        }
        public function get selected():Boolean {
            return _selected;
        }
        public function setSize(width:Number, height:Number):void {
        }
        public function setStyle(style:String, value:Object):void {
        }
        public function setMouseState(state:String):void{
        }
        private function toggleFilter():void {
            if (textField.filters.length == 0) {
                textField.filters = [glowFilter];
            } else {
                textField.filters = [];
            }
        }
    }
}
```

20 Select File > Save. Name the file MyCellRenderer.as, put it in the same directory as the FLA file, and click the OK button.

21 Select Control > Test Movie.

CellRenderer properties

The `data` property is an object that contains all properties that are set for the CellRenderer. For example, in the following class, which defines a custom CellRenderer that extends the Checkbox class, note that the setter function for the `data` property passes the value of `data.label` to the `label` property that is inherited from the CheckBox class:

```
    public class CustomRenderer extends CheckBox implements ICellRenderer {
        private var _listData:ListData;
        private var _data:Object;
        public function CustomRenderer() {
        }
        public function set data(d:Object):void {
            _data = d;
            label = d.label;
        }
        public function get data():Object {
            return _data;
        }
        public function set listData(ld:ListData):void {
            _listData = ld;
        }
        public function get listData():ListData {
            return _listData;
        }
    }
}
```

The `selected` property defines whether or not a cell is selected in the list.

Apply a CellRenderer for a column of a DataGrid object

A DataGrid object can have multiple columns and you can specify different cell renderers for each column. Each column of a DataGrid is represented by a DataGridColumn object and the DataGridColumn class includes a `cellRenderer` property, for which you can define the CellRenderer for the column.

Define a CellRenderer for an editable cell

The DataGridCellEditor class defines a renderer used for editable cells in a DataGrid object. It becomes the renderer for a cell when the `editable` property of the DataGrid object is set to `true` and the user clicks the cell to be edited. To define a CellRenderer for the editable cell, set the `itemEditor` property for each element of the `columns` array of the DataGrid object.

Use an image, SWF file, or movie clip as a CellRenderer

The ImageCell class, a subclass of CellRenderer, defines an object used to render cells in which the main content of the cell is an image, SWF file, or movie clip. The ImageCell class includes the following styles for defining the appearance of the cell:

- `imagePadding`—The padding that separates the edge of the cell from the edge of the image, in pixels
- `selectedSkin`—The skin that is used to indicate the selected state
- `textOverlayAlpha`—The opacity of the overlay behind the cell label
- `textPadding`—The padding that separates the edge of the cell from the edge of the text, in pixels

  The ImageCell class is the default CellRenderer for the TileList class.

Make components accessible

You can make visual content in your Flash applications accessible to visually impaired users through a screen reader, which provides an audio description of the screen's content. For information on how to make your Flash application accessible to a screen reader, see Chapter 18, "Creating Accessible Content," in *Using Flash*.

To make an ActionScript 3.0 component accessible to a screen reader, you must also import its accessibility class and call that class's `enableAccessibility()` method. You can make the following ActionScript 3.0 components accessible to a screen reader:

Component	**Accessibility Class**
Button	`ButtonAccImpl`
CheckBox	`CheckBoxAccImpl`
ComboBox	`ComboBoxAccImpl`
List	`ListAccImpl`
RadioButton	`RadioButtonAccImpl`
TileList	`TileListAccImpl`

The component accessibility classes are in the `fl.accessibility` package. To make a CheckBox accessible to a screen reader, for example, you would add the following statements to your application:

```
import fl.accessibility.CheckBoxAccImpl;

CheckBoxAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances you create.

***Note:** Enabling accessibility marginally increases file size by including the required classes during compilation.*

Most components are also navigable through the keyboard. For more information on enabling accessible components and navigating with the keyboard, see the User Interaction sections of "Using the UI Components" on page 44 and the accessibility classes in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Chapter 4: Using the UI Components

This chapter explains how to use the ActionScript 3.0 user interface (UI) components included with Flash.

Use the Button component

The Button component is a resizable, rectangular button that a user can press with the mouse or the spacebar to initiate an action in the application. You can add a custom icon to a Button. You can also change the behavior of a Button from push to toggle. A toggle Button stays pressed when clicked and returns to its up state when clicked again.

A Button is a fundamental part of many forms and web applications. You can use buttons wherever you want a user to initiate an event. For example, most forms have a Submit button. You could also add Previous and Next buttons to a presentation.

User interaction with the Button component

You can enable or disable a button in an application. In the disabled state, a button doesn't receive mouse or keyboard input. An enabled button receives focus if you click it or tab to it. When a Button instance has focus, you can use the following keys to control it:

Key	Description
Shift+Tab	Moves focus to the previous object.
Spacebar	Presses or releases the button and triggers the `click` event.
Tab	Moves focus to the next object.
Enter/Return	Moves focus to the next object if a button is set as the FocusManager's default Button.

For more information about controlling focus, see the IFocusManager interface and the FocusManager class in the *ActionScript 3.0 Reference for the Adobe Flash Platform* and "Work with FocusManager" on page 26.

A live preview of each Button instance reflects changes made to parameters in the Property inspector or Component inspector during authoring.

Note: *If an icon is larger than the button, it extends beyond the button's borders.*

To designate a button as the default push button in an application (the button that receives the click event when a user presses Enter), set `FocusManager.defaultButton`. For example, the following code sets the default button to be a Button instance called `submitButton`.

```
FocusManager.defaultButton = submitButton;
```

When you add the Button component to an application, you can make it accessible to a screen reader by adding the following lines of ActionScript code:

```
import fl.accessibility.ButtonAccImpl;

ButtonAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances you create.

Button component parameters

You can set the following authoring parameters in the Property inspector (Window > Properties > Properties) or in the Component inspector (Window > Component Inspector) for each Button instance: `emphasized`, `label`, `labelPlacement`, `selected`, and `toggle`. Each of these parameters has a corresponding ActionScript property of the same name. When you assign a value to these parameters you are setting the initial state of the property in the application. Setting the property in ActionScript overrides the value you set in the parameter. For information on the possible values for these parameters, see the Button class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Create an application with the Button component

The following procedure explains how to add a Button component to an application while authoring. In this example, the Button changes the state of a ColorPicker component when you click it.

1. Create a new Flash (ActionScript 3.0) document.
2. Drag a Button component from the Components panel to the Stage and enter the following values for it in the Property inspector:
   - Enter the instance name **aButton**.
   - Enter **Show** for the label parameter.
3. Add a ColorPicker to the Stage and give it an instance name of **aCp**.
4. Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
aCp.visible = false;

aButton.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:MouseEvent):void {

    switch(event.currentTarget.label) {
        case "Show":
            aCp.visible = true;
            aButton.label = "Disable";
            break;
        case "Disable":
            aCp.enabled = false;
            aButton.label = "Enable";
            break;
        case "Enable":
            aCp.enabled = true;
            aButton.label = "Hide";
            break;
        case "Hide":
            aCp.visible = false;
            aButton.label = "Show";
            break;
    }
}
```

   The second line of code registers the function `clickHandler()` as the event handler function for the `MouseEvent.CLICK` event. The event occurs when a user clicks the Button, causing the `clickHandler()` function to take one of the following actions depending on the Button's value:

   - Show makes the ColorPicker visible and changes the Button's label to Disable.

- Disable disables the ColorPicker and changes the Button's label to Enable.
- Enable enables the ColorPicker and changes the Button's label to Hide.
- Hide makes the ColorPicker invisible and changes the Button's label to Show.

5 Select Control > Test Movie to run the application.

Create an application with the Button component

The following procedure creates a toggle Button using ActionScript and displays the event type in the Output panel when you click the Button. The example creates the Button instance by invoking the class's constructor and it adds it to the Stage by calling the `addChild()` method.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the Button component from the Components panel to the current document's Library panel.

This adds the component to the library, but doesn't make it visible in the application.

3 Open the Actions panel, select Frame 1 in the main Timeline and enter the following code to create a Button instance:

```
import fl.controls.Button;

var aButton:Button = new Button();
addChild(aButton);
aButton.label = "Click me";
aButton.toggle =true;
aButton.move(50, 50);
```

The `move()` method positions the button at location 50 (x coordinate), 50 (y coordinate) on the Stage.

4 Now, add the following ActionScript to create an event listener and an event handler function:

```
aButton.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:MouseEvent):void {
    trace("Event type: " + event.type);
}
```

5 Select Control > Test Movie.

When you click the button, Flash displays the message, "Event type: click" in the Output panel.

Use the CheckBox component

A CheckBox is a square box that can be selected or deselected. When it is selected, a check mark appears in the box. You can add a text label to a CheckBox and place it to the left, right, above, or below the CheckBox.

You can use CheckBoxes to gather a set of `true` or `false` values that aren't mutually exclusive. For example, an application that gathers information about what kind of car you want to buy might use CheckBoxes to let you select features.

User interaction with the CheckBox

You can enable or disable a CheckBox in an application. If a CheckBox is enabled and a user clicks it or its label, the CheckBox receives input focus and displays its pressed appearance. If a user moves the pointer outside the bounding area of a CheckBox or its label while pressing the mouse button, the component's appearance returns to its original state and retains input focus. The state of a CheckBox does not change until the mouse is released over the component. Additionally, the CheckBox has two disabled states, selected and deselected, which use `selectedDisabledSkin` and `disabledSkin`, respectively, that do not allow mouse or keyboard interaction.

If a CheckBox is disabled, it displays its disabled appearance, regardless of user interaction. In the disabled state, a CheckBox doesn't receive mouse or keyboard input.

A CheckBox instance receives focus if a user clicks it or tabs to it. When a CheckBox instance has focus, you can use the following keys to control it:

Key	Description
Shift+Tab	Moves focus to the previous element.
Spacebar	Selects or deselects the component and triggers the `change` event.
Tab	Moves focus to the next element.

For more information about controlling focus, see "Work with FocusManager" on page 26 and the FocusManager class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

A live preview of each CheckBox instance reflects changes made to parameters in the Property inspector or Component inspector during authoring.

When you add the CheckBox component to an application, you can make it accessible to a screen reader by adding the following lines of ActionScript code:

```
import fl.accessibility.CheckBoxAccImpl;

CheckBoxAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances you have of the component.

CheckBox component parameters

You can set the following authoring parameters in the Property inspector or in the Component inspector for each CheckBox component instance: `label`, `labelPlacement`, and `selected`. Each of these parameters has a corresponding ActionScript property of the same name. For information on the possible values for these parameters, see the CheckBox class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Create an application using the CheckBox

The following procedure explains how to add a CheckBox component to an application while authoring, using an excerpt from a loan application form. The form asks if the applicant is a home owner and provides a CheckBox for the user to answer "yes." If so, the form presents two radio buttons for the user to indicate the relative value of the house.

Create an application using the Checkbox component

1. Create a new Flash (ActionScript 3.0) document.
2. Drag a CheckBox component from the Components panel to the Stage.

3 In the Property inspector, do the following:

- Enter **homeCh** for the instance name.
- Enter **140** for the width (W) value.
- Enter " **Own your home**?" for the label parameter.

4 Drag two RadioButton components from the Components panel to the Stage and place them below and to the right of the CheckBox. Enter the following values for them in the Property inspector:

- Enter **underRb** and **overRb** for the instance names.
- Enter **120** for the W (width) parameter of both RadioButtons.
- Enter **Under $500,000?** for the label parameter of `underRb`.
- Enter **Over $500,000?** for the label parameter of `overRb`.
- Enter **valueGrp** for the groupName parameter for both RadioButtons.

5 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
homeCh.addEventListener(MouseEvent.CLICK, clickHandler);
underRb.enabled = false;
overRb.enabled = false;

function clickHandler(event:MouseEvent):void {
    underRb.enabled = event.target.selected;
    overRb.enabled = event.target.selected;
}
```

This code creates an event handler for a `CLICK` event that enables the `underRb` and `overRb` RadioButtons if the `homeCh` CheckBox is selected, and disables them if `homeCh` is not selected. For more information, see the MouseEvent class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

6 Select Control > Test Movie.

The following example duplicates the preceding application but creates the CheckBox and RadioButtons with ActionScript.

Create a Checkbox using ActionScript

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the CheckBox component and the RadioButton component from the Components panel to the current document's Library panel. If the Library panel is not open, press Ctrl+L or select Window > Library to open the Library panel.

This makes the components available to your application but does not put them on the Stage.

3 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following code to create and position the component instances:

```
import fl.controls.CheckBox;
import fl.controls.RadioButton;

var homeCh:CheckBox = new CheckBox();
var underRb:RadioButton = new RadioButton();
var overRb:RadioButton = new RadioButton();
addChild(homeCh);
addChild(underRb);
addChild(overRb);
underRb.groupName = "valueGrp";
overRb.groupName = "valueGrp";
homeCh.move(200, 100);
homeCh.width = 120;
homeCh.label = "Own your home?";
underRb.move(220, 130);
underRb.enabled = false;
underRb.width = 120;
underRb.label = "Under $500,000?";
overRb.move(220, 150);
overRb.enabled = false;
overRb.width = 120;
overRb.label = "Over $500,000?";
```

This code uses the `CheckBox()` and `RadioButton()` constructors to create the components and the `addChild()` method to place them on the Stage. It uses the `move()` method to position the components on the Stage.

4 Now, add the following ActionScript to create an event listener and an event handler function:

```
homeCh.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:MouseEvent):void {
    underRb.enabled = event.target.selected;
    overRb.enabled = event.target.selected;
}
```

This code creates an event handler for the `CLICK` event that enables the `underRb` and `overRb` radio buttons if the `homeCh` CheckBox is selected, and disables them if `homeCh` is not selected. For more information, see the MouseEvent class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

5 Select Control > Test Movie.

Use the ColorPicker component

The ColorPicker component allows a user to select a color from a swatch list. The default mode of the ColorPicker shows a single color in a square button. When a user clicks the button, the list of available colors appears in a swatch panel along with a text field that displays the hexadecimal value of the current color selection.

You can set the colors that appear in the ColoPicker by setting its `colors` property with the color values that you want to display.

User interaction with the ColorPicker component

A ColorPicker allows a user to select a color and apply it to another object in the application. For example, if you want to allow the user to personalize elements of the application, such as a background color or the color of text, you can include a ColorPicker and apply the color that the user selects.

A user chooses a color by clicking its swatch in the panel or by entering its hexadecimal value in the text field. Once the user selects a color, you can use the ColorPicker's `selectedColor` property to apply the color to text or another object in the application.

A ColorPicker instance receives focus if a user moves the pointer over it or tabs to it. When a ColorPicker's swatch panel is open, you can use the following keys to control it:

Key	**Description**
Home	Moves the selection to the first color in the swatch panel.
Up Arrow	Moves the selection up one row in the swatch panel.
Down Arrow	Moves the selection down one row in the swatch panel.
Right Arrow	Moves the selection in the swatch panel one color to the right.
Left Arrow	Moves the selection in the swatch panel one color to the left.
End	Moves the selection to the last color in the swatch panel.

ColorPicker component parameters

You can set the following authoring parameters in the Property inspector or in the Component inspector for each ColorPicker instance: `selectedColor` and `showTextField`. Each of these parameters has a corresponding ActionScript property of the same name. For information on the possible values for these parameters, see the ColorPicker class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Create an application with the ColorPicker component

The following example adds a ColorPicker component to an application while authoring. In this example, each time you change the color in the ColorPicker, the `changeHandler()` function calls the `drawBox()` function to draw a new box with the color you selected in the ColorPicker.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag a ColorPicker from the Components panel to the center of the Stage and give it an instance name of **aCp**.

3 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.events.ColorPickerEvent;

var aBox:MovieClip = new MovieClip();
drawBox(aBox, 0xFF0000);//draw a red box
addChild(aBox);

aCp.addEventListener(ColorPickerEvent.CHANGE,changeHandler);

function changeHandler(event:ColorPickerEvent):void {
    drawBox(aBox, event.target.selectedColor);
}

function drawBox(box:MovieClip,color:uint):void {
            box.graphics.beginFill(color, 1);
            box.graphics.drawRect(100, 150, 100, 100);
            box.graphics.endFill();
}
```

4 Select Control > Test Movie.

5 Click the ColorPicker and select a color to color the box.

Create a ColorPicker using ActionScript

This example uses the `ColorPicker()` constructor and `addChild()` to create a ColorPicker on the Stage. It sets the `colors` property to the color values for red (0xFF0000), green (0x00FF00), and blue (0x0000FF) to specify the colors that the ColorPicker will display. It also creates a TextArea and each time you select a different color in the ColorPicker, the example changes the color of the text in the TextArea to match.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the ColorPicker component from the Components panel to the Library panel.

3 Drag the TextArea component from the Components panel to the Library panel.

4 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.controls.ColorPicker;
import fl.controls.TextArea;
import fl.events.ColorPickerEvent;

var aCp:ColorPicker = new ColorPicker();
var aTa:TextArea = new TextArea();
var aTf:TextFormat = new TextFormat();

aCp.move(100, 100);
aCp.colors = [0xff0000, 0x00ff00, 0x0000ff];
aCp.addEventListener(ColorPickerEvent.CHANGE, changeHandler);

aTa.text = "Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vivamus quis nisl vel
tortor nonummy vulputate. Quisque sit amet eros sed purus euismod tempor. Morbi tempor. Class
aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Curabitur
diam. Suspendisse at purus in ipsum volutpat viverra. Nulla pellentesque libero id libero.";
aTa.setSize(200, 200);
aTa.move(200,100);

addChild(aCp);
addChild(aTa);

function changeHandler(event:ColorPickerEvent):void {
    if(TextFormat(aTa.getStyle("textFormat"))){
        aTf = TextFormat(aTa.getStyle("textFormat"));
    }
    aTf.color = event.target.selectedColor;
    aTa.setStyle("textFormat", aTf);
}
```

5 Select Control > Test Movie.

Use the ComboBox component

A ComboBox component allows a user to make a single selection from a drop-down list. A ComboBox can be static or editable. An editable ComboBox allows a user to enter text directly into the text field at the top of the list. If the drop-down list hits the bottom of the document, it opens up instead of down. The ComboBox is made up of three subcomponents: the BaseButton, TextInput, and List components.

In an editable ComboBox, only the button is the hit area—not the text box. For a static ComboBox, the button and the text box constitute the hit area. The hit area responds by opening or closing the drop-down list.

When the user makes a selection in the list, either with the mouse or through the keyboard, the label of the selection is copied to the text field at the top of the ComboBox.

User interaction with the ComboBox component

You can use a ComboBox component in any form or application that requires a single choice from a list. For example, you could provide a drop-down list of states in a customer address form. You can use an editable ComboBox for more complex scenarios. For example, in an application that provides driving directions, you could use an editable ComboBox, to allow a user to enter her origin and destination addresses. The drop-down list would contain her previously entered addresses.

If the ComboBox is editable, meaning the `editable` property is `true`, the following keys remove focus from the text input box and leave the previous value. The exception is the Enter key, which applies the new value first, if the user entered text.

Key	**Description**
Shift + Tab	Moves focus to the previous item. If a new item is selected, a `change` event is dispatched.
Tab	Moves focus to the next item. If a new item is selected, a `change` event is dispatched.
Down Arrow	Moves the selection down one item.
End	Moves the selection to the bottom of the list.
Escape	Closes the drop-down list and returns focus to the ComboBox.
Enter	Closes the drop-down list and returns focus to the ComboBox. When the ComboBox is editable and the user enters text, Enter sets the value to the entered text.
Home	Moves the selection to the top of the list.
Page Up	Moves the selection up one page.
Page Down	Moves the selection down one page.

When you add the ComboBox component to an application, you can make it accessible to a screen reader by adding the following lines of ActionScript code:

```
import fl.accessibility.ComboBoxAccImpl;

ComboBoxAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances you have of the component.

ComboBox component parameters

You can set the following parameters in the Property inspector or in the Component inspector for each ComboBox instance: `dataProvider`, `editable`, `prompt`, and `rowCount`. Each of these parameters has a corresponding ActionScript property of the same name. For information on the possible values for these parameters, see the ComboBox class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*. For information on using the dataProvider parameter, see “Use the dataProvider parameter” on page 28.

Create an application with the ComboBox component

The following procedure describes how to add a ComboBox component to an application while authoring. The ComboBox is editable and if you type **Add** into the text field, the example adds an item to the drop-down list.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag a ComboBox to the Stage and give it an instance name of **aCb**. On the Parameters tab, set the `editable` parameter to `true`**.**

3 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following code:

```
import fl.data.DataProvider;
import fl.events.ComponentEvent;

var items:Array = [
{label:"screen1", data:"screenData1"},
{label:"screen2", data:"screenData2"},
{label:"screen3", data:"screenData3"},
{label:"screen4", data:"screenData4"},
{label:"screen5", data:"screenData5"},
];
aCb.dataProvider = new DataProvider(items);

aCb.addEventListener(ComponentEvent.ENTER, onAddItem);

function onAddItem(event:ComponentEvent):void {
    var newRow:int = 0;
    if (event.target.text == "Add") {
        newRow = event.target.length + 1;
            event.target.addItemAt({label:"screen" + newRow, data:"screenData" + newRow},
            event.target.length);
    }
}
```

4 Select Control > Test Movie.

Create a ComboBox using ActionScript

The following example creates a ComboBox with ActionScript and populates it with a list of universities in the San Francisco, California, area. It sets the ComboBox's `width` property to accommodate the width of the prompt text and sets the `dropdownWidth` property to be slightly wider to accommodate the longest university name.

The example creates the list of universities in an Array instance, using the `label` property to store the school names and the `data` property to store the URLs of each school's website. It assigns the Array to the ComboBox by setting its `dataProvider` property.

When a user selects a university from the list, it triggers an Event.`CHANGE` event and a call to the `changeHandler()` function, which loads the `data` property into a URL request to access the school's website.

Notice that the last line sets the ComboBox instance's `selectedIndex` property to -1 to redisplay the prompt when the list closes. Otherwise, the prompt would be replaced by the name of the school that was selected.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the ComboBox component from the Components panel to the Library panel.

3 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.controls.ComboBox;
import fl.data.DataProvider;
import flash.net.navigateToURL;

var sfUniversities:Array = new Array(
    {label:"University of California, Berkeley",
                data:"http://www.berkeley.edu/"},
    {label:"University of San Francisco",
                data:"http://www.usfca.edu/"},
    {label:"San Francisco State University",
                data:"http://www.sfsu.edu/"},
    {label:"California State University, East Bay",
                data:"http://www.csuhayward.edu/"},
    {label:"Stanford University", data:"http://www.stanford.edu/"},
    {label:"University of Santa Clara", data:"http://www.scu.edu/"},
    {label:"San Jose State University", data:"http://www.sjsu.edu/"}
);

var aCb:ComboBox = new ComboBox();
aCb.dropdownWidth = 210;
aCb.width = 200;
aCb.move(150, 50);
aCb.prompt = "San Francisco Area Universities";
aCb.dataProvider = new DataProvider(sfUniversities);
aCb.addEventListener(Event.CHANGE, changeHandler);

addChild(aCb);

function changeHandler(event:Event):void {
    var request:URLRequest = new URLRequest();
    request.url = ComboBox(event.target).selectedItem.data;
    navigateToURL(request);
    aCb.selectedIndex = -1;
}
```

4 Select Control > Test Movie.

You can implement and run this example in the Flash authoring environment but you will receive warning messages if you attempt to access the university web sites by clicking items in the ComboBox. To access the fully functional ComboBox on the Internet, access the the following location:

http://www.helpexamples.com/peter/bayAreaColleges/bayAreaColleges.html

Use the DataGrid component

The DataGrid component lets you display data in a grid of rows and columns, drawing the data from an array or an external XML file that you can parse into an array for the DataProvider. The DataGrid component includes vertical and horizontal scrolling, event support (including support for editable cells), and sorting capabilities.

You can resize and customize characteristics such as the font, color, and the borders of columns in a grid. You can use a custom movie clip as a cell renderer for any column in a grid. (A cell renderer displays the contents of a cell.) You can turn off scroll bars and use the DataGrid methods to create a page view style display. For more information about customization, see the DataGridColumn class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

More Help topics

Creating, populating, and resizing the DataGrid component

Customizing and sorting the DataGrid component

Filtering and formatting data in the DataGrid component

User interaction with the DataGrid component

You can use the mouse and the keyboard to interact with a DataGrid component.

If the `sortableColumns` property and the column's `sortable` property are both `true`, clicking in a column header sorts the data based on the column's values. You can disable sorting for an individual column by setting its `sortable` property to `false`.

If the `resizableColumns` property is `true`, you can resize columns by dragging the column dividers in the header row.

Clicking in an editable cell gives focus to that cell; clicking a noneditable cell has no effect on focus. An individual cell is editable when both the `DataGrid.editable` and `DataGridColumn.editable` properties of the cell are `true`.

For more information, see the DataGrid and DataGridColumn classes in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

When a DataGrid instance has focus either from clicking or tabbing, you can use the following keys to control it:

Key	**Description**
Down Arrow	When a cell is being edited, the insertion point shifts to the end of the cell's text. If a cell is not editable, the Down Arrow key handles selection as the List component does.
Up Arrow	When a cell is being edited, the insertion point shifts to the beginning of the cell's text. If a cell is not editable, the Up Arrow key handles selection as the List component does.
Shift+Up/Down Arrow	If the DataGrid is not editable and `allowMultipleSelection` is `true`, selects contiguous rows. Reversing direction with the opposite arrow deselects selected rows until you pass the starting row, at which point rows in that direction are selected.
Shift+Click	If `allowMultipleSelection` is `true`, selects all rows between selected row and current caret position (highlighted cell).
Ctrl+Click	If `allowMultipleSelection` is `true`, selects additional rows, which do not need to be contiguous.
Right Arrow	When a cell is being edited, the insertion point shifts one character to the right. If a cell is not editable, the Right Arrow key does nothing.
Left Arrow	When a cell is being edited, the insertion point shifts one character to the left. If a cell is not editable, the Left Arrow key does nothing.
Home	Selects the first row in the DataGrid.
End	Selects the last row in the DataGrid.
PageUp	Selects the first row in a page of the DataGrid. A page consists of the number of rows that the DataGrid can display without scrolling.

Key	**Description**
PageDown	Selects the last row in a page of the DataGrid. A page consists of the number of rows that the DataGrid can display without scrolling.
Return/Enter/Shift+Enter	When a cell is editable, the change is committed, and the insertion point is moved to the cell on the same column, next row (up or down, depending on the shift toggle).
Shift+Tab/Tab	If the DataGrid is editable, moves focus to the previous/next item until the end of the column is reached and then to the previous/next row until the first or last cell is reached. If the first cell is selected, Shift+Tab moves focus to the preceding control. If the last cell is selected, Tab moves focus to the next control. If the DataGrid is not editable, moves focus to the previous/next control.

You can use the DataGrid component as the foundation for numerous types of data-driven applications. You can easily display a formatted tabular view of data, but you can also use the cell renderer capabilities to build more sophisticated and editable user interface pieces. The following are practical uses for the DataGrid component:

- A webmail client
- Search results pages
- Spreadsheet applications such as loan calculators and tax form applications

When you design an application with the DataGrid component, it is helpful to understand the design of the List component because the DataGrid class extends the SelectableList class. For more information on the SelectableList class and the List component, see the SelectableList and List classes in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

When you add a DataGrid component to your application, you can make it accessible to a screen reader by adding the following lines of ActionScript code:

```
import fl.accessibility.DataGridAccImpl;
DataGridAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances the component has. For more information, see Chapter 18, “Creating Accessible Content,” in *Using Flash*.

DataGrid component parameters

You can set the following authoring parameters in the Property inspector or in the Component inspector for each DataGrid component instance: `allowMultipleSelection`, `editable`, `headerHeight`, `horizontalLineScrollSize`, `horizontalPageScrollSize`, `horizontalScrolllPolicy`, `resizableColumns`, `rowHeight`, `showHeaders`, `verticalLineScrollSize`, `verticalPageScrollSize`, and `verticalScrollPolicy`. Each of these parameters has a corresponding ActionScript property of the same name. For information on the possible values for these parameters, see the DataGrid class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Create an application with the DataGrid component

To create an application with the DataGrid component, you must first determine where your data is coming from. Typically, data comes from an Array, which you can pull into the grid by setting the `dataProvider` property. You can also use the methods of the DataGrid and DataGridColumn classes to add data to the grid.

Use a local data provider with a DataGrid component:

This example creates a DataGrid to display a softball team's roster. It defines the roster in an Array (`aRoster`) and assigns it to the DataGrid's `dataProvider` property.

1 In Flash, select File > New, and then select Flash File (ActionScript 3.0).

2 Drag the DataGrid component from the Components panel to the Stage.

3 In the Property inspector, enter the instance name **aDg**.

4 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.data.DataProvider;

bldRosterGrid(aDg);
var aRoster:Array = new Array();
aRoster = [
        {Name:"Wilma Carter", Bats:"R", Throws:"R", Year:"So", Home: "Redlands, CA"},
        {Name:"Sue Pennypacker", Bats:"L", Throws:"R", Year:"Fr", Home: "Athens, GA"},
        {Name:"Jill Smithfield", Bats:"R", Throws:"L", Year:"Sr", Home: "Spokane, WA"},
        {Name:"Shirley Goth", Bats:"R", Throws:"R", Year:"Sr", Home: "Carson, NV"},
        {Name:"Jennifer Dunbar", Bats:"R", Throws:"R", Year:"Fr", Home: "Seaside, CA"},
        {Name:"Patty Crawford", Bats:"L", Throws:"L", Year:"Jr", Home: "Whittier, CA"},
        {Name:"Angelina Davis", Bats:"R", Throws:"R", Year:"So", Home: "Odessa, TX"},
        {Name:"Maria Santiago", Bats:"L", Throws:"L", Year:"Sr", Home: "Tacoma, WA"},
        {Name:"Debbie Ferguson", Bats:"R", Throws:"R", Year: "Jr", Home: "Bend, OR"},
        {Name:"Karen Bronson", Bats:"R", Throws:"R", Year: "Sr", Home: "Billings, MO"},
        {Name:"Sylvia Munson", Bats:"R", Throws:"R", Year: "Jr", Home: "Pasadena, CA"},
        {Name:"Carla Gomez", Bats:"R", Throws:"L", Year: "Sr", Home: "Corona, CA"},
        {Name:"Betty Kay", Bats:"R", Throws:"R", Year: "Fr", Home: "Palo Alto, CA"},
];
aDg.dataProvider = new DataProvider(aRoster);
aDg.rowCount = aDg.length;

function bldRosterGrid(dg:DataGrid){
    dg.setSize(400, 300);
    dg.columns = ["Name", "Bats", "Throws", "Year", "Home"];
    dg.columns[0].width = 120;
    dg.columns[1].width = 50;
    dg.columns[2].width = 50;
    dg.columns[3].width = 40;
    dg.columns[4].width = 120;
    dg.move(50,50);
};
```

The `bldRosterGrid()` function sets the size of the DataGrid and sets the order of the columns and their sizes.

5 Select Control > Test Movie.

Specify columns and add sorting for a DataGrid component in an application

Notice that you can click any column heading to sort the DataGrid's content in descending order by that column's values.

The following example uses the `addColumn()` method to add DataGridColumn instances to a DataGrid. The columns represent player names and their scores. The example also sets the `sortOptions` property to specify the sort options for each column: `Array.CASEINSENSITIVE` for the Name column and `Array.NUMERIC` for the Score column. It sizes the DataGrid appropriately by setting the length to the number of rows and the width to 200.

1 In Flash, select File > New, and then select Flash File (ActionScript 3.0).

2 Drag the DataGrid component from the Components panel to the Stage.

3 In the Property inspector, enter the instance name **aDg**.

4 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.controls.dataGridClasses.DataGridColumn;
import fl.events.DataGridEvent;
import fl.data.DataProvider;
// Create columns to enable sorting of data.
var nameDGC:DataGridColumn = new DataGridColumn("name");
nameDGC.sortOptions = Array.CASEINSENSITIVE;
var scoreDGC:DataGridColumn = new DataGridColumn("score");
scoreDGC.sortOptions = Array.NUMERIC;
aDg.addColumn(nameDGC);
aDg.addColumn(scoreDGC);
var aDP_array:Array = new Array({name:"clark", score:3135}, {name:"Bruce", score:403},
{name:"Peter", score:25})
aDg.dataProvider = new DataProvider(aDP_array);
aDg.rowCount = aDg.length;
aDg.width = 200;
```

5 Select Control > Test Movie.

Create a DataGrid component instance using ActionScript

This example creates a DataGrid using ActionScript and populates it with an Array of player names and scores.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the DataGrid component from the Components panel to the current document’s Library panel.

This adds the component to the library but doesn’t make it visible in the application.

3 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.controls.DataGrid;
import fl.data.DataProvider;

var aDg:DataGrid = new DataGrid();
addChild(aDg);
aDg.columns = [ "Name", "Score" ];
aDg.setSize(140, 100);
aDg.move(10, 40);
```

This code creates the DataGrid instance and then sizes and positions the grid.

4 Create an array, add data to the array, and identify the array as the data provider for the DataGrid:

```
var aDP_array:Array = new Array();
aDP_array.push({Name:"Clark", Score:3135});
aDP_array.push({Name:"Bruce", Score:403});
aDP_array.push({Name:"Peter", Score:25});
aDg.dataProvider = new DataProvider(aDP_array);
aDg.rowCount = aDg.length;
```

5 Select Control > Test Movie.

Load a DataGrid with an XML file

The following example uses the DataGridColumn class to create the DataGrid's columns. It populates the DataGrid by passing an XML object as the `value` parameter of the `DataProvider()` constructor.

1 Using a text editor create an XML file with the following data and save it as `team.xml` in the same folder where you will save the FLA file.

```
<team>
    <player name="Player A" avg="0.293" />
    <player name="Player B" avg="0.214" />
    <player name="Player C" avg="0.317" />
</team>
```

2 Create a new Flash (ActionScript 3.0) document.

3 In the Components panel, double-click the DataGrid component to add it to the Stage.

4 In the Property inspector, enter the instance name **aDg**.

5 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.controls.dataGridClasses.DataGridColumn;
import fl.data.DataProvider;
import flash.net.*;
import flash.events.*;

var request:URLRequest = new URLRequest("team.xml");
var loader:URLLoader = new URLLoader;


loader.load(request);
loader.addEventListener(Event.COMPLETE, loaderCompleteHandler);

function loaderCompleteHandler(event:Event):void {

    var teamXML:XML = new XML(loader.data);

    var nameCol:DataGridColumn = new DataGridColumn("name");
    nameCol.headerText = "Name";
    nameCol.width = 120;
    var avgCol:DataGridColumn = new DataGridColumn("avg");
    avgCol.headerText = "Average";
    avgCol.width = 60;

    var myDP:DataProvider = new DataProvider(teamXML);

    aDg.columns = [nameCol, avgCol];
    aDg.width = 200;
    aDg.dataProvider = myDP;
    aDg.rowCount = aDg.length;
}
```

6 Select Control > Test Movie.

Use the Label component

The Label component displays a single line of text, typically to identify some other element or activity on a web page. You can specify that a label be formatted with HTML to take advantage of its text formatting tags. You can also control the alignment and size of a label. Label components don't have borders, cannot be focused, and don't broadcast any events.

A live preview of each Label instance reflects changes made to parameters in the Property inspector or Component inspector during authoring. The label doesn't have a border, so the only way to see its live preview is to set its text parameter.

User interaction with the Label component

Use a Label component to create a text label for another component in a form, such as a "Name:" label to the left of a TextInput field that accepts a user's name. It's a good idea to use a Label component instead of a plain text field because you can use styles to maintain a consistent look and feel.

If you want to rotate a Label component, you must embed the fonts; otherwise they won't show when you test the movie.

Label component parameters

You can set the following authoring parameters in the Property inspector or in the Component inspector for each Label component instance: `autoSize`, `condenseWhite`, `selectable`, `text`, and `wordWrap`. Each of these parameters has a corresponding ActionScript property of the same name. For information on the possible values for these parameters, see the Label class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Create an application with the Label component

The following procedure explains how to add a Label component to an application while authoring. In this example, the label simply displays the text "Expiration Date."

1 Create a new Flash (ActionScript 3.0) document.

2 Drag a Label component from the Components panel to the Stage and give it the following values in the Property inspector:

- Enter **aLabel** for the instance name.
- Enter **80** for the W value.
- Enter **100** for the X value.
- Enter **100** for the Y value.
- Enter **Expiration Date** for the `text` parameter.

3 Drag a TextArea component to the Stage and give it the following values in the Property inspector:

- Enter **aTa** for the instance name.
- Enter **22** for the H value.
- Enter **200** for the X value.
- Enter **100** for the Y value.

4 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
var today:Date = new Date();
var expDate:Date = addDays(today, 14);
aTa.text = expDate.toDateString();

function addDays(date:Date, days:Number):Date {
return addHours(date, days*24);
}

function addHours(date:Date, hrs:Number):Date {
return addMinutes(date, hrs*60);
}

function addMinutes(date:Date, mins:Number):Date {
return addSeconds(date, mins*60);
}

function addSeconds(date:Date, secs:Number):Date {
var mSecs:Number = secs * 1000;
var sum:Number = mSecs + date.getTime();
return new Date(sum);
}
```

5 Select Control > Test Movie.

Create a Label component instance using ActionScript

The following example creates a Label parameter using ActionScript. It uses a Label to identify the function of a ColorPicker component and it uses the `htmlText` property to apply formatting to the Label's text.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the Label component from the Components panel to the current document's Library panel.

3 Drag the ColorPicker component from the Components panel to the current document's Library panel.

4 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.controls.Label;
import fl.controls.ColorPicker;

var aLabel:Label = new Label();
var aCp:ColorPicker = new ColorPicker();

addChild(aLabel);
addChild(aCp);

aLabel.htmlText = '<font face="Arial" color="#FF0000" size="14">Fill:</font>';
aLabel.x = 200;
aLabel.y = 150;
aLabel.width = 25;
aLabel.height = 22;

aCp.x = 230;
aCp.y = 150;
```

5 Select Control > Test Movie.

Use the List component

The List component is a scrollable single- or multiple-selection list box. A list can also display graphics, including other components. You add the items displayed in the list by using the Values dialog box that appears when you click in the labels or data parameter fields. You can also use the `List.addItem()` and `List.addItemAt()` methods to add items to the list.

The List component uses a zero-based index, where the item with index 0 is the top item displayed. When adding, removing, or replacing list items using the List class methods and properties, you may need to specify the index of the list item.

User interaction with the List component

You can set up a list so that users can make either single or multiple selections. For example, a user visiting an e-commerce website needs to select which item to buy. There are 30 items in a list that the user scrolls through and selects an item by clicking it.

You can also design a List that uses custom movie clips as rows so you can display more information to the user. For example, in an e-mail application, each mailbox could be a List component and each row could have icons to indicate priority and status.

The List receives focus when you click it or tab to it, and you can then use the following keys to control it:

Key	Description
Alphanumeric keys	Jump to the next item that has `Key.getAscii()` as the first character in its label.
Control	Toggle key that allows multiple noncontiguous selections and deselections.
Down Arrow	Selection moves down one item.
Home	Selection moves to the top of the list.
Page Down	Selection moves down one page.
Page Up	Selection moves up one page.
Shift	Allows for contiguous selection.
Up Arrow	Selection moves up one item.

Note: *Note that scroll sizes are in pixels and not rows.*

Note: *The page size used by the Page Up and Page Down keys is one less than the number of items that fit in the display. For example, paging down through a ten-line drop-down list shows items 0-9, 9-18, 18-27, and so on, with one item overlapping per page.*

For more information about controlling focus, see the IFocusManager interface and the FocusManager class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*, and “Work with FocusManager” on page 26.

A live preview of each List instance on the Stage reflects changes made to parameters in the Property inspector or Component inspector during authoring.

When you add the List component to an application, you can make it accessible to a screen reader by adding the following lines of ActionScript code:

```
import fl.accessibility.ListAccImpl;

ListAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances the component has. For more information, see Chapter 18, "Creating Accessible Content," in *Using Flash*.

List component parameters

You can set the following parameters in the Property inspector or in the Component inspector for each List component instance: `allowMultipleSelection`, `dataProvider, horizontalLineScrollSize, horizontalPageScrollSize, horizontalScrollPolicy, multipleSelection, verticalLineScrollSize, verticalPageScrollSize`, and `verticalScrollPolicy`. Each of these parameters has a corresponding ActionScript property of the same name. For information on the possible values for these parameters, see the List class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*. For information on using the dataProvider parameter, see "Use the dataProvider parameter" on page 28.

Create an application with the List component

The following examples describe how to add a List component to an application while authoring.

Add a simple List component to an application

In this example, the List consists of labels that identify car models and data fields that contain prices.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag a List component from the Components panel to the Stage.

3 In the Property inspector, do the following:

- Enter the instance name **aList**.
- Assign a value of **200** to the W (width).

4 Use the Text tool to create a text field below `aList` and give it an instance name of **aTf**.

5 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.controls.List;
import flash.text.TextField;

aTf.type = TextFieldType.DYNAMIC;
aTf.border = false;

// Create these items in the Property inspector when data and label
// parameters are available.
aList.addItem({label:"1956 Chevy (Cherry Red)", data:35000});
aList.addItem({label:"1966 Mustang (Classic)", data:27000});
aList.addItem({label:"1976 Volvo (Xcllnt Cond)", data:17000});
aList.allowMultipleSelection = true;

aList.addEventListener(Event.CHANGE, showData);

function showData(event:Event) {
    aTf.text = "This car is priced at: $" + event.target.selectedItem.data;
}
```

This code uses the `addItem()` method to populate `aList` with three items, assigning each one a `label` value, which appears in the list, and a `data` value. When you select an item in the List, the event listener calls the `showData()` function, which displays the `data` value for the selected item.

6 Select Control > Test Movie to compile and run this application.

Populate a List instance with a data provider

This example creates a List of car models and their prices. It uses a data provider to populate the List, however, rather than the `addItem()` method.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag a List component from the Components panel to the Stage.

3 In the Property inspector, do the following:

- Enter the instance name **aList**.
- Assign a value of **200** to the W (width).

4 Use the Text tool to create a text field below `aList` and give it an instance name of **aTf**.

5 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.controls.List;
import fl.data.DataProvider;
import flash.text.TextField;

aTf.type = TextFieldType.DYNAMIC;
aTf.border = false;

var cars:Array = [
{label:"1956 Chevy (Cherry Red)", data:35000},
{label:"1966 Mustang (Classic)", data:27000},
{label:"1976 Volvo (Xcllnt Cond)", data:17000},
];
aList.dataProvider = new DataProvider(cars);
aList.allowMultipleSelection = true;

aList.addEventListener(Event.CHANGE, showData);

function showData(event:Event) {
    aTf.text = "This car is priced at: $" + event.target.selectedItem.data;
}
```

6 Select Control > Test Movie to see the List with its items.

Use a List component to control a MovieClip instance

The following example creates a List of color names and when you select one, it applies the color to a MovieClip.

1 Create a Flash (ActionScript 3.0) document.

2 Drag the List component from the Components panel to the Stage and give it the following values in the Property inspector:

- Enter **aList** for the instance name.
- Enter **60** for the H value.
- Enter **100** for the X value.
- Enter **150** for the Y value.

3 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
aList.addItem({label:"Blue", data:0x0000CC});
aList.addItem({label:"Green", data:0x00CC00});
aList.addItem({label:"Yellow", data:0xFFFF00});
aList.addItem({label:"Orange", data:0xFF6600});
aList.addItem({label:"Black", data:0x000000});

var aBox:MovieClip = new MovieClip();
addChild(aBox);

aList.addEventListener(Event.CHANGE, changeHandler);
function changeHandler(event:Event) {
    drawBox(aBox, event.target.selectedItem.data);
};

function drawBox(box:MovieClip,color:uint):void {
            box.graphics.beginFill(color, 1.0);
            box.graphics.drawRect(225, 150, 100, 100);
            box.graphics.endFill();
}
```

4 Select Control > Test Movie to run the application.

5 Click colors in the List to see them displayed in a MovieClip.

Create a List component instance using ActionScript:

This example creates a simple list using ActionScript, and populates it using the `addItem()` method.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the List component from the Components panel to the Library panel.

3 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.controls.List;

var aList:List = new List();
aList.addItem({label:"One", data:1});
aList.addItem({label:"Two", data:2});
aList.addItem({label:"Three", data:3});
aList.addItem({label:"Four", data:4});
aList.addItem({label:"Five", data:5});
aList.setSize(60, 40);
aList.move(200,200);
addChild(aList);
aList.addEventListener(Event.CHANGE, changeHandler);
function changeHandler(event:Event):void {
    trace(event.target.selectedItem.data);
}
```

4 Select Control > Test Movie to run the application.

Use the NumericStepper component

The NumericStepper component allows a user to step through an ordered set of numbers. The component consists of a number in a text box displayed beside small up and down arrow buttons. When a user presses the buttons, the number is raised or lowered incrementally according to the unit specified in the `stepSize` parameter until the user releases the buttons or until the maximum or minimum value is reached. The text in the NumericStepper component's text box is also editable.

A live preview of each NumericStepper instance reflects the setting of the value parameter in the Property inspector or Component inspector. However, there is no mouse or keyboard interaction with the NumericStepper's arrow buttons in the live preview.

User interaction with the NumericStepper component

You can use the NumericStepper component anywhere you want a user to select a numeric value. For example, you could use a NumericStepper component in a form to set the month, day, and year of a credit card expiration date. You could also use a NumericStepper component to allow a user to increase or decrease a font size.

The NumericStepper component handles only numeric data. Also, you must resize the stepper while authoring to display more than two numeric places (for example, the numbers 5246 or 1.34).

You can enable or disable a NumericStepper in an application. In the disabled state, a NumericStepper doesn't receive mouse or keyboard input. When it's enabled, the NumericStepper receives focus if you click it or tab to it, and its internal focus is set to the text box. When a NumericStepper instance has focus, you can use the following keys to control it:

Key	**Description**
Down Arrow	Value changes by one unit.
Left Arrow	Moves the insertion point to the left within the text box.
Right Arrow	Moves the insertion point to the right within the text box.
Shift+Tab	Moves focus to the previous object.
Tab	Moves focus to the next object.
Up Arrow	Value changes by one unit.

For more information about controlling focus, see the FocusManager class in the *ActionScript 3.0 Reference for the Adobe Flash Platform* and "Work with FocusManager" on page 26.

NumericStepper component parameters

You can set the following parameters in the Property inspector or in the Component inspector for each NumericStepper instance: `maximum`, `minimum`, `stepSize`, and `value`. Each of these parameters has a corresponding ActionScript property of the same name. For information on the possible values for these parameters, see the NumericStepper class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Create an application with the NumericStepper

The following procedure explains how to add a NumericStepper component to an application while authoring. The example places a NumericStepper component and a Label component on the Stage and creates a listener for an `Event.CHANGE` event on the NumericStepper instance. When the value in the numeric stepper changes, the example displays the new value in the `text` property of the Label instance.

1 Drag a NumericStepper component from the Components panel to the Stage.

2 In the Property inspector, enter the instance name **aNs**.

3 Drag a Label component from the Components panel to the Stage.

4 In the Property inspector, enter the instance name **aLabel**.

5 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import flash.events.Event;

aLabel.text = "value = " + aNs.value;

aNs.addEventListener(Event.CHANGE, changeHandler);
function changeHandler(event:Event) :void {
    aLabel.text = "value = " + event.target.value;
};
```

This example sets the `text` property of the label to the value of the NumericStepper. The `changeHandler()` function updates the label's `text` property whenever the value in the NumericStepper instance changes.

6 Select Control > Test Movie.

Create a NumericStepper using ActionScript:

This example creates three NumericSteppers with ActionScript code, one each for entering the month, day, and year of the user's date of birth. It also adds Labels for a prompt and for identifiers for each of the NumericSteppers.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag a Label to the Library panel.

3 Drag a NumericStepper component to the Library panel.

4 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.controls.Label;
import fl.controls.NumericStepper;

var dobPrompt:Label = new Label();
var moPrompt:Label = new Label();
var dayPrompt:Label = new Label();
var yrPrompt:Label = new Label();

var moNs:NumericStepper = new NumericStepper();
var dayNs:NumericStepper = new NumericStepper();
var yrNs:NumericStepper = new NumericStepper();

addChild(dobPrompt);
addChild(moPrompt);
addChild(dayPrompt);
addChild(yrPrompt);
addChild(moNs);
addChild(dayNs);
addChild(yrNs);

dobPrompt.setSize(65, 22);
dobPrompt.text = "Date of birth:";
dobPrompt.move(80, 150);

moNs.move(150, 150);
moNs.setSize(40, 22);
moNs.minimum = 1;
moNs.maximum = 12;
moNs.stepSize = 1;
moNs.value = 1;

moPrompt.setSize(25, 22);
moPrompt.text = "Mo.";
moPrompt.move(195, 150);

dayNs.move(225, 150);
dayNs.setSize(40, 22);
dayNs.minimum = 1;
dayNs.maximum = 31;
dayNs.stepSize = 1;
dayNs.value = 1;

dayPrompt.setSize(25, 22);
dayPrompt.text = "Day";
dayPrompt.move(270, 150);

yrNs.move(300, 150);
yrNs.setSize(55, 22);
yrNs.minimum = 1900;
yrNs.maximum = 2006;
yrNs.stepSize = 1;
yrNs.value = 1980;

yrPrompt.setSize(30, 22);
yrPrompt.text = "Year";
yrPrompt.move(360, 150);
```

5 Select Control > Test Movie to run the application.

Use the ProgressBar component

The ProgressBar component displays the progress of loading content, which is reassuring to a user when the content is large and can delay the execution of the application. The ProgressBar is useful for displaying the progress of loading images and pieces of an application. The loading process can be determinate or indeterminate. A *determinate* progress bar is a linear representation of a task's progress over time and is used when the amount of content to load is known. An *indeterminate* progress bar is used when the amount of content to load is unknown. You can also add a Label component to display the progress of loading as a percentage.

The ProgressBar component uses 9-slice scaling and has a bar skin, a track skin, and an indeterminate skin.

User interaction with the ProgressBar component

There are three modes in which to use the ProgressBar component. The most commonly used modes are the event mode and the polled mode. These modes specify a loading process that either emits `progress` and `complete` events (event and polled mode) or exposes `bytesLoaded` and `bytesTotal` properties (polled mode). You can also use the ProgressBar component in manual mode by setting the `maximum`, `minimum`, and `value` properties along with calls to the `ProgressBar.setProgress()` method. You can set the indeterminate property to indicate whether the ProgressBar has a striped fill and a source of unknown size (`true`) or a solid fill and a source of known size (`false`).

You set the ProgressBar's mode by setting its `mode` property, either through the `mode` parameter in the Property inspector or the Component inspector or by using ActionScript.

If you use the ProgressBar to show processing status, like parsing 100,000 items, if it is in a single frame loop there will be no visible updates to the ProgressBar because there are no redraws of the screen.

ProgressBar component parameters

You can set the following parameters in the Property inspector or in the Component inspector for each ProgressBar instance: `direction`, `mode`, and `source`. Each of these has a corresponding ActionScript property of the same name.

You can write ActionScript to control these and additional options for the ProgressBar component using its properties, methods, and events. For more information, see the ProgressBar class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Create an application with the ProgressBar

The following procedure shows you how to add a ProgressBar component to an application while authoring. In this example, the ProgressBar uses the event mode. In event mode, the loading content emits `progress` and `complete` events that the ProgressBar dispatches to indicate progress. When the `progress` event occurs, the example updates a label to show the percent of content that has loaded. When the `complete` event occurs, the example displays "Loading complete" and the value of the `bytesTotal` property, which is the size of the file.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the ProgressBar component from the Components panel to the Stage.

- In the Property inspector, enter the instance name **aPb**.
- In the Parameters section, Enter **200** for the X value.
- Enter **260** for the Y value.

- Select `event` for the `mode` parameter.

3 Drag the Button component from the Components panel to the Stage.

- In the Property inspector, enter **loadButton** as the instance name.
- Enter **220** for the X parameter.
- Enter **290** for the Y parameter.
- Enter **Load Sound** for the `label` parameter.

4 Drag the Label component to the Stage and give it an instance name of **progLabel**.

- Enter **150** for the W value.
- Enter **200** for the X parameter.
- Enter **230** for the Y parameter.
- In the Parameters section, clear the value for the `text` parameter.

5 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code, which loads an mp3 audio file:

```
import fl.controls.ProgressBar;
import flash.events.ProgressEvent;
import flash.events.IOErrorEvent;

var aSound:Sound = new Sound();
aPb.source = aSound;
var url:String = "http://www.helpexamples.com/flash/sound/song1.mp3";
var request:URLRequest = new URLRequest(url);

aPb.addEventListener(ProgressEvent.PROGRESS, progressHandler);
aPb.addEventListener(Event.COMPLETE, completeHandler);
aSound.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
loadButton.addEventListener(MouseEvent.CLICK, clickHandler);

function progressHandler(event:ProgressEvent):void {
        progLabel.text = ("Sound loading ... " + aPb.percentComplete);
}

function completeHandler(event:Event):void {
    trace("Loading complete");
    trace("Size of file: " + aSound.bytesTotal);
    aSound.close();
    loadButton.enabled = false;
}

function clickHandler(event:MouseEvent) {
    aSound.load(request);
}

function ioErrorHandler(event:IOErrorEvent):void {
    trace("Load failed due to: " + event.text);
}
```

6 Select Control > Test Movie.

Create an application with the ProgressBar component in polled mode

The following example sets the ProgressBar to polled mode. In polled mode, progress is determined by listening for `progress` events on the content that is loading and using its `bytesLoaded` and `bytesTotal` properties to calculate progress. This example loads a Sound object, listens for its `progress` events, and calculates the percent loaded using its `bytesLoaded` and `bytesTotal` properties. It displays the percent loaded in both a label and in the Output panel.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag a ProgressBar component from the Components panel to the Stage and enter the following values in the Property inspector:

- Enter **aPb** for the instance name.
- Enter **185** for the X value.
- Enter **225** for the Y value.

3 Drag the Label component to the Stage and enter the following values in the Property inspector:

- Enter **progLabel** for the instance name.
- Enter **180** for the X value.
- Enter **180** for the Y value.
- In the Parameters section, clear the value for the text parameter.

4 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code, which creates a Sound object (`aSound`) and calls `loadSound()` to load a sound into the Sound object:

```
import fl.controls.ProgressBarMode;
import flash.events.ProgressEvent;
import flash.media.Sound;

var aSound:Sound = new Sound();
var url:String = "http://www.helpexamples.com/flash/sound/song1.mp3";
var request:URLRequest = new URLRequest(url);

aPb.mode = ProgressBarMode.POLLED;
aPb.source = aSound;
aSound.addEventListener(ProgressEvent.PROGRESS, loadListener);

aSound.load(request);

function loadListener(event:ProgressEvent) {
    var percentLoaded:int = event.target.bytesLoaded / event.target.bytesTotal * 100;
    progLabel.text = "Percent loaded: " + percentLoaded + "%";
    trace("Percent loaded: " + percentLoaded + "%");
}
```

5 Select Control > Test Movie to run the application.

Create an application with the ProgressBar component in manual mode

The following example sets the ProgressBar to manual mode. In manual mode, you must set progress manually by calling the `setProgress()` method and provide it with the current and maximum values to determine the extent of progress. You do not set the `source` property in manual mode. The example uses a NumericStepper component, with a maximum value of 250, to increment the ProgressBar. When the value in the NumericStepper changes and triggers a `CHANGE` event, the event handler (`nsChangeHander`) calls the `setProgress()` method to advance the ProgressBar. It also displays the percent of progress completed, based on the maximum value.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the ProgressBar component from the Components panel to the Stage and give it the following values in the Property inspector:
   - Enter **aPb** for the instance name.
   - Enter **180** for the X value.
   - Enter **175** for the Y value.

3 Drag a NumericStepper component to the Stage and enter the following values in the Property inspector:
   - Enter **aNs** for the instance name.
   - Enter **220** for the X value.
   - Enter **215** for the Y value.
   - In the Parameters section, enter **250** for the maximum parameter, **0** for the minimum value, **1** for the `stepSize` parameter, and **0** for the `value` parameter.

4 Drag a Label component to the Stage and enter the following values in the Property inspector:
   - Enter **progLabel** for the instance name.
   - Enter **150** for the W value.
   - Enter **180** for the X value.
   - Enter **120** for the Y value.
   - In the Parameters tab, clear the value Label for the text parameter.

5 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following code:

```
import fl.controls.ProgressBarDirection;
import fl.controls.ProgressBarMode;
import flash.events.Event;

aPb.direction = ProgressBarDirection.RIGHT;
aPb.mode = ProgressBarMode.MANUAL;
aPb.minimum = aNs.minimum;
aPb.maximum = aNs.maximum;
aPb.indeterminate = false;

aNs.addEventListener(Event.CHANGE, nsChangeHandler);

function nsChangeHandler(event:Event):void {
    aPb.value = aNs.value;
    aPb.setProgress(aPb.value, aPb.maximum);
progLabel.text = "Percent of progress = " + int(aPb.percentComplete) + "%";
}
```

6 Select Control > Test Movie to run the application.

7 Click the Up Arrow on the NumericStepper to advance the ProgressBar.

Create a ProgressBar using ActionScript

This example creates a ProgressBar using ActionScript. Apart from that, it duplicates the functionality of the preceding example, which creates a ProgressBar in manual mode.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the ProgressBar component to the Library panel.

3 Drag the NumericStepper component to the Library panel.

4 Drag the Label component to the Library panel.

5 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following code:

```
import fl.controls.ProgressBar;
import fl.controls.NumericStepper;
import fl.controls.Label;
import fl.controls.ProgressBarDirection;
import fl.controls.ProgressBarMode;
import flash.events.Event;

var aPb:ProgressBar = new ProgressBar();
var aNs:NumericStepper = new NumericStepper();
var progLabel:Label = new Label();

addChild(aPb);
addChild(aNs);
addChild(progLabel);

aPb.move(180,175);
aPb.direction = ProgressBarDirection.RIGHT;
aPb.mode = ProgressBarMode.MANUAL;

progLabel.setSize(150, 22);
progLabel.move(180, 150);
progLabel.text = "";

aNs.move(220, 215);
aNs.maximum = 250;
aNs.minimum = 0;
aNs.stepSize = 1;
aNs.value = 0;

aNs.addEventListener(Event.CHANGE, nsChangeHandler);

function nsChangeHandler(event:Event):void {
aPb.setProgress(aNs.value, aNs.maximum);
progLabel.text = "Percent of progress = " + int(aPb.percentComplete) + "%";
}
```

6 Select Control > Test Movie to run the application.

7 Click the Up Arrow on the NumericStepper to advance the ProgressBar.

Use the RadioButton component

The RadioButton component lets you force a user to make a single choice within a set of choices. This component must be used in a group of at least two RadioButton instances. Only one member of the group can be selected at any given time. Selecting one radio button in a group deselects the currently selected radio button in the group. You set the `groupName` parameter to indicate which group a radio button belongs to.

A radio button is a fundamental part of many form applications on the web. You can use radio buttons wherever you want a user to make one choice from a group of options. For example, you would use radio buttons in a form to ask which credit card a customer wants to use.

User interaction with the RadioButton component

A radio button can be enabled or disabled. A disabled radio button doesn't receive mouse or keyboard input. When the user clicks or tabs into a RadioButton component group, only the selected radio button receives focus. The user can then use the following keys to control it:

Key	Description
Up Arrow/Left Arrow	The selection moves to the previous radio button within the radio button group.
Down Arrow/Right Arrow	The selection moves to the next radio button within the radio button group.
Tab	Moves focus from the radio button group to the next component.

For more information about controlling focus, see the IFocusManager interface and the FocusManager class in the *ActionScript 3.0 Reference for the Adobe Flash Platform* and "Work with FocusManager" on page 26.

A live preview of each RadioButton instance on the Stage reflects changes made to parameters in the Property inspector or Component inspector during authoring. However, the mutual exclusion of selection does not display in the live preview. If you set the selected parameter to `true` for two radio buttons in the same group, they both appear selected even though only the last instance created appears selected at run time. For more information, see "RadioButton component parameters" on page 74.

When you add the RadioButton component to an application, you can make it accessible to a screen reader by adding the following lines of code:

```
import fl.accessibility.RadioButtonAccImpl;
RadioButtonAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances you have of the component. For more information, see Chapter 18, "Creating Accessible Content," in Using Flash.

RadioButton component parameters

You can set the following authoring parameters in the Property inspector or in the Component inspector for each RadioButton component instance: `groupName, label, LabelPlacement, selected`, and `value`. Each of these parameters has a corresponding ActionScript property of the same name. For information on the possible values for these parameters, see the RadioButton class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

You can write ActionScript to set additional options for RadioButton instances using the methods, properties, and events of the RadioButton class.

Create an application with the RadioButton component

The following procedure explains how to add RadioButton components to an application while authoring. In this example, the RadioButtons are used to present a yes-or-no question. The data from the RadioButton is displayed in a TextArea.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag two RadioButton components from the Components panel to the Stage.

3 Select the first radio button. In the Property inspector, give it an instance name of **yesRb** and a group name of **rbGroup.**

4 Select the second radio button. In the Property inspector, give it an instance name of **noRb** and a group name of **rbGroup.**

5 Drag a TextArea component from the Components panel to the Stage and give it an instance name of **aTa**.

6 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
yesRb.label = "Yes";
yesRb.value = "For";
noRb.label = "No";
noRb.value = "Against";

yesRb.move(50, 100);
noRb.move(100, 100);
aTa.move(50, 30);
noRb.addEventListener(MouseEvent.CLICK, clickHandler);
yesRb.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:MouseEvent):void {
    aTa.text = event.target.value;
}
```

7 Select Control > Test Movie to run the application.

Create a RadioButton using ActionScript

This example uses ActionScript to create three RadioButtons for the colors red, blue, and green and draws a gray box. The `value` property for each RadioButton specifies the hexadecimal value for the color associated with the button. When a user clicks one of the RadioButtons, the `clickHandler()` function calls `drawBox()`, passing the color from the RadioButton's `value` property to color the box.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the RadioButton component to the Library panel.

3 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.controls.RadioButton;
import fl.controls.RadioButtonGroup;

var redRb:RadioButton = new RadioButton();
var blueRb:RadioButton = new RadioButton();
var greenRb:RadioButton = new RadioButton();
var rbGrp:RadioButtonGroup = new RadioButtonGroup("colorGrp");

var aBox:MovieClip = new MovieClip();
drawBox(aBox, 0xCCCCCC);

addChild(redRb);
addChild(blueRb);
addChild(greenRb);
addChild(aBox);

redRb.label = "Red";
redRb.value = 0xFF0000;
blueRb.label = "Blue";
blueRb.value = 0x0000FF;
greenRb.label = "Green";
greenRb.value = 0x00FF00;
redRb.group = blueRb.group = greenRb.group = rbGrp;
redRb.move(100, 260);
blueRb.move(150, 260);
greenRb.move(200, 260);

rbGrp.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:MouseEvent):void {
    drawBox(aBox, event.target.selection.value);
}

function drawBox(box:MovieClip,color:uint):void {
            box.graphics.beginFill(color, 1.0);
            box.graphics.drawRect(125, 150, 100, 100);
            box.graphics.endFill();
}
```

4 Select Control > Test Movie to run the application.

For more information, see the RadioButton class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Use the ScrollPane component

You can use the ScrollPane component to display content that is too large for the area into which it is loaded. For example, if you have a large image and only a small space for it in an application, you could load it into a ScrollPane. The ScrollPane can accept movie clips, JPEG, PNG, GIF, and SWF files.

Components such as the ScrollPane and the UILoader have `complete` events that allow you to determine when content has finished loading. If you want to set properties on the content of a ScrollPane or UILoader component, listen for the `complete` event and set the property in the event handler. For example, the following code creates a listener for the Event.`COMPLETE` event and an event handler that sets the `alpha` property of the ScrollPane's content to .5:

```
function spComplete(event:Event):void{
aSp.content.alpha = .5;
}
aSp.addEventListener(Event.COMPLETE, spComplete);
```

If you specify a location when loading content to the ScrollPane, you must specify the location (X and Y coordinates) as 0, 0. For example, the following code loads the ScrollPane properly because the box is drawn at location 0, 0:

```
var box:MovieClip = new MovieClip();
box.graphics.beginFill(0xFF0000, 1);
box.graphics.drawRect(0, 0, 150, 300);
box.graphics.endFill();
aSp.source = box;//load ScrollPane
```

For more information, see the ScrollPane class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

User interaction with the ScrollPane component

A ScrollPane can be enabled or disabled. A disabled ScrollPane doesn't receive mouse or keyboard input. A user can use the following keys to control a ScrollPane when it has focus:

Key	Description
Down Arrow	Content moves up one vertical line scroll.
Up Arrow	Content moves down one vertical line scroll.
End	Content moves to the bottom of the ScrollPane.
Left Arrow	Content moves to the right one horizontal line scroll.
Right Arrow	Content moves to the left one horizontal line scroll.
Home	Content moves to the top of the ScrollPane.
End	Content moves to the bottom of the ScrollPane.
PageDown	Content moves up one vertical scroll page.
PageUp	Content moves down one vertical scroll page.

A user can use the mouse to interact with the ScrollPane both on its content and on the vertical and horizontal scroll bars. The user can drag content by using the mouse when the `scrollDrag` property is set to `true`. The appearance of a hand pointer on the content indicates that the user can drag the content. Unlike most other controls, actions occur when the mouse button is pressed and continue until it is released. If the content has valid tab stops, you must set `scrollDrag` to false. Otherwise all mouse hits on the contents will invoke scroll dragging.

ScrollPane component parameters

You can set the following parameters for each ScrollPane instance in the Property inspector or in the Component inspector: `horizontalLineScrollSize`, `horizontalPageScrollSize`, `horizontalScrollPolicy`, `scrollDrag`, `source`, `verticalLineScrollSize`, `verticalPageScrollSize`, and `verticalScrollPolicy`. Each of these parameters has a corresponding ActionScript property of the same name. For information on the possible values for these parameters, see the ScrollPane class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

You can write ActionScript to control these and additional options for a ScrollPane component using its properties, methods, and events.

Create an application with the ScrollPane component

The following procedure explains how to add a ScrollPane component to an application while authoring. In this example, the ScrollPane loads a picture from a path specified by the `source` property.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the ScrollPane component from the Components panel to the Stage and give it an instance name of **aSp**.

3 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.events.ScrollEvent;

aSp.setSize(300, 200);

function scrollListener(event:ScrollEvent):void {
    trace("horizontalScPosition: " + aSp.horizontalScrollPosition +
     ", verticalScrollPosition = " + aSp.verticalScrollPosition);
};
aSp.addEventListener(ScrollEvent.SCROLL, scrollListener);

function completeListener(event:Event):void {
    trace(event.target.source + " has completed loading.");
};
// Add listener.
aSp.addEventListener(Event.COMPLETE, completeListener);

aSp.source = "http://www.helpexamples.com/flash/images/image1.jpg";
```

4 Select Control > Test Movie to run the application.

Create a ScrollPane instance using ActionScript

The example creates a ScrollPane, sets its size, and loads an image to it using the `source` property. It also creates two listeners. The first one listens for a `scroll` event and displays the image's position as the user scrolls vertically or horizontally. The second one listens for a `complete` event and displays a message in the Output panel that says the image has completed loading.

This example creates a ScrollPane using ActionScript and places a MovieClip (a red box) in it that is 150 pixels wide by 300 pixels tall.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the ScrollPane component from the Components panel to the Library panel.

3 Drag the DataGrid component from the Components panel to the Library panel.

4 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.containers.ScrollPane;
import fl.controls.ScrollPolicy;
import fl.controls.DataGrid;
import fl.data.DataProvider;

var aSp:ScrollPane = new ScrollPane();
var aBox:MovieClip = new MovieClip();
drawBox(aBox, 0xFF0000);//draw a red box

aSp.source = aBox;
aSp.setSize(150, 200);
aSp.move(100, 100);

addChild(aSp);

function drawBox(box:MovieClip,color:uint):void {
            box.graphics.beginFill(color, 1);
            box.graphics.drawRect(0, 0, 150, 300);
            box.graphics.endFill();
}
```

5 Select Control > Test Movie to run the application.

Use the Slider component

The Slider component lets a user select a value by sliding a graphical *thumb* between the end points of a track that corresponds to a range of values. You can use a slider to allow a user to choose a value such as a number or a percentage, for example. You can also use ActionScript to cause the slider's value to influence the behavior of a second object. For example, you could associate the slider with a picture and shrink it or enlarge it based on the relative position, or value, of the slider's thumb.

The current value of the Slider is determined by the thumb's relative location between the end points of the track or the Slider's minimum and maximum values.

The Slider allows for a continuous range of values between its minimum and maximum values, but you can also set the `snapInterval` parameter to specify intervals between the minimum and maximum value. A Slider can show tick marks, which are independent of the slider's assigned values, at specified intervals along the track.

The slider has a horizontal orientation by default, but you can give it a vertical orientation by setting the value of the `direction` parameter to vertical. The slider track stretches from end to end and the tick marks are placed from left to right just above the track.

User interaction with the Slider component

When a Slider instance has focus, you can use the following keys to control it:

Key	**Description**
Right Arrow	Increases the associated value for a horizontal slider.
Up Arrow	Increases the associated value for a vertical slider.
Left Arrow	Decreases the associated value for a horizontal slider.

Key	**Description**
Down Arrow	Decreases the associated value for a vertical slider.
Shift+Tab	Moves focus to the previous object.
Tab	Moves focus to the next object.

For more information about controlling focus, see the IFocusManager interface and the FocusManager class in the *ActionScript 3.0 Reference for the Adobe Flash Platform* and "Work with FocusManager" on page 26.

A live preview of each Slider instance reflects changes made to parameters in the Property inspector or the Component inspector during authoring.

Slider component parameters

You can set the following authoring parameters in the Property inspector or in the Component inspector for each Slider component instance: `direction`, `liveDragging`, `maximum`, `minimum`, `snapInterval`, `tickInterval`, and `value`. Each of these parameters has a corresponding ActionScript property of the same name. For information on the possible values for these parameters, see the Slider class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Create an application with the Slider

The following example creates a Slider instance to allow the user to express his or her level of satisfaction with some hypothetical event. The user moves the Slider to the right or the left to indicate a higher or lower level of satisfaction.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag a Label component from the Components panel to the center of the Stage.

- Give it an instance name of **valueLabel**.
- Assign the value **0percent** to the `text` parameter.

3 Drag a Slider component from the Components panel and center it below `value_lbl`.

- Give it an instance name of **aSlider**.
- Assign it a width (W:) of **200**.
- Assign it a height (H:) of **10**.
- Assign a value of **100** to the `maximum` parameter.
- Assign a value of **10** to both the `snapInterval` and `tickInterval` parameters.

4 Drag another Label instance from the Library panel and center it below `aSlider`.

- Give it an instance name of **promptLabel.**
- Assign it a width (W:) of 250.
- Assign it a height (H:) of 22.
- Enter **Please indicate your level of satisfaction** for the `text` parameter.

5 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.controls.Slider;
import fl.events.SliderEvent;
import fl.controls.Label;

aSlider.addEventListener(SliderEvent.CHANGE, changeHandler);

function changeHandler(event:SliderEvent):void {
    valueLabel.text = event.value + "percent";
}
```

6 Select Control > Test Movie.

In this example, as you move the thumb of the slider from one interval to another, a listener for the `SliderEvent.CHANGE` event updates the `text` property of `valueLabel` to display the percentage that corresponds to the thumb's position.

Create an application with the Slider component using ActionScript

The following example creates a Slider using ActionScript.The example downloads an image of a flower and uses the Slider to let the user fade or brighten the image by changing its `alpha` property to correspond to the Slider's value.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the Label component and the Slider component from the Components panel to the current document's Library panel.

This adds the components to the library, but doesn't make them visible in the application.

3 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following code to create and position the component instances:

```
import fl.controls.Slider;
import fl.events.SliderEvent;
import fl.controls.Label;
import fl.containers.UILoader;

var sliderLabel:Label = new Label();
sliderLabel.width = 120;
sliderLabel.text = "< Fade - Brighten >";
sliderLabel.move(170, 350);

var aSlider:Slider = new Slider();
aSlider.width = 200;
aSlider.snapInterval = 10;
aSlider.tickInterval = 10;
aSlider.maximum = 100;
aSlider.value = 100;
aSlider.move(120, 330);

var aLoader:UILoader = new UILoader();
aLoader.source = "http://www.flash-mx.com/images/image1.jpg";
aLoader.scaleContent = false;

addChild(sliderLabel);
addChild(aSlider);
addChild(aLoader);

aLoader.addEventListener(Event.COMPLETE, completeHandler);

function completeHandler(event:Event) {
    trace("Number of bytes loaded: " + aLoader.bytesLoaded);
}

aSlider.addEventListener(SliderEvent.CHANGE, changeHandler);

function changeHandler(event:SliderEvent):void {
        aLoader.alpha = event.value * .01;
}
```

4 Select Control > Test Movie to run the application.

5 Move the Slider's thumb to the left to fade the image and to the right to brighten it.

Use the TextArea component

The TextArea component is a wrapper for the native ActionScript TextField object. You can use the TextArea component to display text and also to edit and receive text input if the `editable` property is `true`. The component can display or receive multiple lines of text and wraps long lines of text if the `wordWrap` property is set to `true`. The `restrict` property allows you to restrict the characters that a user can enter and `maxChars` allows you to specify the maximum number of characters that a user can enter. If the text exceeds the horizontal or vertical boundaries of the text area, horizontal and vertical scroll bars automatically appear unless their associated properties, `horizontalScrollPolicy` and `verticalScrollPolicy`, are set to `off`.

You can use a TextArea component wherever you need a multiline text field. For example, you could use a TextArea component as a comment field in a form. You could set up a listener that checks whether the field is empty when a user tabs out of the field. That listener could display an error message indicating that a comment must be entered in the field.

If you need a single-line text field, use the TextInput component.

You can set the `textFormat` style using the `setStyle()` method to change the style of text that appears in a TextArea instance. You can also format a TextArea component with HTML using the `htmlText` property in ActionScript, and you can set the `displayAsPassword` property to `true` to mask text with asterisks. If you set the `condenseWhite` property to `true`, Flash removes extra white space in new text that is due to spaces, line breaks, and so on. It has no effect on text that is already in the control.

User interaction with the TextArea component

A TextArea component can be enabled or disabled in an application. In the disabled state, it cannot receive mouse or keyboard input. When enabled, it follows the same focus, selection, and navigation rules as an ActionScript TextField object. When a TextArea instance has focus, you can control it using the following keys:

Key	**Description**
Arrow keys	Move the insertion point up, down, left, or right within the text, if the text is editable.
Page Down	Moves the insertion point to the end of the text, if the text is editable.
Page Up	Moves the insertion point to the beginning of the text, if the text is editable.
Shift+Tab	Moves focus to the previous object in the Tab loop.
Tab	Moves focus to the next object in the Tab loop.

For more information about controlling focus, see the FocusManager class in the *ActionScript 3.0 Reference for the Adobe Flash Platform* and “Work with FocusManager” on page 26.

TextArea component parameters

You can set the following authoring parameters for each TextArea component instance in the Property inspector or the Component inspector: `condenseWhite`, `editable`, `hortizontalScrollPolicy`, `maxChars`, `restrict`, `text`, `verticalScrollPolicy`, and `wordwrap`. Each of these parameters has a corresponding ActionScript property of the same name. For information on the possible values for these parameters, see the TextArea class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

A live preview of each TextArea instance reflects changes made to parameters in the Property inspector or Component inspector during authoring. If a scroll bar is needed, it appears in the live preview, but it does not function. Text is not selectable in the live preview, and you cannot enter text in the component instance on the Stage.

You can write ActionScript to control these and additional options for the TextArea component using its properties, methods, and events. For more information, see the TextArea class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Create an application with the TextArea component

The following procedure explains how to add a TextArea component to an application while authoring. The example sets up a `focusOut` event handler on the TextArea instance that verifies that the user typed something in the text area before giving focus to a different part of the interface.

1 Create a new Flash document (ActionScript 3.0).

2 Drag a TextArea component from the Components panel to the Stage and give it an instance name of **aTa**. Leave its parameters set to the default settings.

3 Drag a second TextArea component from the Components panel to the Stage, place it below the first one and give it an instance name of **bTa**. Leave its parameters set to the default settings.

4 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import flash.events.FocusEvent;

aTa.restrict = "a-z,'\" \"";
aTa.addEventListener(Event.CHANGE,changeHandler);
aTa.addEventListener(FocusEvent.KEY_FOCUS_CHANGE, k_m_fHandler);
aTa.addEventListener(FocusEvent.MOUSE_FOCUS_CHANGE, k_m_fHandler);

function changeHandler(ch_evt:Event):void {
    bTa.text = aTa.text;
}
function k_m_fHandler(kmf_event:FocusEvent):void {
    kmf_event.preventDefault();
}
```

This example restricts the characters you can enter into the `aTa` text area to lowercase characters, the comma, the apostrophe, and spaces. It also sets up event handlers for the `change`, `KEY_FOCUS_CHANGE`, and `MOUSE_FOCUS_CHANGE` events on the `aTa` text area. The `changeHandler()` function causes the text that you enter in the `aTa` text area to automatically appear in the `bTa` text area by assigning `aTa.text` to `bTa.text` on each `change` event. The `k_m_fHandler()` function for the `KEY_FOCUS_CHANGE` and `MOUSE_FOCUS_CHANGE` events prevents you from pressing the Tab key to move to the next field without entering any text. It does this by preventing the default behavior.

5 Select Control > Test Movie.

If you press the Tab key to move focus to the second text area without entering any text, you should see an error message and focus should return to the first text area. As you enter text in the first text area, you will see it duplicated in the second text area.

Create a TextArea instance using ActionScript

The following example creates a TextArea component with ActionScript. It sets the `condenseWhite` property to `true` to condense white space and assigns text to the `htmlText` property to take advantage of HTML text formatting attributes.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the TextArea component to the Library panel.

3 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.controls.TextArea;

var aTa:TextArea = new TextArea();

aTa.move(100,100);
aTa.setSize(200, 200);
aTa.condenseWhite = true;
aTa.htmlText = '<b>Lorem ipsum dolor</b> sit amet, consectetuer adipiscing elit. <u>Vivamus
quis nisl vel tortor nonummy vulputate.</u> Quisque sit amet eros sed purus euismod tempor.
Morbi tempor. <font color="#FF0000">Class aptent taciti sociosqu ad litora torquent per
conubia nostra, per inceptos hymenaeos.</font> Curabitur diam. Suspendisse at purus in ipsum
volutpat viverra. Nulla pellentesque libero id libero.';
addChild(aTa);
```

This example uses the `htmlText` property to apply HTML bold and underline attributes to a block of text and display it in the `a_ta` text area. The example also sets the `condenseWhite` property to `true` to condense the white space within the text block. The `setSize()` method sets the text area's height and width, and the `move()` method sets its position. The `addChild()` method adds the TextArea instance to the Stage.

4 Select Control > Test Movie.

Use the TextInput component

The TextInput component is a single-line text component that is a wrapper for the native ActionScript TextField object. If you need a multiline text field, use the TextArea component. For example, you could use a TextInput component as a password field in a form. You could also set up a listener that checks whether the field has enough characters when a user tabs out of the field. That listener could display an error message indicating that the proper number of characters must be entered.

You can set the `textFormat` property using the `setStyle()` method to change the style of text that appears in a TextInput instance. A TextInput component can also be formatted with HTML or as a password field that disguises the text.

User interaction with the TextInput component

A TextInput component can be enabled or disabled in an application. In the disabled state, it doesn't receive mouse or keyboard input. When enabled, it follows the same focus, selection, and navigation rules as an ActionScript TextField object. When a TextInput instance has focus, you can also use the following keys to control it:

Key	Description
Arrow keys	Move the insertion point one character left or right.
Shift+Tab	Moves focus to the previous object.
Tab	Moves focus to the next object.

For more information about controlling focus, see the FocusManager class in the *ActionScript 3.0 Reference for the Adobe Flash Platform* and "Work with FocusManager" on page 26.

A live preview of each TextInput instance reflects changes made to parameters in the Property inspector or Component inspector during authoring. Text is not selectable in the live preview, and you cannot enter text in the component instance on the Stage.

When you add the TextInput component to an application, you can use the Accessibility panel to make it accessible to screen readers.

TextInput component parameters

You can set the following authoring parameters for each TextInput component instance in the Property inspector or the Component inspector: `editable`, `displayAsPassword`, `maxChars`, `restrict`, and `text`. Each of these parameters has a corresponding ActionScript property of the same name. For information on the possible values for these parameters, see the TextInput class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

You can write ActionScript to control these and additional options for the TextInput component using its properties, methods, and events. For more information, see the TextInput class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Create an application with the TextInput component

The following procedure explains how to add a TextInput component to an application. The example uses two TextInput fields to receive and confirm a password. It uses an event listener to see that a minimum of eight characters have been entered and that the text for the two fields matches.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag a Label component from the Components panel to the Stage and give it the following values in the Property inspector:
   - Enter the instance name **pwdLabel**.
   - Enter a value for W of **100**.
   - Enter a value for X of **50**.
   - Enter a value for Y of **150**.
   - In the Parameters section, enter a value of **Password:** for the text parameter.

3 Drag a second Label component from the Components panel to the Stage and give it the following values:
   - Enter the instance name **confirmLabel**.
   - Enter a value for W of **100**.
   - Enter a value for X of **50**.
   - Enter a value for Y of **200**.
   - In the Parameters section, enter a value of **ConfirmPassword:** for the text parameter.

4 Drag a TextInput component from the Components panel to the Stage and give it the following values:
   - Enter the instance name **pwdTi**.
   - Enter a value for W of **150**.
   - Enter a value for X of **190**.
   - Enter a value for Y of **150**.
   - In the Parameters section, double-click the value for the `displayAsPassword` parameter and select **true**. This causes the value entered in the text field to be masked with asterisks.

5 Drag a second TextInput component from the Components panel to the Stage and give it the following values:
   - Enter the instance name **confirmTi**.

- Enter a value for W of **150**.
- Enter a value for X of **190**.
- Enter a value for Y of **200**.
- In the Parameters section, double-click the value for the `displayAsPassword` parameter and select **true**. This causes the value entered in the text field to be masked with asterisks.

6 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
function tiListener(evt_obj:Event){
if(confirmTi.text != pwdTi.text || confirmTi.length < 8)
{
        trace("Password is incorrect. Please reenter it.");
}
else {
        trace("Your password is: " + confirmTi.text);
}
}
confirmTi.addEventListener("enter", tiListener);
```

This code sets up an `enter` event handler on the TextInput instance called `confirmTi`. If the two passwords don't match or the user types fewer than eight characters, the example displays the message: "Password is incorrect. Please reenter it." If the passwords are eight characters or more and they match, the example displays the value entered in the Output panel.

7 Select Control > Test Movie.

Create a TextInput instance using ActionScript

The following example creates a TextInput component using ActionScript. The example also creates a Label that it uses to prompt a user to enter his or her name. The example sets the component's `restrict` property to allow only uppercase and lowercase letters, a period, and a space. It also creates a TextFormat object that it uses to format the text in both the Label and TextInput components.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag a TextInput component from the Components panel to the Library panel.

3 Drag a Label component from the Components panel to the Library panel.

4 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.controls.Label;
import fl.controls.TextInput;

var nameLabel:Label = new Label();
var nameTi:TextInput = new TextInput();
var tf:TextFormat = new TextFormat();

addChild(nameLabel);
addChild(nameTi);

nameTi.restrict = "A-Z .a-z";

tf.font = "Georgia";
tf.color = 0x0000CC;
tf.size = 16;

nameLabel.text = "Name: " ;
nameLabel.setSize(50, 25);
nameLabel.move(100,100);
nameLabel.setStyle("textFormat", tf);
nameTi.move(160, 100);
nameTi.setSize(200, 25);
nameTi.setStyle("textFormat", tf);
```

5 Select Control > Test Movie to run the application.

Use the TileList component

The TileList component consists of a list that is made up of rows and columns that are supplied with data by a data provider. An *item* refers to a unit of data that is stored in a cell in the TileList. An item, which originates in the data provider, typically has a `label` property and a `source` property. The `label` property identifies the content to display in a cell and the `source` provides a value for it.

You can create an Array instance or retrieve one from a server. The TileList component has methods that proxy to its data provider, for example, the `addItem()` and `removeItem()` methods. If no external data provider is provided to the list, these methods create a data provider instance automatically, which is exposed through `List.dataProvider`.

User interaction with the TileList component

A TileList renders each cell using a Sprite that implements the ICellRenderer interface. You can specify this renderer with the TileList `cellRenderer` property. The TileList component's default CellRenderer is the ImageCell, which displays an image (class, bitmap, instance, or URL), and an optional label. The label is a single line that always aligns to the bottom of the cell. You can scroll a TileList in only one direction.

When a TileList instance has focus, you can also use the following keys to access the items within it:

Key	**Description**
Up Arrow and Down Arrow	Allow you to move up and down through a column. If the `allowMultipleSelection` property is `true`, you can use these keys in combination with the Shift key to select multiple cells.
Left Arrow and Right Arrow	Allow you to move to the left or right in a row. If the `allowMultipleSelection` property is `true`, you can use these keys in combination with the Shift key to select multiple cells.
Home	Selects the first cell in a TileList. If the `allowMultipleSelection` property is `true`, holding Shift and pressing Home will select all the cells from your current selection to the first cell.
End	Selects the last cell in a TileList. If the `allowMultipleSelection` property is `true`, holding Shift and pressing End will select all the cells from your current selection to the last cell.
Ctrl	If the `allowMultipleSelection` property is set to `true`, allows you to select multiple cells, in no specific order.

When you add the TileList component to an application, you can make it accessible to a screen reader by adding the following lines of ActionScript code:.

```
import fl.accessibility.TileListAccImpl;

TileListAccImpl.enableAccessibility();
```

You enable accessibility for a component only once, regardless of how many instances the component has. For more information, see Chapter 18, "Creating Accessible Content," in *Using Flash*.

TileList component parameters

You can set the following authoring parameters in the Property inspector or in the Component inspector for each TileList component instance: `allowMultipleSelection`, `columnCount`, `columnWidth`, `dataProvider`, `direction`, `horizontalScrollLineSize`, `horizontalScrollPageSize`, `labels`, `rowCount`, `rowHeight`, `ScrollPolicy`, `verticalScrollLineSize, and verticalScrollPageSize`. Each of these parameters has a corresponding ActionScript property of the same name. For information on using the dataProvider parameter, see "Use the dataProvider parameter" on page 28.

You can write ActionScript to set additional options for TileList instances using its methods, properties, and events. For more information, see the TileList class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Create an application with the TileList component

This example uses MovieClips to fill a TileList with an array of paint colors.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag a TileList component to the Stage and give it an instance name of **aTl**.

3 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.data.DataProvider;
import flash.display.DisplayObject;

var aBoxes:Array = new Array();
var i:uint = 0;
var colors:Array = new Array(0x00000, 0xFF0000, 0x0000CC, 0x00CC00, 0xFFFF00);
var colorNames:Array = new Array("Midnight", "Cranberry", "Sky", "Forest", "July");
var dp:DataProvider = new DataProvider();
for(i=0; i < colors.length; i++) {
    aBoxes[i] = new MovieClip();
    drawBox(aBoxes[i], colors[i]);// draw box w next color in array
    dp.addItem( {label:colorNames[i], source:aBoxes[i]} );
}
aTl.dataProvider = dp;
aTl.columnWidth = 110;
aTl.rowHeight = 130;
aTl.setSize(280,150);
aTl.move(150, 150);
aTl.setStyle("contentPadding", 5);

function drawBox(box:MovieClip,color:uint):void {
            box.graphics.beginFill(color, 1.0);
            box.graphics.drawRect(0, 0, 100, 100);
            box.graphics.endFill();
}
```

4 Select Control > Test Movie to test the application.

Create a TileList component using ActionScript

This example dynamically creates a TileList instance and adds instances of the ColorPicker, ComboBox, NumericStepper, and CheckBox components to it. It creates an Array that contains labels and the names of the component to display and assigns the Array (`dp`) to the TileList's `dataProvider` property. It uses the `columnWidth` and `rowHeight` properties and the `setSize()` method to lay out the TileList, the `move()` method to position it on the Stage, and the `contentPadding` style to put space between the TileList instance's borders and its content, and the `sortItemsOn()` method to sort the content by its labels.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the following components from the Components panel to the Library panel: ColorPicker, ComboBox, NumericStepper, CheckBox, and TileList.

3 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.controls.CheckBox;
import fl.controls.ColorPicker;
import fl.controls.ComboBox;
import fl.controls.NumericStepper;
import fl.controls.TileList;
import fl.data.DataProvider;

var aCp:ColorPicker = new ColorPicker();
var aCb:ComboBox = new ComboBox();
var aNs:NumericStepper = new NumericStepper();
var aCh:CheckBox = new CheckBox();
var aTl:TileList = new TileList();

var dp:Array = [
{label:"ColorPicker", source:aCp},
{label:"ComboBox", source:aCb},
{label:"NumericStepper", source:aNs},
{label:"CheckBox", source:aCh},
];
aTl.dataProvider = new DataProvider(dp);
aTl.columnWidth = 110;
aTl.rowHeight = 100;
aTl.setSize(280,130);
aTl.move(150, 150);
aTl.setStyle("contentPadding", 5);
aTl.sortItemsOn("label");
addChild(aTl);
```

4 Select Test > Control Movie to test the application.

Use the UILoader component

The UILoader component is a container that can display SWF, JPEG, progressive JPEG, PNG, and GIF files. You can use a UILoader whenever you need to retrieve content from a remote location and pull it into a Flash application. For example, you could use a UILoader to add a company logo (JPEG file) to a form. You could also use the UILoader component in an application that displays photos. Use the `load()` method to load content, the `percentLoaded` property to determine how much content has loaded, and the `complete` event to determine when loading is finished.

You can scale the contents of the UILoader or resize the UILoader itself to accommodate the size of the contents. By default, the contents are scaled to fit the UILoader. You can also load content at run time and monitor loading progress (although after content has been loaded once it is cached, the progress jumps to 100% quickly). If you specify a location when loading content in the UILoader, you must specify the location (X and Y coordinates) as 0, 0.

User interaction with the UILoader component

A UILoader component can't receive focus. However, content loaded into the UILoader component can accept focus and have its own focus interactions. For more information about controlling focus, see the FocusManager class in the *ActionScript 3.0 Reference for the Adobe Flash Platform* and "Work with FocusManager" on page 26.

UILoader component parameters

You can set the following authoring parameters in the Property inspector or in the Component inspector for each UILoader component instance: `autoLoad`, `maintainAspectRatio`, `source`, and `scaleContent`. Each of these parameters has a corresponding ActionScript property of the same name.

A live preview of each UILoader instance reflects changes made to parameters in the Property inspector or Component inspector during authoring.

You can write ActionScript to set additional options for UILoader instances using its methods, properties, and events. For more information, see the UILoader class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Create an application with the UILoader component

The following procedure explains how to add a UILoader component to an application while authoring. In this example, the loader loads a GIF image of a logo.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag a UILoader component from the Components panel to the Stage.

3 In the Property inspector, enter the instance name **aUI**.

4 Select the loader on the Stage and in the Component inspector, and enter **http://www.helpexamples.com/images/logo.gif** for the `source` parameter.

Create a UILoader component instance using ActionScript

This example creates a UILoader component using ActionScript and loads a JPEG image of a flower. When the `complete` event occurs, it displays the number of bytes loaded in the Output panel.

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the UILoader component from the Components panel to the Library panel.

3 Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code:

```
import fl.containers.UILoader;

var aLoader:UILoader = new UILoader();
aLoader.source = "http://www.flash-mx.com/images/image1.jpg";
aLoader.scaleContent = false;
addChild(aLoader);

aLoader.addEventListener(Event.COMPLETE, completeHandler);
function completeHandler(event:Event) {
    trace("Number of bytes loaded: " + aLoader.bytesLoaded);
}
```

4 Select Control > Test Movie.

Use the UIScrollBar component

The UIScrollBar component allows you to add a scroll bar to a text field. You can add a scroll bar to a text field while authoring, or at run time with ActionScript. To use the UIScrollBar component, create a text field on the Stage and drag the UIScrollBar component from the Components panel to any quadrant of the text field’s bounding box.

If the length of the scroll bar is smaller than the combined size of its scroll arrows, it does not display correctly. One of the arrow buttons becomes hidden behind the other. Flash does not provide error checking for this. In this case it is a good idea to hide the scroll bar with ActionScript. If the scroll bar is sized so that there is not enough room for the scroll box (thumb), Flash makes the scroll box invisible.

The UIScrollBar component functions like any other scroll bar. It contains arrow buttons at either end and a scroll track and scroll box (thumb) in between. It can be attached to any edge of a text field and used both vertically and horizontally.

For information on the TextField, see the TextField class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

User interaction with the UIScrollBar component

Unlike many other components, the UIScrollBar component can receive continuous mouse input, such as when the user holds the mouse button down, rather than requiring repeated clicks.

There is no keyboard interaction with the UIScrollBar component.

UIScrollBar component parameters

You can set the following authoring parameters in the Property inspector or in the Component inspector for each UIScrollBar component instance: `direction` and `scrollTargetName`. Each of these parameters has a corresponding ActionScript property of the same name.

You can write ActionScript to set additional options for UIScrollBar instances using class methods, properties, and events. For more information, see the UIScrollBar class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Create an application with the UIScrollBar component

The following procedure describes how to add a UIScrollBar component to an application while authoring.

1. Create a new Flash (ActionScript 3.0) document.
2. Create a dynamic text field that is tall enough to hold one or two lines of text and give it an instance name **myText** in the Property inspector.
3. In the Property inspector, set the Line Type of the text input field to Multiline or to Multiline No Wrap if you plan to use the scroll bar horizontally.
4. Open the Actions panel, select Frame 1 in the main Timeline, and enter the following ActionScript code to fill the `text` property so that a user will need to scroll it to view it all:

   ```
   myText.text="When the moon is in the seventh house and Jupiter aligns with Mars, then peace
   will guide the planet and love will rule the stars."
   ```

   ***Note:** Make sure that the text field on the Stage is small enough that you need to scroll it to see all the text. If it isn't, the scroll bar does not appear or may appear simply as two lines with no thumb grip (the part you drag to scroll the content).*

5. Verify that object snapping is turned on (View > Snapping > Snap To Objects).
6. Drag a UIScrollBar instance from the Components panel onto the text input field near the side you want to attach it to. The component must overlap with the text field when you release the mouse in order for it to be properly bound to the field. Give it an instance name of **mySb.**

   The `scrollTargetName` property of the component is automatically populated with the text field instance name in the Property and Component inspectors. If it does not appear on the Parameters tab, you may not have overlapped the UIScrollBar instance enough.

7 Select Control > Test Movie.

Create a UIScrollBar component instance using ActionScript

You can create a UIScrollBar instance with ActionScript and associate it with a text field at run time. The following example creates a horizontally oriented UIScrollBar instance and attaches it to the bottom of a text field instance named **myTxt**, which is loaded with text from a URL. The example also sets the size of the scroll bar to match the size of the text field:

1 Create a new Flash (ActionScript 3.0) document.

2 Drag the ScrollBar component to the Library panel.

3 Open the Actions panel, select Frame 1 of the main Timeline, and enter the following ActionScript code:

```
import flash.net.URLLoader;
import fl.controls.UIScrollBar;
import flash.events.Event;

var myTxt:TextField = new TextField();
myTxt.border = true;
myTxt.width = 200;
myTxt.height = 16;
myTxt.x = 200;
myTxt.y = 150;

var mySb:UIScrollBar = new UIScrollBar();
mySb.direction = "horizontal";
// Size it to match the text field.
mySb.setSize(myTxt.width, myTxt.height);

// Move it immediately below the text field.
mySb.move(myTxt.x, myTxt.height + myTxt.y);

// put them on the Stage
addChild(myTxt);
addChild(mySb);
// load text
var loader:URLLoader = new URLLoader();
var request:URLRequest = new URLRequest("http://www.helpexamples.com/flash/lorem.txt");
loader.load(request);
loader.addEventListener(Event.COMPLETE, loadcomplete);

function loadcomplete(event:Event) {
    // move loaded text to text field
    myTxt.text = loader.data;
    // Set myTxt as target for scroll bar.
    mySb.scrollTarget = myTxt;
}
```

4 Select Control > Test Movie.

Chapter 5: Customizing the UI Components

About UI component customization

You can customize the appearance of components in your applications by modifying one or both of the following elements:

Styles Each component has a set of styles that you can set to specify what values Flash uses to render the component's appearance. Styles generally specify skins and icons to use for a component in its different states and also what text formatting and padding values to use.

Skins A *skin* consists of the collection of symbols that make up the component's graphical appearance in a given state. While a style specifies what skin to use, a skin is a graphical element that Flash uses to draw the component. *Skinning* is the process of changing a component's appearance by modifying or replacing its graphics.

Note: *The default appearance of ActionScript 3.0 components could be considered a theme (Aeon Halo), but these skins are built into the components. The ActionScript 3.0 components do not support the external theme files that ActionScript 2.0 components did.*

Setting styles

A component's styles generally specify values for its skins, icons, text formatting, and padding when Flash draws the component in its various states. For example, Flash draws a Button with a different skin to show its down state, which occurs when you click the mouse button on it, than it does to show its up or normal state. It also uses a different skin when it's in a disabled state, which is caused by setting the `enabled` property to `false`.

You can set styles for components at the document, class, and instance levels. In addition, some style properties can be inherited from a parent component. For example, the List component inherits ScrollBar styles by inheriting from BaseScrollPane.

You can set styles to customize a component in the following ways:

- Set styles on a component instance. You can change color and text properties for a single component instance. This is effective in some situations, but it can be time-consuming if you need to set individual properties on all the components in a document.
- Set styles for all components of a given type in a document. If you want to apply a consistent look to all components of a given type, for example to all CheckBoxes or all Buttons in a document, you can set styles at the component level.

  The values of style properties set on containers are inherited by contained components.

  Flash does not display changes made to style properties when you view components on the Stage using the Live Preview feature.

Understanding style settings

Here are a few key points about using styles:

Inheritance A component child is set to inherit a style from the parent component by design. You cannot set inheritance for styles within ActionScript.

Precedence If a component style is set in more than one way, Flash uses the first style it encounters according to its order of precedence. Flash looks for styles in the following order until a value is found:

1. Flash looks for a style property on the component instance.
2. If the style is one of the inheriting styles, Flash looks through the parent hierarchy for an inherited value.
3. Flash looks for the style on the component.
4. Flash looks for a global setting on StyleManager.
5. If the property is still not defined, the property has the value `undefined`.

Accessing a component's default styles

You can access the default styles for a component using the static `getStyleDefinition()` method for the component class. For example, the following code retrieves the default styles for the ComboBox component and displays the default values for the `buttonWidth` and `downArrowDownSkin` properties:

```
import fl.controls.ComboBox;
var styleObj:Object = ComboBox.getStyleDefinition();
trace(styleObj.buttonWidth); // 24
trace(styleObj.downArrowDownSkin); // ScrollArrowDown_downSkin
```

Setting and getting styles on a component instance

Any UI component instance can call the `setStyle()` and `getStyle()` methods directly to set or retrieve a style. The following syntax sets a style and value for a component instance:

```
instanceName.setStyle("styleName", value);
```

This syntax retrieves a style for a component instance:

```
var a_style:Object = new Object();
a_style = instanceName.getStyle("styleName");
```

Notice that the `getStyle()` method returns the type Object because it can return multiple styles having different data types. For example, the following code sets the font style for a TextArea instance (`aTa`) and then retrieves it using the `getStyle()` method. The example casts the returned value to a TextFormat object to assign it to a TextFormat variable. Without the cast, the compiler would issue an error for attempting to coerce an Object variable to a TextFormat variable.

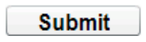
```
import flash.text.TextFormat;

var tf:TextFormat = new TextFormat();
tf.font = "Georgia";
aTa.setStyle("textFormat",tf);
aTa.text = "Hello World!";
var aStyle:TextFormat = aTa.getStyle("textFormat") as TextFormat;
trace(aStyle.font);
```

Use TextFormat to set text properties

Use the TextFormat object to format text for a component instance. The TextFormat object has properties that allow you to specify text characteristics such as `bold`, `bullet`, `color`, `font`, `italic`, `size`, and several others. You can set these properties in the TextFormat object and then call the `setStyle()` method to apply them to a component instance. For example, the following code sets the `font`, `size`, and `bold` properties of a TextFormat object and applies them to a Button instance:

```
/* Create a new TextFormat object to set text formatting properties. */
var tf:TextFormat = new TextFormat();
tf.font = "Arial";
tf.size = 16;
tf.bold = true;
a_button.setStyle("textFormat", tf);
```

The following illustration shows the effect of these settings on a button having a Submit label:



Style properties set on a component instance through `setStyle()` have the highest priority and override all other style settings. However, the more properties you set using `setStyle()` on a single component instance, the slower the component will render at run time.

Setting a style for all instances of a component

You can set a style for all instances of a component class using the static `setComponentStyle()` method of the StyleManager class. For example, you can set the color of text to red for all Buttons by first dragging a Button to the Stage and then adding the following ActionScript code to the Actions panel on Frame 1 of the Timeline:

```
import fl.managers.StyleManager;
import fl.controls.Button;

var tf:TextFormat = new TextFormat();
tf.color = 0xFF0000;
StyleManager.setComponentStyle(Button, "textFormat", tf);
```

All Buttons that you subsequently add to the Stage will have red labels.

Set a style for all components

You can set a style for all component using the static `setStyle()` method of the StyleManager class.

1 Drag a List component to the Stage and give it an instance name of **aList**.

2 Drag a Button component to the Stage and give it an instance name of **aButton**.

3 Press **F9** or select Actions from the Window menu to open the Actions panel, if it isn't already open, and enter the following code in Frame 1 of the Timeline to set the color of text to red for all components:

```
import fl.managers.StyleManager;

var tf:TextFormat = new TextFormat();
tf.color = 0xFF0000;
StyleManager.setStyle("textFormat", tf);
```

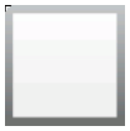
4 Add the following code to the Actions panel to populate the List with text.

```
aList.addItem({label:"1956 Chevy (Cherry Red)", data:35000});
aList.addItem({label:"1966 Mustang (Classic)", data:27000});
aList.addItem({label:"1976 Volvo (Xcllnt Cond)", data:17000});
aList.allowMultipleSelection = true;
```

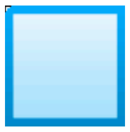
5 Select Control > Test Movie or press Ctrl+Enter to compile the code and test your content. The text in both the button label and in the list should be red.

About Skins

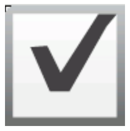
A component's appearance is made up of graphical elements such as an outline, a fill color, icons, and even other components. A ComboBox, for example, contains a List component and a List component contains a ScrollBar. Together the graphical elements make up the appearance for the ComboBox. A component's appearance changes, however, based on its current state. For example, a CheckBox, without its label, looks something like this when it appears in your application:


A CheckBox in its normal up state

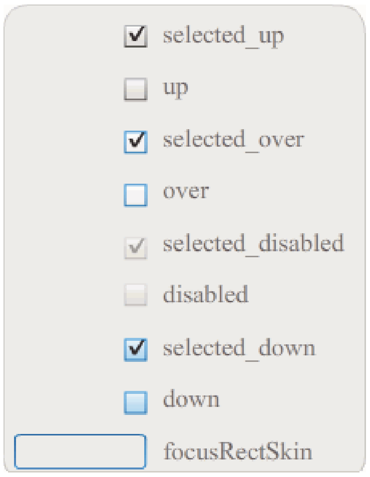
If you click the mouse button and hold it down on the CheckBox, its appearance changes to this:


A CheckBox in its down state

And when you release the mouse button, the CheckBox reverts to its original appearance but now has a check mark to show that it has been selected.

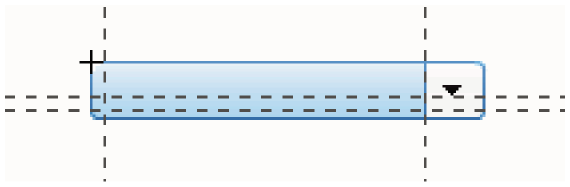

A CheckBox in its selected state

Collectively, the icons that represent the component in its various states are called its *skins*. You can change a component's appearance in any or all of its states by editing its skins in Flash, just as you would any other Flash symbol. You can access a component's skins in two ways. The easiest way is to drag the component to the Stage and double-click it. This opens a palette of the component's skins, which looks like this for a CheckBox.



A CheckBox’s skins

You can also access a component’s skins individually from the Library panel. When you drag a component to the Stage, you also copy it to the library along with a folder of its assets and any other components that it contains. For example, if you drag a ComboBox to the Stage, the Library panel will also contain the List, ScrollBar, and TextInput components, which are built into the ComboBox, along with a folder of skins for each of these components and a Shared Assets folder that contains elements that these components share. You can edit the skins for any of these components by opening its skins folder (ComboBoxSkins, ListSkins, ScrollBarSkins, or TextInputSkins) and double-clicking the icon for the skin that you want to edit. Double-clicking ComboBox_downSkin, for example, opens the skin in symbol editing mode, as shown in the following illustration:


The ComboBox_downSkin

Create a new skin

If you want to create a new look for a component in your document, you edit the component’s skins to change their appearance. To access a component’s skins, simply double-click the component on the Stage to open a palette of its skins. Then double-click the skin that you want to edit to open it in symbol-editing mode. For example, double-click the TextArea component on the Stage to open its assets in symbol-editing mode. Set the zoom control to 400%, or higher if you like, and edit the symbol to change its look. When you’re finished, the change will affect all instances of the component in the document. As an alternative, you can double-click a particular skin in the Library panel to open it on the Stage in symbol-editing mode.

You can modify component skins in the following ways:

- Create a new skin for all instances
- Create new skins for some instances

Create a skin for all instances

When you edit a component's skin, by default you change the component's appearance for all instances of it in the document. If you want to create different looks for the same component, you must duplicate the skins that you want to change and give them different names, edit them, and then set the appropriate styles to apply them. For more information, see "Create skins for some instances" on page 100.

This chapter describes how to alter one or more skins for each of the UI components. If you follow one of these procedures to change one or more of a UI component's skins, you will change it for all instances in the document.

Create skins for some instances

You can create a skin for some instances of a component using the following general procedure:

- Select the skin in the component's Assets folder in the Library panel.
- Duplicate the skin and assign it a unique class name.
- Edit the skin to give it the appearance you want.
- Call the `setStyle()` method for the component instance to assign the new skin to the skin style.

The following procedure creates a new selectedDownSkin for one of two Button instances.

1. Create a new Flash file (ActionScript 3.0) document.
2. Drag two Buttons from the Components panel onto the Stage and give them instance names of **aButton** and **bButton**.
3. Open the Library panel and then the Component Assets and ButtonSkins folders within it.
4. Click the selectedDownSkin skin to select it.
5. Right-click to open the context menu and select Duplicate.
6. In the Duplicate Symbol dialog box, give the new skin a unique name, for example **Button_mySelectedDownSkin**. Then click OK.
7. In the Library > Component Assets > ButtonSkins folder, select Button_mySelectedDownSkin and right-click to open the context menu. Select Linkage to open the Linkage Properties dialog box.
8. Click the Export For ActionScript check box. Leave the Export In First Frame check box selected and ensure that the class name is unique. Click OK, and then click OK again in response to the warning that says a class definition could not be found and one will be created.
9. Double-click the Button_mySelectedDownSkin skin in the Library panel to open it in symbol-editing mode.
10. Click the blue fill in the center of the skin until the color appears in the Fill color picker in the Property inspector. Click the color picker and select color #00CC00 for the skin fill.
11. Click the Back button at the left side of the edit bar above the Stage to return to document-editing mode.
12. In the Property inspector, click the Parameters tab for each button and set the toggle parameter to `true`.
13. Add the following code to the Actions panel on Frame 1 of the Timeline:

```
bButton.setStyle("selectedDownSkin", Button_mySelectedDownSkin);
bButton.setStyle("downSkin", Button_mySelectedDownSkin);
```

14. Select Control > Text Movie.
15. Click each button. Note that the down skin (selected and unselected) for the bButton object uses the new skin symbol.

Customize the Button component

You can transform a Button component horizontally and vertically while authoring and at run time. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At run time, use the `setSize()` method or any applicable properties of the Button class, such as `height` and `width`, and `scaleX` and `scaleY`.

Resizing the button does not change the size of the icon or label. The bounding box of a Button corresponds to the Button's border and also designates the hit area for the instance. If you increase the size of the instance, you also increase the size of the hit area. If the bounding box is too small to fit the label, the label is clipped to fit.

If the Button has an icon and it is larger than the Button, the icon extends beyond the Button's borders.

Use styles with the Button component

A Button's styles generally specify values for a Button's skins, icons, text formatting, and padding when the component is drawn in its various states.

The following procedure puts two Buttons on the Stage and sets the `emphasized` property to `true` for both Buttons when the user clicks one of them. It also sets the `emphasizedSkin` style for the second Button to the `selectedOverSkin` style when the user clicks it so the two Buttons show different skins for the same state.

1 Create a Flash file (ActionScript 3.0).

2 Drag two Buttons to the Stage, one at a time, and give them instance names of **aBtn** and **bBtn**. In the Parameters tab of the Property inspector, give them labels of Button A and Button B.

3 Add the following code to the Actions panel on Frame 1 of the Timeline:

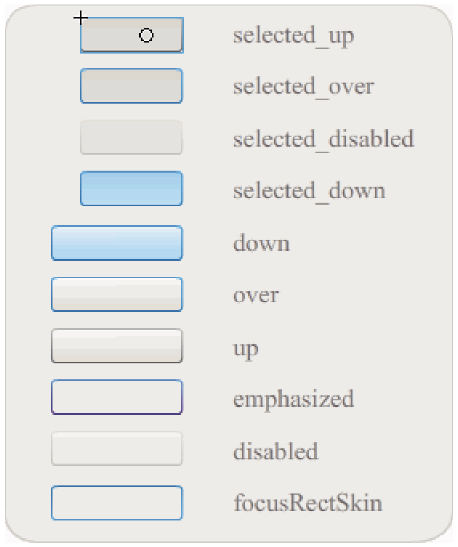
```
bBtn.emphasized = true;
aBtn.emphasized = true;
bBtn.addEventListener(MouseEvent.CLICK, Btn_handler);
function Btn_handler(evt:MouseEvent):void {
    bBtn.setStyle("emphasizedSkin", "Button_selectedOverSkin");
}
```

4 Select Control > Test Movie.

5 Click one of the buttons to see the effect of the `emphasizedSkin` style on each button.

Use skins with the Button component

The Button component uses the following skins that correspond to its different states. To edit one or more skins to change the Button's appearance, double-click the Button instance on the Stage to open a palette of its skins, as shown in the following illustration:

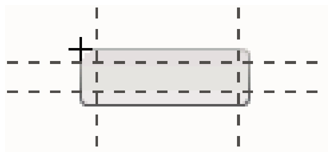


Button skins

If a button is enabled, it displays its over state when the pointer moves over it. The button receives input focus and displays its down state when it's pressed. The button returns to its over state when the mouse is released. If the pointer moves off the button while the mouse is pressed, the button returns to its original state. If the toggle parameter is set to `true`, the pressed state is shown with the selectedDownSkin, the up state with the selectedUpSkin, and the over state with the selectedOverSkin.

If a Button is disabled, it displays its disabled state, regardless of user interaction.

To edit one of the skins, double-click it to open it in symbol-editing mode, as shown in the following illustration:



Button in symbol-editing mode

At this point you can use the Flash authoring tools to edit the skin to your liking.

The following procedure changes the color of the Button's selected_over skin.

1 Create new Flash file (ActionScript 3.0).

2 Drag a Button from the Components panel to the Stage. In the Parameters tab, set the toggle parameter to `true`.

3 Double-click the Button to open the palette of its skins.

4 Double-click the selected_over skin to open it in symbol-editing mode.

5 Set the zoom control to 400% to enlarge the icon for editing.

6 Double-click the background until its color appears in the Fill color picker in the Property inspector.

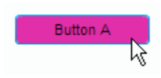
7 Select color #CC0099 in the Fill color picker to apply it to the background of the selected_over skin.

8 Click the Back button at the left side of the edit bar above the Stage to return to document-editing mode.

9 Select Control > Test Movie.

10 Click the button to put it in the selected state.

When you move the mouse pointer over the Button, the selected_over state should appear as it does in the following illustration.



Button showing selected_over skin with modified color

Customize the CheckBox component

You can transform a CheckBox component horizontally and vertically while authoring and at run time. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At run time, use the `setSize()` method or applicable properties of the CheckBox class. For example, you can change the size of a CheckBox by setting its `height` and `width` and `scaleX` and `scaleY` properties. Resizing the CheckBox does not change the size of the label or the check box icon; it only changes the size of the bounding box.

The bounding box of a CheckBox instance is invisible and also designates the hit area for the instance. If you increase the size of the instance, you also increase the size of the hit area. If the bounding box is too small to fit the label, the label is clipped to fit.

Use styles with the CheckBox

You can set style properties to change the appearance of a CheckBox instance. For example, the following procedure changes the size and color of a CheckBox label.

1 Drag the CheckBox component from the Components panel to the Stage and give it an instance name of **myCb.**

2 Click the Parameters tab in the Property inspector and enter the following value for the label parameter: **Less than $500?**

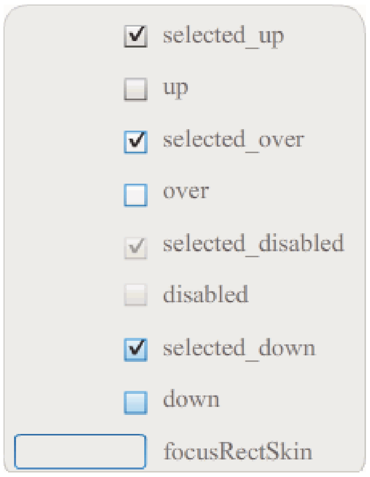
3 On Frame 1 of the main Timeline, enter the following code in the Actions Panel:

```
var myTf:TextFormat = new TextFormat();
myCb.setSize(150, 22);
myTf.size = 16;
myTf.color = 0xFF0000;
myCb.setStyle("textFormat", myTf);
```

For more information, see "Setting styles" on page 95. For information on setting style properties to change the component's icons and skins, see "Create a new skin" on page 99 and "Use skins with the CheckBox" on page 103.

Use skins with the CheckBox

The CheckBox component has the following skins, which you can edit to change its appearance.



CheckBox skins

This example changes the outline color and background color of the component in its `up` and `selectedUp` states. You would follow similar steps to change the skins for other states.

1 Create a new Flash file (ActionScript 3.0) document.

2 Drag the CheckBox component to the Stage, which also places it in the library with a folder of its assets.

3 Double-click the CheckBox component on the Stage to open its panel of skin icons.

4 Double-click the selected_up icon to open it in symbol-editing mode.

5 Set the zoom control to 800% to enlarge the icon for editing.

6 Click the border of the CheckBox to select it. Use the Fill color picker in the Property inspector to select color #0033FF and apply it to the border.

7 Double-click the background of the CheckBox to select it and again use the Fill color picker to set the color of the background to #00CCFF.

8 Repeat steps 4 to 8 for the CheckBox up skin.

9 Select Control > Test Movie.

Customize the ColorPicker component

The only resizing that you can do to a ColorPicker is through its styles: `swatchWidth`, `swatchHeight`, `backgroundPadding`, `textFieldWidth`, and `textFieldHeight`. If you try to change the size of the ColorPicker with the Transform tool or with ActionScript using the `setSize()` method, or the `width`, `height`, `scaleX`, or `scaleY` properties, these values are ignored when you create the SWF file and the ColorPicker displays at its default size. The palette background will resize to match the number of columns that has been set using `setStyle()` for the `columnCount` style. The default number of columns is 18. You can set custom colors to 1024 and the palette will resize vertically to match the number of swatches.

Use Styles with the ColorPicker component

You can set several styles to change the appearance of the ColorPicker component. For example the following procedure changes the number of columns (`columnCount`) in the ColorPicker to 12, changes the height (`swatchHeight`) and width (`swatchWidth`) of the color swatches, and changes the padding for both the text field (`textPadding`) and the background (`backgroundPadding`).

1 Create a new Flash file (ActionScript 3.0) document.

2 Drag the ColorPicker component to the Stage and give it an instance name of **aCp**.

3 Open the Actions panel, select Frame 1 in the main Timeline and enter the following code:

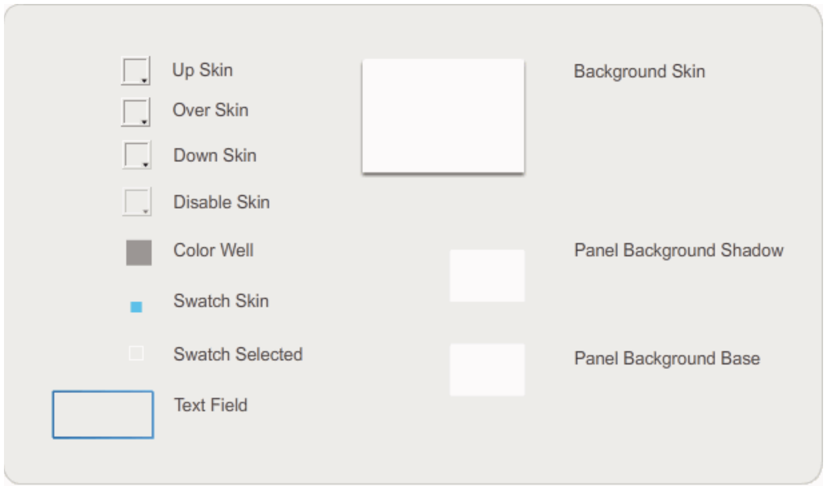
```
aCp.setStyle("columnCount", 12);
aCp.setStyle("swatchWidth", 8);
aCp.setStyle("swatchHeight", 12);
aCp.setStyle("swatchPadding", 2);
aCp.setStyle("backgroundPadding", 3);
aCp.setStyle("textPadding", 7);
```

4 Select Control > Test Movie.

5 Click the ColorPicker to open it and see how these settings have altered its appearance.

Use Skins with the ColorPicker component

The ColorPicker component uses the following skins to represent its visual states.



ColorPicker skins

You can change the color of the Background skin to change the color of the palette background.

1 Create a new Flash file (ActionScript 3.0) document.

2 Drag the ColorPicker component to the Stage.

3 Double-click it to open its palette of skins.

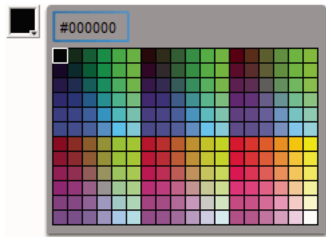
4 Double-click the Background skin until it is selected and the Fill color picker appears in the Property inspector.

5 Select color #999999 using the Fill color picker to apply it to the Background skin.

6 Click the Back button at the left side of the edit bar above the Stage to return to document-editing mode.

7 Select Control > Test Movie.

When you click on the ColorPicker, the background of the palette should be gray as shown in the following illustration.



ColorPicker with dark gray Background skin

Customize the ComboBox component

You can transform a ComboBox component horizontally and vertically while authoring and at run time. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At run time, use the `setSize()` method or applicable properties of the ComboBox class such as `height` and `width` and `scaleX` and `scaleY`.

The ComboBox will resize to fit the specified width and height. The list will resize to fit the width of the component, unless the `dropdownWidth` property has been set.

If the text is too long to fit in the ComboBox, the text is clipped to fit. You must resize the ComboBox and set the `dropdownWidth` property to fit the text.

Use Styles with the ComboBox component

You can set style properties to change the appearance of a ComboBox component. The styles specify values for the component's skins, cell renderer, padding, and button width. The following example sets the `buttonWidth` and `textPadding` styles. The `buttonWidth` style sets the width of the button's hit area and is in effect when the ComboBox is editable and you can only press the button to open the drop-down list.The textPadding style specifies the amount of space between the outside border of the text field and the text. It is useful for centering the text vertically in the text field if you make the ComboBox taller. Otherwise the text could appear to be at the top of the text field.

1 Create a new Flash file (ActionScript 3.0) document.

2 Drag the ComboBox component to the Stage and give it an instance name of **aCb**.

3 Open the Actions panel, select Frame 1 in the main Timeline and enter the following code:

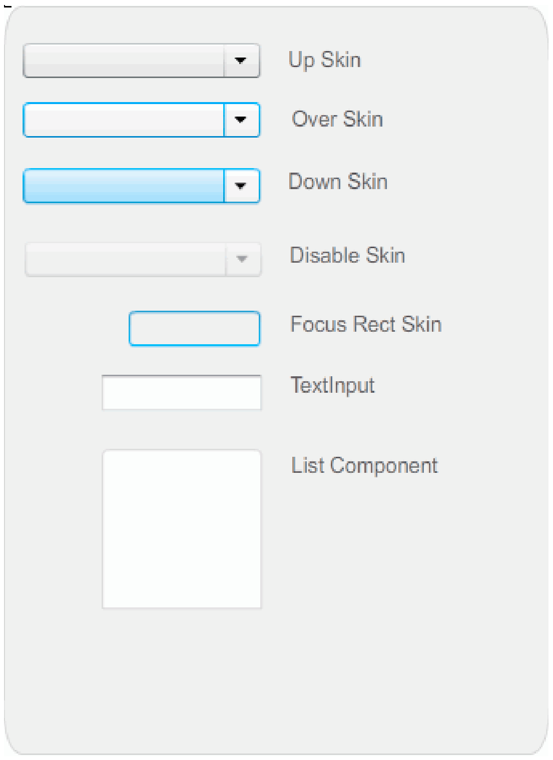
```
import fl.data.DataProvider;

aCb.setSize(150, 35);
aCb.setStyle("textPadding", 10);
aCb.setStyle("buttonWidth", 10);
aCb.editable = true;

var items:Array = [
{label:"San Francisco", data:"601 Townsend St."},
{label:"San Jose", data:"345 Park Ave."},
{label:"San Diego", data:"10590 West Ocean Air Drive, Suite 100"},
{label:"Santa Rosa", data:"2235 Mercury Way, Suite 105"},
{label:"San Luis Obispo", data:"3220 South Higuera Street, Suite 311"}
];
aCb.dataProvider = new DataProvider(items);
```

4 Select Control > Test Movie.

Notice that the area of the button that you can click to open the drop-down list is only a narrow area on the right side. Notice also that the text is centered vertically in the text field. You can try running the example without the two `setStyle()` statements to see their effect.

Use Skins with the ComboBox

The ComboBox uses the following skins to represent its visual states:



ComboBox skins

You can change the color of the Up skin to change the color of the component in its inactive state on the Stage.

1 Create a new Flash file (ActionScript 3.0) document.

2 Drag the ComboBox component to the Stage.

3 Double-click it to open its palette of skins.

4 Double-click the Up skin until it is selected and open for editing.

5 Set the zoom control to 400%.

6 Click the center area of the skin until its color appears in the Fill color picker in the Property inspector.

7 Select color #33FF99 using the Fill color picker to apply it to the Up skin.

8 Click the Back button at the left side of the edit bar above the Stage to return to document-editing mode.

9 Select Control > Test Movie.

The ComboBox should appear on the Stage as shown in the following illustration.

ComboBox with customized color for background skin

Customize the DataGrid component

You can transform a DataGrid component horizontally and vertically during authoring and run time. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At run time, use the `setSize()` method or applicable properties, such as `width`, `height`, `scaleX`, and `scaleY`. If there is no horizontal scroll bar, column widths adjust proportionally. If column (and therefore, cell) size adjustment occurs, text in the cells may be clipped.

Use styles with the DataGrid component

You can set style properties to change the appearance of a DataGrid component. The DataGrid component inherits styles from the List component. (See "Use styles with the List component" on page 114.)

Set styles for an individual column

A DataGrid object can have multiple columns and you can specify different cell renderers for each column. Each column of a DataGrid is represented by a DataGridColumn object and the DataGridColumn class includes a `cellRenderer` property, for which you can define the CellRenderer for the column.

1 Create a new Flash document (ActionScript 3.0).

2 Drag the DataGrid component to the Library panel.

3 Add the following code to the Actions panel on Frame 1 of the Timeline. This code creates a DataGrid with a long string of text in the third column. At the end, it sets the column's `cellRenderer` property to the name of a cell renderer that renders a multiline cell.

```
/* This is a simple cell renderer example.It invokes
the MultiLineCell cell renderer to display a multiple
line text field in one of a DataGrid's columns. */

import fl.controls.DataGrid;
import fl.controls.dataGridClasses.DataGridColumn;
import fl.data.DataProvider;
import fl.controls.ScrollPolicy;

// Create a new DataGrid component instance.
var aDg:DataGrid = new DataGrid();

var aLongString:String = "An example of a cell renderer class that displays a multiple line 
TextField"
var myDP:Array = new Array();
myDP = [{firstName:"Winston", lastName:"Elstad", note:aLongString, item:100},
    {firstName:"Ric", lastName:"Dietrich", note:aLongString, item:101},
    {firstName:"Ewing", lastName:"Canepa", note:aLongString, item:102},
    {firstName:"Kevin", lastName:"Wade", note:aLongString, item:103},
    {firstName:"Kimberly", lastName:"Dietrich", note:aLongString, item:104},
    {firstName:"AJ", lastName:"Bilow", note:aLongString, item:105},
    {firstName:"Chuck", lastName:"Yushan", note:aLongString, item:106},
    {firstName:"John", lastName:"Roo", note:aLongString, item:107},
];

// Assign the data provider to the DataGrid to populate it.
// Note: This has to be done before applying the cellRenderers.
aDg.dataProvider = new DataProvider(myDP);

/* Set some basic grid properties.
Note: The data grid's row height should reflect
the number of lines you expect to show in the multiline cell.
The cell renderer wil size to the row height.
About 40 for 2 lines or 60 for 3 lines.*/

aDg.columns = ["firstName", "lastName", "note", "item"];
aDg.setSize(430,190);
aDg.move(40,40);
aDg.rowHeight = 40;// Allows for 2 lines of text at default text size.
aDg.columns[0].width = 70;
aDg.columns[1].width = 70;
aDg.columns[2].width = 230;
aDg.columns[3].width = 60;
aDg.resizableColumns = true;
aDg.verticalScrollPolicy = ScrollPolicy.AUTO;
addChild(aDg);
// Assign cellRenderers.
var col3:DataGridColumn = new DataGridColumn();
col3 = aDg.getColumnAt(2);
col3.cellRenderer = MultiLineCell;
```

4 Save the FLA file as MultiLineGrid.fla.

5 Create a new ActionScript file.

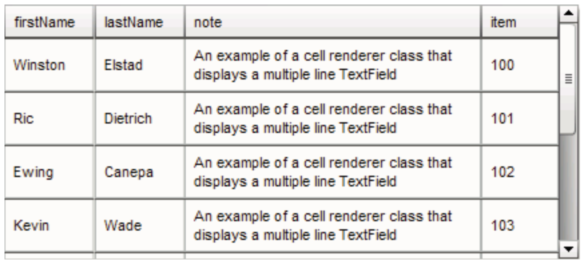
6 Copy the following ActionScript code into the Script window:

```
package {

    import fl.controls.listClasses.CellRenderer;

    public class MultiLineCell extends CellRenderer
    {

        public function MultiLineCell()
        {
            textField.wordWrap = true;
            textField.autoSize = "left";
        }
        override protected function drawLayout():void {
            textField.width = this.width;
            super.drawLayout();
        }
    }
}
```

7 Save the ActionScript file as MultiLineCell.as in the same folder where you saved the MultiLineGrid.fla.

8 Return to the MultiLineGrid.fla application and select Control > Test Movie.

The DataGrid should look like this:



firstName	lastName	note	item
Winston	Elstad	An example of a cell renderer class that displays a multiple line TextField	100
Ric	Dietrich	An example of a cell renderer class that displays a multiple line TextField	101
Ewing	Canepa	An example of a cell renderer class that displays a multiple line TextField	102
Kevin	Wade	An example of a cell renderer class that displays a multiple line TextField	103

DataGrid for the MultiLineGrid.fla application

Set header styles

You can set the text style for a header row by using the `headerTextFormat` style. The following example uses the TextFormat object to set the `headerTextFormat` style to use the Arial font, the color red, a font size of 14, and italic.

1 Create a new Flash file (ActionScript 3.0) document.

2 Drag the DataGrid component to the Stage and give it an instance name of **aDg**.

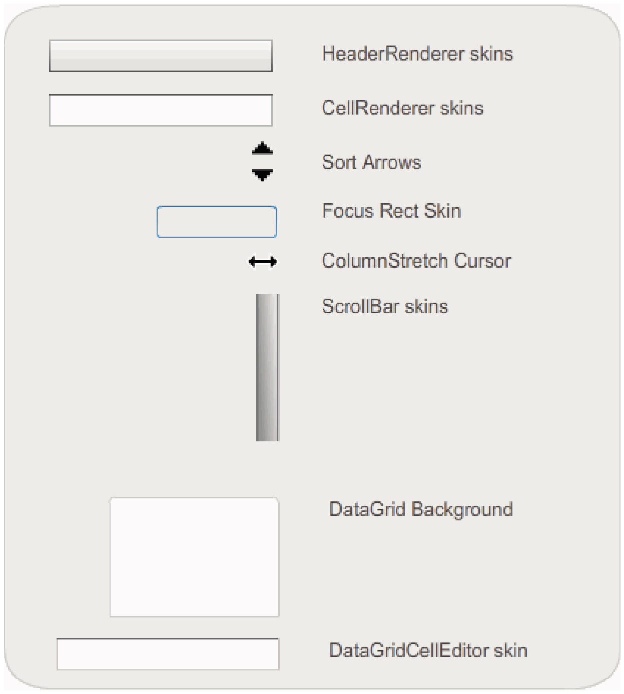
3 Open the Actions panel, select Frame 1 in the main Timeline and enter the following code:

```
import fl.data.DataProvider;
import fl.controls.dataGridClasses.DataGridColumn;

var myDP:Array = new Array();
myDP = [{FirstName:"Winston", LastName:"Elstad"},
    {FirstName:"Ric", LastName:"Dietrich"},
    {FirstName:"Ewing", LastName:"Canepa"},
    {FirstName:"Kevin", LastName:"Wade"},
    {FirstName:"Kimberly", LastName:"Dietrich"},
    {FirstName:"AJ", LastName:"Bilow"},
    {FirstName:"Chuck", LastName:"Yushan"},
    {FirstName:"John", LastName:"Roo"},
];

// Assign the data provider to the DataGrid to populate it.
// Note: This has to be done before applying the cellRenderers.
aDg.dataProvider = new DataProvider(myDP);
aDg.setSize(160,190);
aDg.move(40,40);
aDg.columns[0].width = 80;
aDg.columns[1].width = 80;
var tf:TextFormat = new TextFormat();
tf.size = 14;
tf.color = 0xff0000;
tf.italic = true;
tf.font = "Arial"
aDg.setStyle("headerTextFormat", tf);
```

4 Select Control > Test Movie to run the application.

Use skins with the DataGrid component

The DataGrid component uses the following skins to represent its visual states:



DataGrid skins

The CellRenderer skin is the skin used for the body cells of the DataGrid, while the HeaderRenderer skin is used for the header row. The following procedure changes the background color of the header row but you could use the same process to change the background color of the DataGrid's body cells by editing the CellRenderer skin.

1 Create a new Flash document (ActionScript 3.0).

2 Drag the DataGrid component to the Stage and give it an instance name of **aDg**.

3 Double-click the component to open its palette of skins.

4 Set the zoom control to 400% to enlarge the icons for editing.

5 Double-click the HeaderRenderer skin to open the palette of HeaderRenderer skins.

6 Double-click the Up_Skin to open it in symbol-editing mode and click its background until it is selected and the Fill color picker appears in the Property inspector.

7 Select color #00CC00 using the Fill color picker to apply it to the background of the Up_Skin HeaderRenderer skin.

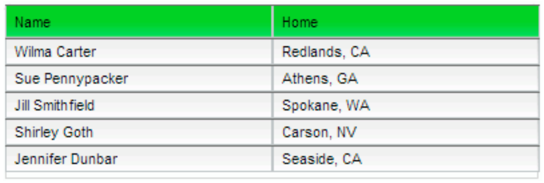
8 Click the Back button at the left side of the edit bar above the Stage to return to document-editing mode.

9 Add the following code to the Actions panel on Frame 1 of the Timeline to add data to the DataGrid:

```
import fl.data.DataProvider;

bldRosterGrid(aDg);
var aRoster:Array = new Array();
aRoster = [
        {Name:"Wilma Carter",Home: "Redlands, CA"},
        {Name:"Sue Pennypacker",Home: "Athens, GA"},
        {Name:"Jill Smithfield",Home: "Spokane, WA"},
        {Name:"Shirley Goth", Home: "Carson, NV"},
        {Name:"Jennifer Dunbar",Home: "Seaside, CA"}
];
aDg.dataProvider = new DataProvider(aRoster);
function bldRosterGrid(dg:DataGrid){
    dg.setSize(400, 130);
    dg.columns = ["Name", "Home"];
    dg.move(50,50);
    dg.columns[0].width = 120;
    dg.columns[1].width = 120;
};
```

10 Select Control > Test Movie to test the application.

The DataGrid should appear as it does in the following illustration with the background of the header row in green.



Name	Home
Wilma Carter	Redlands, CA
Sue Pennypacker	Athens, GA
Jill Smithfield	Spokane, WA
Shirley Goth	Carson, NV
Jennifer Dunbar	Seaside, CA

DataGrid with customized header row background

Customize the Label component

You can transform a Label component horizontally and vertically while authoring and at run time. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. You can also set the `autoSize` authoring parameter; setting this parameter doesn't change the bounding box in the live preview, but the Label is resized. The Label is resized depending on the `wordwrap` parameter. If the parameter is `true`, the Label is resized vertically to fit the text. If the parameter is `false`, the Label is resized horizontally. At run time, use the `setSize()` method. For more information see the `Label.setSize()` method and `Label.autoSize` property in the *ActionScript 3.0 Reference for the Adobe Flash Platform*. Also see "Create an application with the Label component" on page 60.

Use styles with the Label component

You can set style properties to change the appearance of a label instance. All text in a Label component instance must share the same style. The Label component has a `textFormat` style, which has the same attributes as the TextFormat object and allows you to set the same properties for the content of `Label.text` as you can for a regular Flash TextField. The following example sets the color of text in a label to red.

1 Drag the Label component from the Components panel to the Stage and give it an instance name of `a_label`.

2 Click the Parameters tab and replace the value of the text property with the text:

Color me red

3 Select Frame 1 in the main Timeline, open the Actions panel, and enter the following code:

```
/* Create a new TextFormat object, which allows you to set multiple text properties at a
time. */

var tf:TextFormat = new TextFormat();
tf.color = 0xFF0000;
/* Apply this specific text format (red text) to the Label instance. */
a_label.setStyle("textFormat", tf);
```

4 Select Control > Test Movie.

For more information about Label styles, see the Label class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Skins and the Label

The Label component does not have any visual elements to skin.

Customize the List component

You can transform a List component horizontally and vertically while authoring and at run time. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At run time, use the `setSize()` method and applicable properties of the List class such as `height`, `width`, `scaleX`, and `scaleY`.

When a list is resized, the rows of the list shrink horizontally, clipping any text within them. Vertically, the list adds or removes rows as needed. Scroll bars are positioned automatically as needed.

Use styles with the List component

You can set style properties to change the appearance of a List component. The styles specify values for the component's skins and padding when the component is drawn.

The various skin styles allow you to specify different classes to use for the skin. For more information on using skin styles, see "About Skins" on page 98.

The following procedure sets the value of the `contentPadding` style for the List component. Notice that the value of this setting is subtracted from the size of the List to achieve the padding around the content, so you might need to increase the size of the List to prevent text in the List from being cropped.

1 Create a new Flash file (ActionScript 3.0) document.

2 Drag the List component from the Components panel to the Stage and give it an instance name of **aList**.

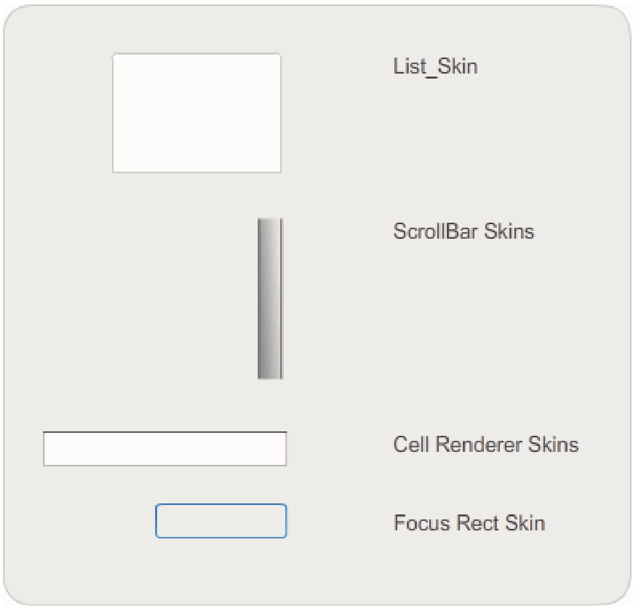
3 Select Frame 1 in the main Timeline, open the Actions panel, and enter the following code, which sets the `contentPadding` style and adds data to the List:

```
aList.setStyle("contentPadding", 5);
aList.setSize(145, 200);
aList.addItem({label:"1956 Chevy (Cherry Red)", data:35000});
aList.addItem({label:"1966 Mustang (Classic)", data:27000});
aList.addItem({label:"1976 Volvo (Xcllnt Cond)", data:17000});
aList.rowCount = aList.length;
```

4 Select Control > Test Movie.

Use skins with the List component

The List component uses the following skins to represent its visual states:

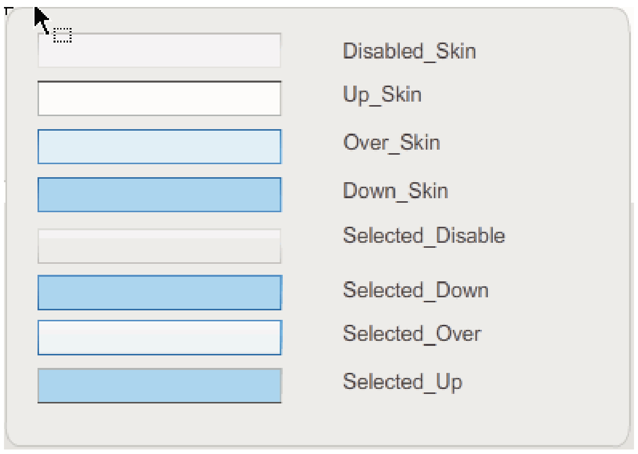


List skins

For more information about skinning the ScrollBar, see "Customize the UIScrollBar component" on page 128. For information on skinning the Focus Rect skin, see "Customize the TextArea component" on page 123

Note: *Changing the ScrollBar skin in one component will change it for all other components that use the ScrollBar.*

Double-click the Cell Renderer skin to open a second palette of skins for the different states of a List cell.



List Cell Renderer skins

You can change the appearance of the List's cells by editing these skins. The following procedure changes the color of the Up skin to change the List's appearance in its normal inactive state.

1 Create a new Flash file (ActionScript 3.0) document.

2 Drag the List component from the Components panel to the Stage and give it an instance name of **aList**.

3 Double-click the List to open its palette of skins.

4 Double-click the Cell Renderer skin to open the palette of Cell Renderer skins.

5 Double-click the Up_Skin skin to open it for editing.

6 Click the fill area of the skin to select it. A Fill color picker should appear in the Property inspector with the skin's current fill color.

7 Select color #CC66FF using the Fill color picker to apply it to the fill of the Up_Skin skin.

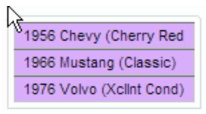
8 Click the Back button at the left side of the edit bar above the Stage to return to document-editing mode.

9 Add the following code to the Actions panel on Frame 1 of the Timeline to add data to the List:

```
aList.setStyle("contentPadding", 5);
aList.setSize(145, 200);
aList.addItem({label:"1956 Chevy (Cherry Red)", data:35000});
aList.addItem({label:"1966 Mustang (Classic)", data:27000});
aList.addItem({label:"1976 Volvo (Xcllnt Cond)", data:17000});
aList.rowCount = aList.length;
```

10 Select Control > Test Movie.

The List should display as it does in the following illustration:


1956 Chevy (Cherry Red
1966 Mustang (Classic)
1976 Volvo (Xcllnt Cond)

List cells with custom Up_Skin color

The framing results from setting the `contentPadding` style.

Customize the NumericStepper component

You can transform a NumericStepper component horizontally and vertically while authoring and at run time. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At run time, use the `setSize()` method or any applicable properties and methods of the NumericStepper class such as `width`, `height`, `scaleX`, and `scaleY`.

Resizing the NumericStepper component does not change the width of the down and up arrow buttons. If the stepper is resized to be greater than the default height, the default behavior pins the arrow buttons to the top and bottom of the component. Otherwise, 9-slice scaling determines how the buttons are drawn. The arrow buttons always appear to the right of the text box.

Styles and the NumericStepper component

You can set the style properties of the NumericStepper component to change its appearance. The styles specify values for the component's skins, padding, and text format when the component is drawn. The `textFormat` style allows you to change the size and appearance of the NumericStepper's value. The various skin styles allow you to specify different classes to use for the skin. For more information on using skin styles, see "About Skins" on page 98.

This procedure uses the `textFormat` style to change the appearance of the value that the NumericStepper displays.

1 Create a new Flash document (ActionScript 3.0).

2 Drag the NumericStepper component from the Component's panel to the Stage and give it an instance name of **myNs**.

3 Add the following code to the Actions panel on Frame 1 of the main Timeline:

```
var tf:TextFormat = new TextFormat();
myNs.setSize(100, 50);
tf.color = 0x0000CC;
tf.size = 24;
tf.font = "Arial";
tf.align = "center";
myNs.setStyle("textFormat", tf);
```

4 Select Control > Test Movie.

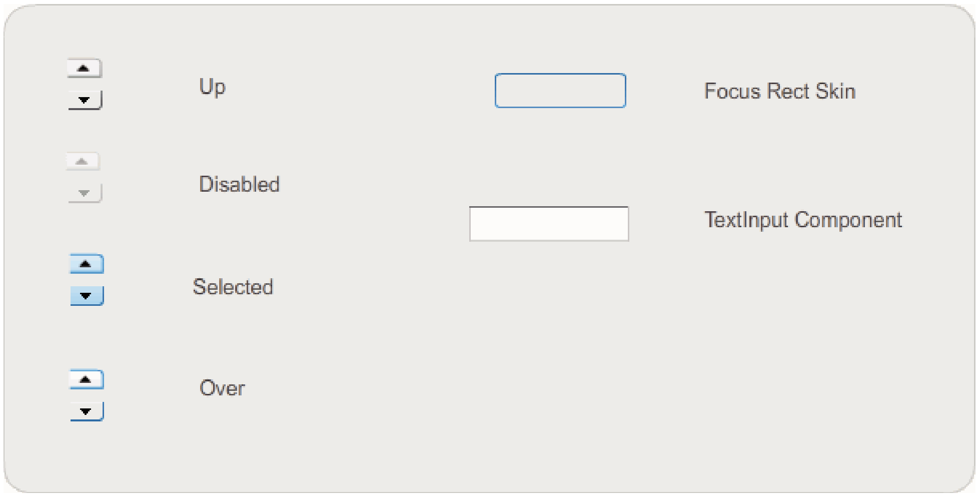
Skins and the NumericStepper component

The NumericStepper component has skins to represent the up, down, disabled, and selected states of its buttons.

If a stepper is enabled, the down and up buttons display their over states when the pointer moves over them. The buttons display their down state when pressed. The buttons return to their over state when the mouse is released. If the pointer moves off the buttons while the mouse is pressed, the buttons return to their original state.

If a stepper is disabled, it displays its disabled state, regardless of user interaction.

A NumericStepper component has the following skins:



NumericStepper skins

1 Create a new FLA file.

2 Drag the NumericStepper component to the Stage.

3 Set the Zoom control to 400% to enlarge the image for editing.

4 Double-click the background of the TextInput skin on the skins panel until you drill down to the Group level and the background color appears in the Fill color picker in the Property inspector.

5 Using the Fill color picker in the Property inspector, select color #9999FF to apply it to the background of the TextInput skin.

6 Click the Back button at the left side of the edit bar above the Stage to return to document-editing mode.

7 Double-click the NumericStepper again to reopen the skins panel.

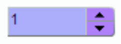
8 Double-click the background of the up arrow button in the Up group until the background is selected and its color appears in the Fill color picker in the Property inspector.

9 Select color #9966FF to apply it to the background of the up arrow button.

10 Repeat steps 8 and 9 for the down arrow in the Up group.

11 Select Control > Test Movie.

The NumericStepper instance should appear as shown in the following illustration:



Customize the ProgressBar component

You can transform a ProgressBar component horizontally and vertically while authoring and at run time. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At run time, use the `setSize()` method or the appropriate properties of the ProgressBar class such as `height`, `width`, `scaleX`, and `scaleY`.

The ProgressBar has three skins: a track skin, a bar skin, and an indeterminate skin. It uses 9-slice scaling to scale the assets.

Styles and the ProgressBar component

You can set style properties to change the appearance of a ProgressBar instance. The ProgressBar's styles specify values for its skin and padding when the component is drawn. The following example enlarges the size of a ProgressBar instance and sets its barPadding style.

1 Create a new FLA file.

2 Drag the ProgressBar component from the Components panel to the Stage and give it an instance name of **myPb.**

3 On Frame 1 of the main Timeline, enter the following code in the Actions Panel:

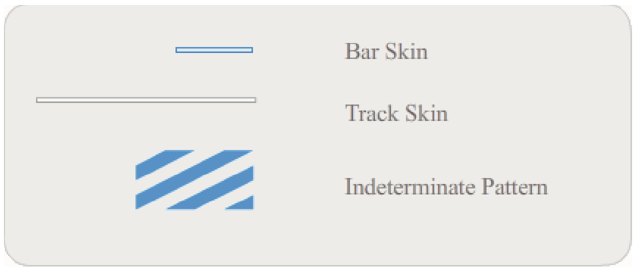
```
myPb.width = 300;
myPb.height = 30;

myPb.setStyle("barPadding", 3);
```

4 Select Control > Test Movie.

For information on setting skin styles, see "About Skins" on page 98.

Skins and the ProgressBar component

The ProgressBar component uses skins to represent the progress bar track, the completed bar, and an indeterminate bar as shown in the following illustration.



ProgressBar skins

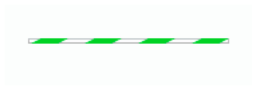
The bar is placed over the track skin, using the barPadding to determine the positioning. The assets are scaled using 9-slice scaling.

The indeterminate bar is used when the ProgressBar instance's `indeterminate` property is set to `true`. The skin is resized vertically and horizontally to fit the size of the ProgressBar.

You can edit these skins to change the appearance of the ProgressBar. For example, the following example changes the color of the indeterminate bar.

1. Create a new FLA file.
2. Drag a ProgressBar component to the stage and double-click it to open its panel of skin icons.
3. Double-click the indeterminate bar skin.
4. Set the zoom control to 400% to enlarge the icon for editing.
5. Double-click one of the diagonal bars, then hold down the Shift key and click on each of the others. The current color appears in the Fill color picker in the Property inspector.
6. Click the Fill color picker in the Property inspector to open it and select color #00CC00 to apply it to the selected diagonal bars.
7. Click the Back button at the left side of the edit bar above the Stage to return to document-editing mode.
8. Select Control > Test Movie.

   The ProgressBar should appear as shown in the following illustration.



Customize the RadioButton component

You can transform a RadioButton component horizontally and vertically while authoring and at run time. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At run time, use the `setSize()` method.

The bounding box of a RadioButton component is invisible and also designates the hit area for the component. If you increase the size of the component, you also increase the size of the hit area.

If the component's bounding box is too small to fit the component label, the label is clipped to fit.

Use styles with the RadioButton component

You can set style properties to change the appearance of a RadioButton. The RadioButton's style properties specify values for its skins, icons, text formatting, and padding when the component is drawn. The RadioButton's styles specify values for its skins and padding for its layout when the component is drawn.

The following example retrieves the `textFormat` style from a CheckBox component and applies it to a RadioButton to make the style of their labels identical.

1 Create a new Flash document (ActionScript 3.0).

2 Drag a CheckBox component to the Stage and give it an instance name of **myCh** in the Property inspector.

3 Drag a RadioButton to the Stage and give it an instance name of **myRb** in the Property inspector.

4 Add the following code to the Actions panel on Frame 1 of the Timeline.

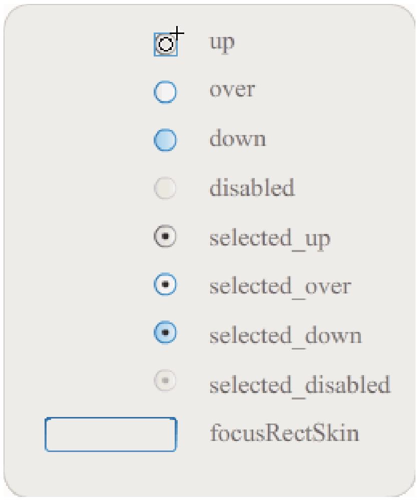
```
var tf:TextFormat = new TextFormat();
tf.color = 0x00FF00;
tf.font = "Georgia";
tf.size = 18;
myCh.setStyle("textFormat", tf);
myRb.setStyle("textFormat", myCh.getStyle("textFormat"));
```

This code sets the `textFormat` style for the CheckBox, then applies it to the RadioButton by calling the `getStyle()` method on the CheckBox.

5 Select Control > Test Movie.

Skins and the RadioButton component

The RadioButton has the following skins which you can edit to change its appearance:



RadioButton skins

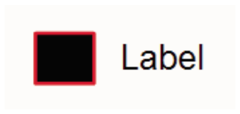
If a RadioButton is enabled and not selected, it displays its over skin when a user moves the pointer over it. When a user clicks a RadioButton, it receives input focus and displays its selected_down skin. When a user releases the mouse, the RadioButton displays its selected_up skin. If a user moves the pointer out of the RadioButton's hit area while pressing the mouse button, the RadioButton redisplays its up skin.

If a RadioButton is disabled, it displays its disabled state, regardless of user interaction.

The following example replaces the selected_up skin that indicates the selected state.

1 Create a new Flash document (ActionScript 3.0).
2 Drag the RadioButton component to the Stage and double-click it to open its palette of skins.
3 Set the zoom control to 800% to enlarge the icon for editing.
4 Double-click the selected_up skin to select it and hit the Delete key to delete it.
5 Select the Rectangle tool on the Tools panel.
6 In the Property inspector, set the line color to red (#FF0000) and the Fill color to black (#000000).
7 Starting at the cross hairs that mark the symbol's registration point (also *origin point* or *zero point*), click and drag the pointer to draw a rectangle.
8 Click the Back button at the left side of the edit bar above the Stage to return to document-editing mode.
9 Select Control > Test Movie.
10 Click the RadioButton to select it.

The RadioButton in the selected state should appear similar to the one in the following illustration.



Customize the ScrollPane component

You can transform a ScrollPane component horizontally and vertically while authoring and at run time. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At run time, use the `setSize()` method or any applicable properties or methods of the ScrollPane class, such as the `height`, `width`, `scaleX`, and `scaleY`.

The ScrollPane component has the following graphical characteristics:

- The registration point (also *origin point* or *zero point*) of its content is in the upper-left corner of the pane.
- When the horizontal scroll bar is turned off, the vertical scroll bar is displayed from top to bottom along the right side of the scroll pane. When the vertical scroll bar is turned off, the horizontal scroll bar is displayed from left to right along the bottom of the scroll pane. You can also turn off both scroll bars.
- If the scroll pane is too small, the content may not display correctly.
- When the scroll pane is resized, the scroll track and scroll box (thumb) expand or contract, and their hit areas are resized. The buttons remain the same size.

Use styles with the ScrollPane component

The style properties of the ScrollPane component specify values for its skins and padding for its layout when the component is drawn. The various skin styles allow you to specify different classes to use for the component's skins. For more information on using skin styles, see "About Skins" on page 98.

1 Create a new Flash document (ActionScript 3.0).
2 Drag a ScrollPane component to the Stage and give it an instance name of **mySp**.

3 Click the Parameters tab in the Property inspector and enter the following value for the `source` parameter: **http://www.helpexamples.com/flash/images/image1.jpg**.

4 On Frame 1 of the main Timeline, add the following code to the Actions panel.

```
mySp.setStyle("contentPadding", 5);
```

Note that the padding is applied between the component's border and its content, on the outside of the scroll bars.

5 Select Control > Test Movie.

Skins and the ScrollPane

The ScrollPane component uses a border and scroll bars for scroll assets. For information on skinning scroll bars see "Use skins with the UIScrollBar component" on page 129.

Customize the Slider component

You can transform a Slider component horizontally while authoring and at run time. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At run time, use the `setSize()` method or any applicable properties of the Slider class, such as the `width` and `scaleX` properties.

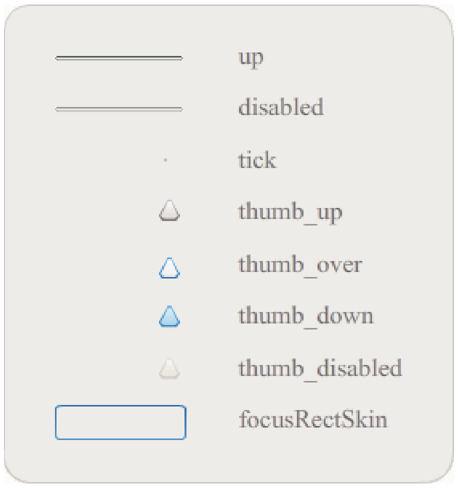
You can only make a slider longer. You cannot increase its height. Flash ignores the `height` property and the height parameter of the `setSize()` method. You can create a vertical slider and make it longer vertically, though.

Styles and the Slider component

The Slider component's styles specify only the classes for its skins and a value for `FocusRectPadding`, which specifies the number of pixels to use for padding between the component's bounding box and its outside boundary. For more information about using skin styles, see "About Skins" on page 98.

Skins and the Slider component

The Slider component uses the following skins, which you can edit to change its appearance.

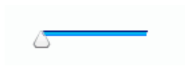


Slider skins

The following example edits the up track to change its color to blue.

1 Create a new Flash document (ActionScript 3.0).
2 Drag the Slider component from the Components panel to the Stage.
3 Double-click the Slider component to open its skins panel.
4 Double-click the up track on its registration mark to open it in symbol-editing mode.
5 Set the zoom control to 800% to enlarge the icon for editing. Notice that the Slider's track consists of three bars.
6 Click the top bar to select it. When it's selected, its color will appear in the Fill color picker in the Property inspector.
7 Using the Fill color picker in the Property inspector, select color #000066 to apply it to the top bar of the Slider track.
8 Click the middle bar of the Slider track to select it. When it's selected, its color appears in the Fill color picker in the Property inspector.
9 Using the Fill color picker in the Property inspector, select color #0066FF to apply it to the middle bar of the Slider track.
10 Click the bottom bar of the Slider track to select it. When it's selected, its color appears in the Fill color picker in the Property inspector.
11 Using the Fill color picker in the Property inspector, select color #00CCFF to apply it to the bottom bar of the Slider track.
12 Click the Back button at the left side of the edit bar above the Stage to return to document-editing mode.
13 Select Control > Test Movie.

The Slider should appear as it does in the following illustration.



Customize the TextArea component

You can transform a TextArea component horizontally and vertically while authoring and at run time. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At run time, use the `setSize()` method or any applicable properties, such as `height`, `width`, `scaleX` and `scaleY` of the TextArea class.

When a TextArea component is resized, the border is resized to the new bounding box. Scroll bars are placed on the bottom and right edges if they are required. The text area is then resized within the remaining area; there are no fixed-size elements in a TextArea component. If the width of the TextArea component is too narrow to display the size of the text, the text is clipped.

Styles and the TextArea component

The TextArea component's styles specify values for its skins, padding, and text format when the component is drawn. The `texFormat` and `disabledTextFormat` styles govern the style of the text that the TextArea displays. For more information about skin style properties, see "Use skins with the TextArea component" on page 124.

The following example sets the `disabledTextFormat` style to change the appearance of text when the TextArea is disabled but the same process applies to setting the `textFormat` style for an enabled TextArea.

1 Create a new Flash file.

2 Drag a TextArea component to the Stage and give it an instance name of **myTa**.

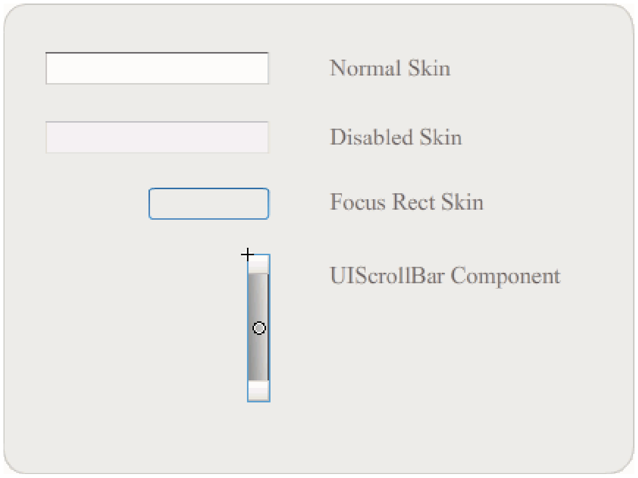
3 Add the following code to the Actions panel on Frame 1 of the main Timeline.

```
var tf:TextFormat = new TextFormat();
tf.color = 0xCC99FF;
tf.font = "Arial Narrow";
tf.size = 24;
myTa.setStyle("disabledTextFormat", tf);
myTa.text = "Hello World";
myTa.setSize(120, 50);
myTa.move(200, 50);
myTa.enabled = false;
```

4 Select Control > Test Movie.

Use skins with the TextArea component

The TextArea component uses the following skins, which you can edit to change its appearance.



TextArea skins

Note: *Changing the ScrollBar skin in one component will change it for all other components that use the ScrollBar.*

The following procedure changes the border colors of the Focus Rect Skin, which appears when the TextArea has focus, and the Normal skin.

1 Create a new Flash file.

2 Drag a TextArea component to the stage and double-click it to open its panel of skin icons.

3 Double-click the Focus Rect Skin.

4 Click the border of the Focus Rect Skin to select it. When it’s selected its current color appears in the Fill color picker in the Property inspector.

5 Click the Fill color picker in the Property inspector to open it and select color #CC0000 to apply it to the border.

6 Click the Back button at the left side of the edit bar above the Stage to return to document-editing mode.

7 Double-click the TextArea component to open its panel of skin icons.

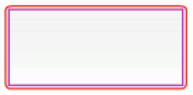
8 Double-click the Normal skin.

9 Select each edge of the Normal skin’s border, one at a time, and set its color to #990099.

10 Click the Back button at the left side of the edit bar above the Stage to return to document-editing mode.

11 Select Control > Test Movie.

When you select the TextArea to begin entering text, its border should appear as shown in the following illustration:



The outer border is the Focus Rect skin and the inner border is the border of the Normal skin.

For information about editing the UIScrollBar skin see "Customize the UIScrollBar component" on page 128.

Customize the TextInput component

You can change the size of a TextInput instance while authoring and at run time. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At run time, use the `setSize()` method or applicable properties of the TextInput class, such as `height`, `width`, `scaleX`, and `scaleY`.

When a TextInput component is resized, the border is resized to the new bounding box. The TextInput component doesn't use scroll bars, but the insertion point scrolls automatically as the user interacts with the text. The text field is then resized within the remaining area; there are no fixed-size elements in a TextInput component. If the TextInput component is too small to display the text, the text is clipped.

Styles and the TextInput component

The TextInput component's styles specify values for its skins, padding, and text formatting when the component is drawn. The `texFormat` and `disabledTextFormat` styles govern the style of the text that displays in the component. For more information about skin style properties, see "Skins and the TextInput component" on page 126.

The following example sets the `textFormat` style to set the font, size, and color of the text that displays in the TextInput component. The same process applies to setting the `disabledTextFormat` style that is applied when the component is disabled.

1 Create a new Flash document (ActionScript 3.0).

2 Drag a TextInput component to the Stage and give it an instance name of **myTi**.

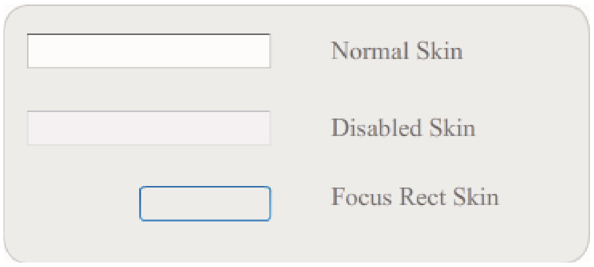
3 Add the following code to the Actions panel on Frame 1 of the main Timeline.

```
var tf:TextFormat = new TextFormat();
tf.color = 0x0000FF;
tf.font = "Verdana";
tf.size = 30;
tf.align = "center";
tf.italic = true;
myTi.setStyle("textFormat", tf);
myTi.text = "Enter your text here";
myTi.setSize(350, 50);
myTi.move(100, 50);
```

4 Select Control > Test Movie.

Skins and the TextInput component

The TextInput component uses the following skins, which you can edit to change its appearance:



TextInput caption

The following procedure changes the border and background colors of a TextInput component:

1. Create a new Flash file.
2. Drag a TextInput component to the Stage and double-click it to open its panel of skins.
3. Double-click the Normal skin.
4. Set the zoom control to 800% to enlarge the icon for editing.
5. Select each edge of the Normal skin's border, one at a time, and set its color to #993399 to apply it.
6. Double-click the background until its color appears in the Fill color picker in the Property inspector. Select color #99CCCC to apply it to the background.
7. Click the Back button at the left side of the edit bar above the Stage to return to document-editing mode.
8. Select Control > Test Movie.

   The TextInput component should appear as shown in the following illustration:

Customize the TileList component

You can transform the TileList component horizontally and vertically while authoring and at run time. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At run time, use the `setSize()` method or the appropriate properties such as the `width`, `height`, `columnCount`, `rowCount`, `scaleX`, and `scaleY`. The ScrollBar, which TileList contains, scales with the list box.

Styles and the TileList component

The TileList's styles specify values for its skins, padding, and text formatting when the component is drawn. The `texFormat` and `disabledTextFormat` styles govern the style of the text that displays in the component. For more information about the skin styles, see "Use skins with the TileList component" on page 127.

The following example calls the `setRendererStyle()` method using the `textFormat` style to set the font, size, color, and text attributes of the labels that display in a TileList instance. The same process applies also to setting the `disabledTextFormat` style that is applied when the `enabled` property is set to `false`.

1. Create a new Flash document (ActionScript 3.0).

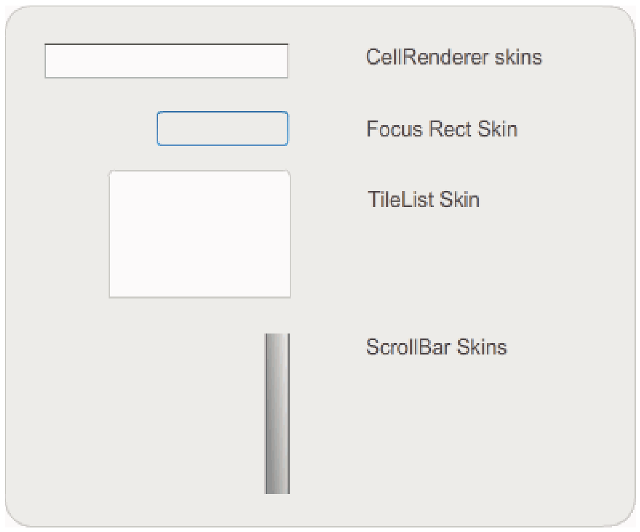
2 Drag the TileList component to the Stage and give it an instance name of **myTl**.

3 Add the following code to the Actions panel on Frame 1 of the Timeline.

```
myTl.setSize(100, 100);
myTl.addItem({label:"#1"});
myTl.addItem({label:"#2"});
myTl.addItem({label:"#3"});
myTl.addItem({label:"#4"});
var tf:TextFormat = new TextFormat();
tf.font = "Arial";
tf.color = 0x00FF00;
tf.size = 16;
tf.italic = true;
tf.bold = true;
tf.underline = true;
tf.align = "center";
myTl.setRendererStyle("textFormat", tf);
```

Use skins with the TileList component

The TileList component has a TileList skin, a CellRenderer skin, and a ScrollBar skin. You can edit these skins to change the TileList's appearance:



TileList skins

Note: *Changing the ScrollBar skin in one component will change it in all other components that use the ScrollBar.*

The following procedure changes the color of the TileList's CellRenderer Selected_Up skin.

1 Create a Flash document (ActionScript 3.0).

2 Drag the TileList component to the Stage and double-click it to open its panel of skins.

3 Double-click the CellRenderer skin, then double-click the Selected_Up skin, and then click the rectangular background.

4 Select color #99FFFF using the Fill color picker in the Property inspector to apply it to the Selected_Up skin.

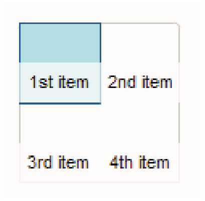
5 Click the Back button at the left side of the edit bar above the Stage until you return to document-editing mode.

6 On the Parameters tab of the Property inspector, double-click the second column of the dataProvider row to open the Values dialog box. Add items with the following labels: 1st item, 2nd item, 3rd item, 4th item.

7 Select Control > Test Movie.

8 Click one of the cells in the TileList to select it, and then move the mouse away from the selected cell.

The selected cell should appear as it does in the following illustration:



TileList component with modfied Selected_Up skin color

Customize the UILoader component

You can transform a UILoader component horizontally and vertically while authoring and at run time. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At run time, use the `setSize()` method or the appropriate properties such as the `width`, `height`, `scaleX`, and `scaleY` properties.

The sizing behavior of the UILoader component is controlled by the `scaleContent` property. When `scaleContent` is `true`, the content is scaled to fit within the bounds of the loader (and is rescaled when `setSize()` is called). When `scaleContent` is `false`, the size of the component is fixed to the size of the content and `setSize()` and the sizing properties have no no effect.

The UILoader component has no user interface elements to which you can apply styles or skins.

Customize the UIScrollBar component

You can transform a UIScrollBar component horizontally and vertically while authoring and at run time. However, a vertical UIScrollBar does not allow you to modify the width, and a horizontal UIScrollBar does not allow you to modify the height. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At run time, use the `setSize()` method or any applicable properties of the UIScrollBar class such as the `width`, `height`, `scaleX`, and `scaleY` properties.

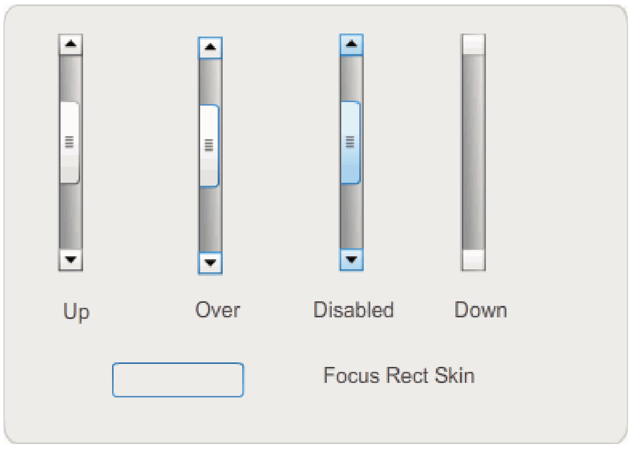
***Note:** If you use the `setSize()` method, you can change only the width of a horizontal scroll bar or the height of a vertical scroll bar. At the time of authoring you can set the height of a horizontal scroll bar or the width of a vertical scroll bar, but the values will be reset when the movie is published. Only the dimension of a scroll bar that corresponds to its length can be changed.*

Use styles with the UIScrollBar component

The UIScrollBar component's styles specify only the classes for its skins and a value for FocusRectPadding, which specifies the number of pixels to use for padding between the component's bounding box and its outside boundary. For more information about using skin styles, see "About Skins" on page 98.

Use skins with the UIScrollBar component

The UIScrollBar component uses the following skins.



UIScrollBar skins

Both horizontal and vertical scroll bars use the same skins; when displaying a horizontal scroll bar the UIScrollBar component rotates the skins as appropriate.

Note: *Changing the ScrollBar skin in one component will change it in all other components that use the ScrollBar.*

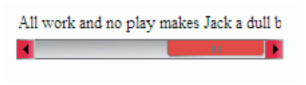
The following example demonstrates how to change the color of the UIScrollBar's thumb and arrow buttons.

1. Create a new Flash document (ActionScript 3.0).
2. Drag the UIScrollBar component to the Stage and give it an instance name of **mySb**. In the Parameters tab, set the direction to horizontal.
3. Double-click the scroll bar to open its panel of skins.
4. Click the Up skin to select it.
5. Set the zoom control to 400% to enlarge the icon for editing.
6. Double-click the background of the right arrow (or up arrow for a vertical scroll bar) until the background is selected and its color appears in the Fill color picker in the Property inspector.
7. Select color #CC0033 to apply it to the button background.
8. Click the Back button at the left side of the edit bar above the Stage until you return to document-editing mode.
9. Repeat steps 6, 7, and 8 for the thumb and the left-hand arrow (or down arrow for a vertical scroll bar) elements.
10. Add the following code to the Actions panel on Frame 1 of the Timeline to attach the scroll bar to a TextField.

```
var tf:TextField = new TextField();
addChild(tf);
tf.x = 150;
tf.y = 100;
mySb.width = tf.width = 200;
tf.height = 22;
tf.text = "All work and no play makes Jack a dull boy. All work and no play makes Jack a 
dull boy. All . . .";
mySb.y = tf.y + tf.height;
mySb.x = tf.x + tf.width;x
mySb.scrollTarget = tf;
```

11 Select Control > Test Movie.

The UIScrollBar component should appear as it does in the following illustration.



Horizontal ScrollBar with thumb and left and right arrows in red

Chapter 6: Using the FLVPlayback Component

The FLVPlayback component lets you easily include a video player in your Adobe Flash CS5 Professional application to play progressively downloaded video files over HTTP, or streaming video files from Adobe's Macromedia Flash Media Server, or from a Flash Video Streaming Service (FVSS).

With the release of Adobe Flash Player 9 Update 3 (version 9.0.115.0 or later), significant improvements for playing video content in Flash Player have been incorporated. This update includes changes to the FLVPlayback component that take advantage of the end user's system video hardware to provide better video playback performance. The changes to the FLVPlayback component also increase the fidelity of video files displayed in full-screen mode.

Additionally, Flash Player 9 Update 3 improves the functionality of the FLVPlayback component by adding support for the high-definition MPEG-4 video formats that utilize industry standard H.264 encoding. These formats include MP4, M4A, MOV, MP4V, 3GP, and 3G2.

***Note:** Protected MP4 files—such as those downloaded from Apple® iTunes® or digitally encrypted by FairPlay®—are not supported.*

The easy-to-use FLVPlayback component has the following characteristics and benefits:

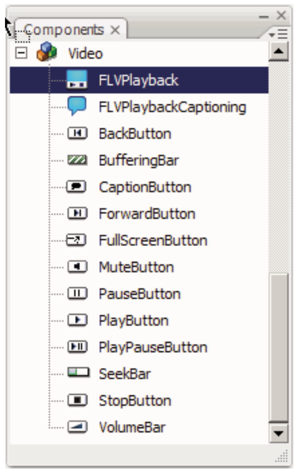
- Can be dragged to the Stage and implemented quickly and successfully
- Supports full-screen size
- Provides a collection of predesigned *skins* that allow you to customize the appearance of its playback controls
- Allows you to select color and alpha values for predesigned skins
- Allows advanced users to create their own skins
- Provides live preview during authoring
- Provides layout properties to keep the video file centered when resizing
- Allows start of playback when enough of a progressively downloaded video file has downloaded
- Provides cue points that allow you to synchronize your video with text, graphics, and animation
- Maintains a reasonably sized SWF file

Use the FLVPlayback component

Using the FLVPlayback component consists of putting it on the Stage and specifying a video file for it to play. In addition, you can also set various parameters that govern its behavior and describe the video file.

The FLVPlayback component also includes an ActionScript application programming interface (API). The API includes the following classes, which are fully described in the *ActionScript 3.0 Reference for the Adobe Flash Platform*: CuePointType, FLVPlayback, FLVPlaybackCaptioning, NCManager, NCManagerNative, VideoAlign, VideoError, VideoPlayer, VideoState, and several event classes - AutoLayoutEvent, LayoutEvent, MetadataEvent, SkinErrorEvent, SoundEvent, VideoEvent, and VideoProgressEvent.

The FLVPlayback component includes the FLV Playback Custom UI components. The FLVPlayback component is a combination of the display area, or video player, in which you view the video file and the controls that allow you to operate it. The FLV Playback Custom UI components provide control buttons and mechanisms that you can use to play, stop, pause, and otherwise control the video file. These controls include the BackButton, BufferingBar, CaptionButton (for FLVPlaybackCaptioning), ForwardButton, FullScreenButton, MuteButton, PauseButton, PlayButton, PlayPauseButton, SeekBar, StopButton, and VolumeBar. The FLVPlayback component and the FLV Playback Custom UI controls appear in the Components panel, as shown in the following figure:



FLVPlayback components in the Components panel

The process of adding playback controls to the FLVPlayback component is called *skinning*. The FLVPlayback component has an initial default skin, SkinOverAll.swf, that provides the play, stop, back, forward, seekbar, mute, volume, full screen, and captioning controls. To change this skin, you have the following choices:

- Select from a collection of predesigned skins
- Create a custom skin and add it to the collection of predesigned skins
- Select individual controls from the FLV Playback Custom UI components and customize them

  When you select a predesigned skin, you can choose the skin color and alpha values separately, either during authoring or at run time. For more information, see “Select a predesigned skin” on page 149.

  After you select a different skin, the selected skin becomes the new default skin.

  For more information about selecting or creating a skin for the FLVPlayback component, see “Customize the FLVPlayback component” on page 149.

Create an application with the FLVPlayback component

You can include the FLVPlayback component in your application in the following ways:

- Drag the FLVPlayback component from the Components panel to the Stage, and specify a value for the `source` parameter.
- Use the Video Import wizard to create the component on the Stage, and customize it by selecting a skin.
- Use the `FLVPlayback()` constructor to dynamically create an FLVPlayback instance on the Stage, assuming the component is in the library.

***Note:** If you create an FLVPlayback instance with ActionScript, you must also assign a skin to it by setting the `skin` property with ActionScript. When you apply a skin this way, it is not automatically published with the SWF file. You must copy both the application SWF file and the skin SWF file to your application server or the skin SWF file won't be available when you run the application.*

Drag the FLVPlayback component from the Components panel

1 In the Components panel, click the Plus (+) button to open the video entry.

2 Drag the FLVPlayback component to the Stage.

3 With the FLVPlayback component selected on the Stage, locate the Value cell for the `source` parameter on the Parameters tab of the Component inspector, and enter a string that specifies one of the following:

- A local path to a video file
- A URL to a video file
- A URL to a synchronized Multimedia Integration Language (SMIL) file that describes how to play a video file

  For information on how to create a SMIL file to describe one or more FLV files, see "Use a SMIL file" on page 159.

4 On the Parameters tab in the Component inspector, with the FLVPlayback component selected on the Stage, click the Value cell for the `skin` parameter.

5 Click the magnifying glass icon to open the Select Skin dialog box.

6 Select one of the following options:

- From the drop-down Skin list, select one of the predesigned skins to attach a set of playback controls to the component.
- If you created a custom skin, select Custom Skin URL from the pop-up menu, and enter, in the URL box, the URL for the SWF file that contains the skin.
- Select None, and drag individual FLV Playback Custom UI components to the Stage to add playback controls.

  ***Note:** In the first two cases, a preview of the skin appears in the viewing pane above the pop-up menu. You can use the Color picker to change the color of the skin.*

  To change the color of a custom UI control, you must customize it. For more information on using custom UI controls, see "Skin FLV Playback Custom UI components individually" on page 150.

7 Click OK to close the Select Skin dialog box.

8 Select Control > Test Movie to execute the SWF file and start the video.

The following procedure uses the Video Import wizard to add an FLVPlayback component:

Use the Video Import wizard:

1 Select File > Import > Import Video.

2 Indicate the location of the video file by selecting one of the following options:

- On my local computer
- Already deployed to a web server, Flash Video Streaming Service, or Flash Media Server

3 Depending on your choice, enter either the path or the URL that specifies the location of the video file, then click Next.

4 If you selected a file path, you'll see a Deployment dialog box next in which you can select one of the options listed to specify how you would like to deploy your video:

- Progressive download from a standard web server

- Stream from Flash Video Streaming Service
- Stream from Flash Media Server
- Embed video in a SWF file and play in the Timeline

  Important: *Do not select the Embed Video option. The FLVPlayback component plays only external streaming video. This option will not place an FLVPlayback component on the Stage.*

5 Click Next.

6 Select one of the following options:

- From the drop-down Skin list, select one of the predesigned skins to attach a set of playback controls to the component.
- If you created a custom skin for the component, select Custom Skin URL from the pop-up menu, and enter the URL for the SWF file that contains the skin in the URL box.
- Select None, and drag individual FLV Playback Custom UI components to the Stage to add playback controls.

  Note: *In the first two cases, a preview of the skin appears in the viewing pane above the pop-up menu.*

7 Click OK to close the Select Skin dialog box.

8 Read the Finish Video Import dialog box to note what happens next, and then click Finish.

9 If you have not saved your FLA file, a Save As dialog box appears.

10 Select Control > Test Movie to execute the SWF, and start the video.

The following procedure adds the FLVPlayback component using ActionScript.

Create an instance dynamically using ActionScript:

1 Drag the FLVPlayback component from the Components panel to the Library panel (Window > Library).

2 Add the following code to the Actions panel on Frame 1 of the Timeline. Change *install_drive* to the drive on which you installed Flash and modify the path to reflect the location of the Skins folder for your installation:

On a Windows computer:

```
import fl.video.*;
var my_FLVPlybk = new FLVPlayback();
my_FLVPlybk.x = 100;
my_FLVPlybk.y = 100;
addChild(my_FLVPlybk);
my_FLVPlybk.skin = "file:///install_drive|/Program Files/Adobe/Adobe Flash 
CS5/en/Configuration/FLVPlayback Skins/ActionScript 3.0/SkinOverPlaySeekMute.swf"
my_FLVPlybk.source = "http://www.helpexamples.com/flash/video/water.flv";
```

On a Macintosh computer:

```
import fl.video.*;
var my_FLVPlybk = new FLVPlayback();
my_FLVPlybk.x = 100;
my_FLVPlybk.y = 100;
addChild(my_FLVPlybk);
my_FLVPlybk.skin = "file:///Macintosh HD:Applications:Adobe Flash 
CS5:Configuration:FLVPlayback Skins:ActionScript 3.0SkinOverPlaySeekMute.swf"
my_FLVPlybk.source = "http://www.helpexamples.com/flash/video/water.flv";
```

Note: *If you do not set the* `source` *and* `skin` *properties, the generated movie clip appears to be empty.*

3 Select Control > Test Movie to execute the SWF file and start the video file.

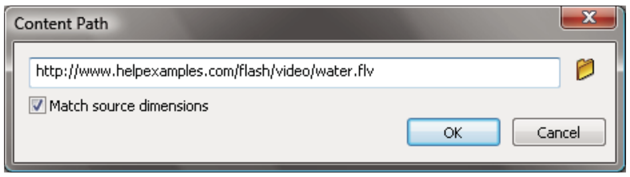
FLVPlayback component parameters

For each instance of the FLVPlayback component, you can set the following parameters in the Component inspector or the Property inspector: `align`, `autoPlay`, `cuePoints`, `preview`, `scaleMode`, `skin`, `skinAutoHide`, `skinBackgroundAlpha`, `skinBackgroundColor`, `source`, and `volume`. Each of these parameters has a corresponding ActionScript property of the same name. When you assign a value to these parameters you are setting the initial state of the property in the application. Setting the property in ActionScript overrides the value you set in the parameter. For information on the possible values for these parameters, see the FLVPlayback class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Specify the FLVPlayback source parameter

The `source` parameter lets you specify the name and location of the video file, both of which inform Flash how to play the file.

Open the Content Path dialog box by double-clicking the Value cell for the `source` parameter in the Component inspector.


Content Path

http://www.helpexamples.com/flash/video/water.flv

☑ Match source dimensions

FLVPlayback Content Path dialog box

The Content Path dialog box provides a check box, Match Source FLV Dimensions, that specifies whether the FLVPlayback instance on the Stage should match the dimensions of the source video file. The source video file contains preferred height and width dimensions for playing. If you select this option, the dimensions of the FLVPlayback instance are resized to match these preferred dimensions.

The source

Enter the URL or local path for either the video file or an XML file that describes how to play the video file. If you do not know the exact location of a video file, click the folder icon to open a Browser dialog box to help you find the correct location. When browsing for a video file, if it is at or below the location of the target SWF file, Flash automatically makes the path relative to that location so you can serve it from a web server. Otherwise, the path is an absolute Windows or Macintosh path. To enter the name of a local XML file, type the path and name.

If you specify an HTTP URL, the video file plays as a progressive download. If you specify a URL that is an RTMP URL, the video file streams from Flash Media Server or FVSS. A URL to an XML file could also be a streaming video file from Flash Media Server or FVSS.

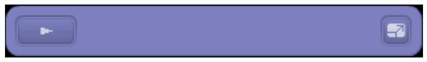
Important:

You can also specify the location of a SMIL file that describes how to play multiple video file streams for multiple bandwidths. The file uses the Synchronized Multimedia Integration Language (SMIL) to describe the FLV files. For a description of the SMIL file, see “Use a SMIL file” on page 159.

You can also specify the name and location of the video file using the ActionScript `FLVPlayback.source` property and the `FLVPlayback.play()` and `FLVPlayback.load()` methods. These three alternatives take precedence over the `source` parameter in the Component inspector. For more information, see the `FLVPlayback.source`, `FLVPlayback.play()`, and `FLVPlayback.load()` entries for the FLVPlayback class in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Full-screen support

The ActionScript 3.0 version of the FLVPlayback component supports full-screen mode, which requires Flash Player 9.0.28.0, or later, and also that the HTML is set up correctly for full-screen viewing. Some predesigned skins include a toggle button to switch full-screen mode on and off. The FullScreenButton icon appears on the right side of the control bar in the following illustration.



Full-screen icon on control bar

Full-screen support occurs only if the `fullScreenTakeOver` property is set to `true`, which it is by default.

Full-screen support can occur with or without hardware-acceleration support. For information about hardware-acceleration support, see “Hardware acceleration” on page 139.

To implement full-screen support for FLVPlayback:

1 Add the FLVPlayback component to your application and assign a video file to it.

2 Select a skin for the FLVPlayback component that has the full-screen button (e.g., SkinUnderPlaySeekFullscreen.swf) or add the FullScreenButton user-interface component to the FLVPlayback component from the Video section in the Components panel.

3 Select File > Publish Settings.

4 In the Publish Settings dialog, click the HTML tab.

5 On the HTML tab, select Flash With Full Screen Support from the Template pop-up menu.

6 Also on the HTML tab, select the Detect Flash Version check box and specify a version of 9.0.28 or later, depending on the version of Flash Player that you are using.

7 Select the Formats tab and be sure that both the Flash (.swf) and HTML (.html) options are selected. You can replace the default file names.

8 Click Publish and then click OK.

   As an alternative to step 7, you can click OK and then select File > Publish Preview > Default - (HTML) to automatically open the exported HTML file in your default browser. Otherwise, open the exported HTML file with your browser to test the full-screen option.

   To add the FLVPlayback component with full-screen support to your web page, open the exported HTML file and copy the code that embeds the SWF file into the HTML file for your web page. This code should look similar to the following example:

```
//from the <head> section

<script language="javascript"> AC_FL_RunContent = 0; </script>
<script language="javascript"> DetectFlashVer = 0; </script>
<script src="AC_RunActiveContent.js" language="javascript"></script>
<script language="JavaScript" type="text/javascript">
<!--
// -----------------------------------------------------------------------------
// Globals
// Major version of Flash required
var requiredMajorVersion = 9;
// Minor version of Flash required
var requiredMinorVersion = 0;
// Revision of Flash required
var requiredRevision = 28;
// -----------------------------------------------------------------------------
// -->
</script>

//and from the <body> section

<script language="JavaScript" type="text/javascript">
<!--
if (AC_FL_RunContent == 0 || DetectFlashVer == 0) {
    alert("This page requires AC_RunActiveContent.js.");
} else {
    var hasRightVersion = DetectFlashVer(requiredMajorVersion,
        requiredMinorVersion, requiredRevision);
    if(hasRightVersion) { // if we&apos;ve detected an acceptable version
        // embed the Flash movie
        AC_FL_RunContent(
            &apos;codebase&apos;, &apos;http://download.macromedia.com/pub/
                shockwave/cabs/flash/swflash.cab#version=9,0,28,0&apos;,
            &apos;width&apos;, &apos;550&apos;,
            &apos;height&apos;, &apos;400&apos;,
            &apos;src&apos;, &apos;fullscreen&apos;,
            &apos;quality&apos;, &apos;high&apos;,
            &apos;pluginspage&apos;, &apos;http://www.macromedia.com/go/
                getflashplayer&apos;,
            &apos;align&apos;, &apos;middle&apos;,
            &apos;play&apos;, &apos;true&apos;,
            &apos;loop&apos;, &apos;true&apos;,
            &apos;scale&apos;, &apos;showall&apos;,
            &apos;wmode&apos;, &apos;window&apos;,
            &apos;devicefont&apos;, &apos;false&apos;,
            &apos;id&apos;, &apos;fullscreen&apos;,
            &apos;bgcolor&apos;, &apos;#ffffff&apos;,
            &apos;name&apos;, &apos;fullscreen&apos;,
            &apos;menu&apos;, &apos;true&apos;,
            &apos;allowScriptAccess&apos;,&apos;sameDomain&apos;,
            &apos;allowFullScreen&apos;,&apos;true&apos;,
            &apos;movie&apos;, &apos;fullscreen&apos;,
            &apos;salign&apos;, &apos;&apos; ); //end AC code
    } else { // Flash is too old or we can&apos;t detect the plug-in.
        var alternateContent = &apos;Alternative HTML content should be placed
                here.&apos;
            + &apos;This content requires Adobe Flash Player.&apos;
            + &apos;<a href=http://www.macromedia.com/go/getflash/>Get Flash</a>
                    &apos;;
        document.write(alternateContent); // Insert non-Flash content.
    }
}
// -->
</script>
<noscript>
    // Provide alternative content for browsers that do not support scripting
    // or for those that have scripting disabled.
    Alternative HTML content should be placed here. This content requires Adobe Flash Player.
    <a href="http://www.macromedia.com/go/getflash/">Get Flash</a>
</noscript>
```

As an alternative, you can use the exported HTML file as the template for your web page and add your other content to it. If you do this, however, change the name of the HTML file so that you don't accidentally overwrite it in the event that you later export the FLVPlayback HTML file from Flash again.

In any case, you must also upload to your web server the AC_RunActiveContent.js file that is exported to the same folder as the HTML file.

ActionScript support for full-screen mode includes the `fullScreenBackgroundColor`, `fullScreenSkinDelay`, and `fullScreenTakeOver` properties and the `enterFullScreenDisplayState()` method. For information on these ActionScript elements, see the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Use enterFullScreenDisplayState()

You can also invoke full-screen mode by calling the `enterFullScreenDisplayState()` ActionScript method, as shown in the following example.

```
function handleClick(e:MouseEvent):void {
    myFLVPlybk.enterFullScreenDisplayState();
}
myButton.addEventListener(MouseEvent.CLICK, handleClick);
```

In this example, full-screen mode is *not* invoked by clicking the full-screen toggle button on an FLVPlayback skin, but rather, by clicking a button (MyButton) that the creator of the web page included to invoke full-screen mode. Clicking the button triggers the `handleClick` event handler, which calls the `enterFullScreen DisplayState()` method.

The `enterFullScreenDisplayState()` method sets the `Stage.displayState` property to `StageDisplayState.FULL_SCREEN`, and therefore carries the same restrictions as the `displayState` property. For more information on the `enterFullScreenDisplayState()` method and the `Stage.displayState` property, see the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Hardware acceleration

Flash Player 9.0.115.0 and later versions include code that takes advantage of available video hardware to improve the performance and fidelity of FLV files that FLVPlayback plays in full-screen mode. If the prerequisites are met and the `fullScreenTakeOver` property is set to `true`, Flash Player uses hardware acceleration to scale the video file, rather than scaling it through software. If the FLVPlayback component runs in an earlier version of Flash Player, or if the prerequisites for hardware acceleration do not exist, Flash Player scales up the video file itself, as it did previously.

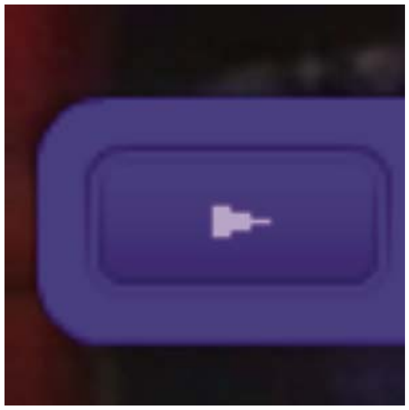
To take advantage of hardware acceleration for full-screen support, your computer must have a DirectX 7-compatible video card with 4 MB or more of VRAM (video RAM). This hardware support is available in Windows 2000 or Mac OS X 10.2, and later versions of those operating systems. Direct X® provides APIs that comprise an interface between software and the video hardware to accelerate three-dimensional and two-dimensional graphics, among other things.

To take advantage of hardware-acceleration mode, you must also invoke full-screen mode in one of the following ways:

- Using the full-screen toggle button on an FLVPlayback skin
- Using the FullScreenButton video control
- Using the ActionScript `enterFullScreenDisplayState()` method. For more information, see "Use enterFullScreenDisplayState()" on page 138.

  If you invoke full-screen mode by setting the `Stage.displayState` property to `StageDisplayState.FULLSCREEN`, FLVPlayback does not use hardware acceleration, even if the video hardware and memory are available.

  One consequence of using hardware acceleration for full-screen support is that the FLVPlayback skins are scaled along with the video player and the video file. The following image shows the effect of full-screen mode with hardware acceleration on the FLVPlayback skin, a detail of which is shown here at full resolution.


Full-screen mode on a 1600 x 1200 monitor with a 320x240 pixel video

This image shows the result of using full-screen mode on a 1600 x 1200 monitor with a video file that has a width of 320 and a height of 240, which are the default FLVPlayback dimensions. The distortion effect on the skin is more pronounced on FLV files with smaller dimensions or on a larger monitor. Conversely, the distortion effect is less pronounced on larger FLV files or on smaller monitors. For example, changing from 640 x 480 to 1600 x 1200 still increases the size of the skin, but it looks less distorted.

You can set the `skinScaleMaximum` property to limit the scaling of the FLVPlayback skin. The default value is 4.0, or 400%. Limiting the scaling of the skin, however, requires a combination of hardware and software to scale the FLV and this can adversely affect performance on FLVs with large dimensions that are encoded at a high bit rate. If the video is large (640 pixels wide or more, 480 pixels tall or more, for example), you should not set `skinScaleMaximum` to a small value because it could cause large performance problems on large display monitors. The `skinScaleMaximum` property allows you to manage the trade offs between performance and quality and the appearance of a large skin.

Exit full-screen mode

To exit full-screen mode, click the full-screen button again or press the Esc key.

Setting the following properties and calling the following methods can cause layout changes that cause the FLVPLayback component to exit full-screen mode: `height`, `registrationHeight`, `registrationWidth`, `registrationX`, `registrationY`, `scaleX`, `scaleY`, `width`, `x`, `y`, `setScale()` ,or `setSize()`.

If you set the `align` or `scaleMode` properties, FLVPlayback sets them to `center` and `maintainAspectRatio` until full-screen mode is exited.

Changing the value of the `fullScreenTakeOver` property from `true` to `false` when you are using full-screen, hardware-acceleration mode also causes Flash to exit full-screen mode.

Layout alignment for playing multiple video files

ActionScript 3.0 FLVPlayback has an `align` property that specifies whether the video file should be centered when it is resized or positioned at the top, bottom, left, or right of the component. In addition to the component's `x`, `y`, `width`, and `height` properties, the ActionScript 3.0 component also has `registrationX`, `registrationY`, `registrationWidth`, and `registrationHeight` properties. Initially, these match the `x`, `y`, `width`, and `height` properties. When loading subsequent video files, automatic re-layout does not change these so the new video file can be centered in the same place. If `scaleMode = VideoScaleMode.MAINTAIN_ASPECT_RATIO`, subsequent FLV files can fit into the component's original dimensions rather than causing the component's width and height to be altered.

Automatic playing of progressively downloaded video files

When loading a progressively downloaded video file, FLVPlayback starts playing the video file only when enough of it has downloaded so that it can play the video file from start to finish.

If you want to play the video file before enough of it has downloaded, call the `play()` method without any parameters.

If you want to return to the state of waiting for enough of the video file to download, call the `pause()` method and then call the `playWhenEnoughDownloaded()` method.

Use cue points

A cue point is a point at which the video player dispatches a `cuePoint` event while a video file plays. You can add cue points to an FLV file at times that you want an action to occur for another element on the web page. You might want to display text or a graphic, for example, or synchronize with a Flash animation, or affect the playing of the FLV file by pausing it, seeking a different point in the video, or switching to a different FLV file. Cue points allow you to receive control in your ActionScript code to synchronize points in your FLV file with other actions on the web page.

There are three types of cue points: navigation, event, and ActionScript. The navigation and event cue points are also known as *embedded* cue points because they are embedded in the FLV file stream and in the FLV file's metadata packet.

A *navigation cue point* allows you to seek a particular frame in the FLV file because it creates a keyframe within the FLV file as near as possible to the time that you specify. A *keyframe* is a data segment that occurs between image frames in the FLV file stream. When you seek a navigation cue point, the component seeks to the keyframe and starts the `cuePoint` event.

An *event cue point* enables you to synchronize a point in time within the FLV file with an external event on the web page. The `cuePoint` event occurs precisely at the specified time. You can embed navigation and event cue points in an FLV file using either the Video Import wizard or the Adobe Media Encoder. For more information on the Video Import wizard and the Adobe Media Encoder, see "Working with Video," in *Using Flash*.

An *ActionScript cue point* is an external cue point that you can add either through the component's Flash Video Cue Points dialog box or through the `FLVPlayback.addASCuePoint()` method. The component stores and tracks ActionScript cue points apart from the FLV file, and consequently, they are less accurate than embedded cue points. ActionScript cue points are accurate to a tenth of a second. You can increase the accuracy of ActionScript cue points by lowering the value of the `playheadUpdateInterval` property because the component generates the `cuePoint` event for ActionScript cue points when the playhead updates. For more information, see the FLVPlayback.playheadUpdateInterval property in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

In ActionScript and within the FLV file's metadata, a cue point is represented as an object with the following properties: `name`, `time`, `type`, and `parameters`. The `name` property is a string that contains the assigned name of the cue point. The `time` property is a number representing the time in hours, minutes, seconds, and milliseconds (HH:MM:SS.mmm) when the cue point occurs. The `type` property is a string whose value is `"navigation"`, `"event"`, or `"actionscript"`, depending on the type of cue point that you created. The `parameters` property is an array of specified name and value pairs.

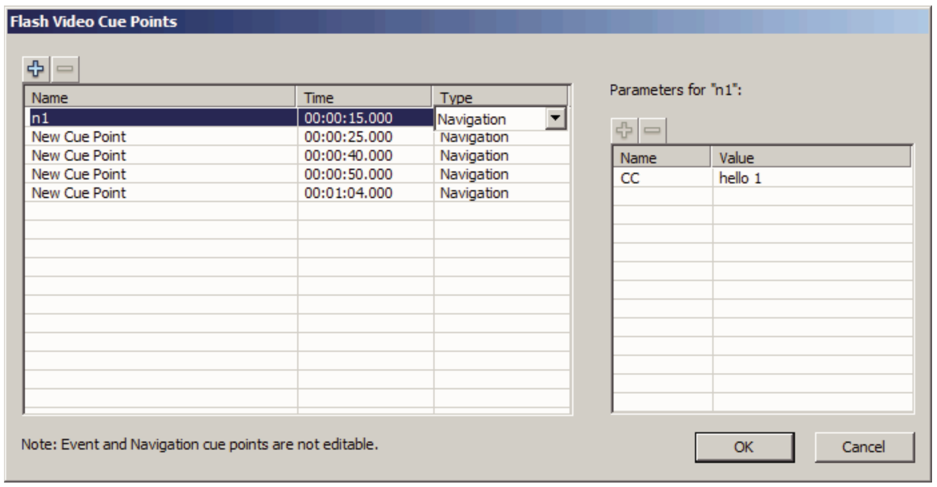
When a `cuePoint` event occurs, the cue point object is available in the event object through the `info` property.

Use the Flash Video Cue Points dialog box

In Flash Professional CS5, click the value cell of the `cuePoints` parameter in the Component inspector for the FLVPlayback component. The Cue Points dialog box appears at the bottom of the Property inspector.

In Flash Professional CS4, open the Flash Video Cue Points dialog box by double-clicking the value cell of the `cuePoints` parameter in the Component inspector.

The dialog box looks like the following figure:



Name	Time	Type
n1	00:00:15.000	Navigation
New Cue Point	00:00:25.000	Navigation
New Cue Point	00:00:40.000	Navigation
New Cue Point	00:00:50.000	Navigation
New Cue Point	00:01:04.000	Navigation

Parameters for "n1":

Name	Value
CC	hello 1

Note: Event and Navigation cue points are not editable.

Cue Points dialog box

The dialog box displays embedded and ActionScript cue points. You can use this dialog box to add and delete ActionScript cue points as well as cue point parameters. You can also enable or disable embedded cue points. However, you cannot add, change, or delete embedded cue points.

Add an ActionScript cue point:

1 Click the Value cell of the `cuePoints` parameter in the Component parameters to open the Cue Points dialog at the bottom of the Property inspector. (In Flash Professional CS4, double-click the value cell of the `cuePoints` parameter in the Component inspector to open the Flash Cue Points dialog box.)

2 Click the plus (+) sign in the upper-left corner, above the list of cue points, to add a default ActionScript cue point entry.

3 Click the New Cue Point text in the Name column, and edit the text to name the cue point.

4 Click the Time value of 00:00:00:000 to edit it, and assign a time for the cue point to occur. You can specify the time in hours, minutes, seconds, and milliseconds (HH:MM:SS.mmm).

   If multiple cue points exist, the dialog box moves the new cue point to its chronological position in the list.

5 To add a parameter for the selected cue point, click the plus (+) sign above the Parameters section, and enter values in the Name and Value columns. Repeat this step for each parameter.

6 To add more ActionScript cue points, repeat steps 2 through 5 for each one.

7 Press Enter to save your changes. (In Flash Professional CS4, Click OK to save your changes.)

Delete an ActionScript cue point:

1 Click the value cell of the `cuePoints` parameter. (In Flash Professional CS4, double-click the Value cell of the `cuePoints` parameter in the Component inspector to open the Flash Cue Points dialog box.)

2 Select the cue point that you want to delete.

3 Click the minus (-) sign in the upper-left corner, above the list of cue points, to delete it.

4 Repeat steps 2 and 3 for each cue point that you want to delete.

5 Press Enter to save your changes. (In Flash Professional CS4, Click OK to save your changes.)

To enable or disable an embedded FLV file cue point:

1 Click the value cell of the `cuePoints` parameter. (In Flash Professional CS4, double-click the value cell of the `cuePoints` parameter in the Component inspector to open the Flash Cue Points dialog box.)

2 Select the cue point you want to enable or disable.

3 Click the value in the Type column to trigger the pop-up menu, or click the down arrow.

4 Click the name of the type of cue point (for example, Event or Navigation) to enable it. Click Disabled to disable it.

5 Press Enter to save your changes. (In Flash Professional CS4, Click OK to save your changes.)

Use cue points with ActionScript

You can use ActionScript to add ActionScript cue points, listen for `cuePoint` events, find cue points of any type or a specific type, seek a navigation cue point, enable or disable a cue point, check whether a cue point is enabled, and remove a cue point.

The examples in this section use an FLV file called cuepoints.flv, which contains the following three cue points:

Name	Time	Type
point1	00:00:00.418	Navigation
point2	00:00:07.748	Navigation
point3	00:00:16.020	Navigation

Add ActionScript cue points

You can add ActionScript cue points to an FLV file using the `addASCuePoint()` method. The following example adds two ActionScript cue points to the FLV file when it is ready to play. It adds the first cue point using a cue point object, which specifies the time, name, and type of the cue point in its properties. The second call specifies the time and name using the method's `time` and `name` parameters.

```
// Requires an FLVPlayback instance called my_FLVPlybk on Stage
import fl.video.*;
import fl.video.MetadataEvent;
my_FLVPlybk.source = "http://www.helpexamples.com/flash/video/cuepoints.flv"
var cuePt:Object = new Object(); //create cue point object
cuePt.time = 2.02;
cuePt.name = "ASpt1";
cuePt.type = "actionscript";
my_FLVPlybk.addASCuePoint(cuePt);//add AS cue point
// add 2nd AS cue point using time and name parameters
my_FLVPlybk.addASCuePoint(5, "ASpt2");
```

For more information, see the FLVPlayback.addASCuePoint() method in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Listen for cuePoint events

The `cuePoint` event allows you to receive control in your ActionScript code when a `cuePoint` event occurs. When cue points occur in the following example, the `cuePoint` listener calls an event handler function that displays the value of the `playheadTime` property and the name and type of the cue point. Use this example in combination with the example in the preceding section, Add ActionScript cue points, to see the results.

```
my_FLVPlybk.addEventListener(MetadataEvent.CUE_POINT, cp_listener);
function cp_listener(eventObject:MetadataEvent):void {
        trace("Elapsed time in seconds: " + my_FLVPlybk.playheadTime);
        trace("Cue point name is: " + eventObject.info.name);
        trace("Cue point type is: " + eventObject.info.type);
}
```

For more information on the `cuePoint` event, see the FLVPlayback.cuePoint event in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Find cue points

Using ActionScript, you can find a cue point of any type, find the nearest cue point to a time, or find the next cue point with a specific name.

The `ready_listener()` event handler in the following example calls the `findCuePoint()` method to find the cue point `ASpt1` and then calls the `findNearestCuePoint()` method to find the navigation cue point that is nearest to the time of cue point `ASpt1`:

```
import fl.video.FLVPlayback;
import fl.video.CuePointType;
import fl.video.VideoEvent;
my_FLVPlybk.source = "http://www.helpexamples.com/flash/video/cuepoints.flv"
var rtn_obj:Object; //create cue point object
my_FLVPlybk.addASCuePoint(2.02, "ASpt1");//add AS cue point
function ready_listener(eventObject:VideoEvent):void {
    rtn_obj = my_FLVPlybk.findCuePoint("ASpt1", CuePointType.ACTIONSCRIPT);
    traceit(rtn_obj);
    rtn_obj = my_FLVPlybk.findNearestCuePoint(rtn_obj.time, CuePointType.NAVIGATION);
    traceit(rtn_obj);
}
my_FLVPlybk.addEventListener(VideoEvent.READY, ready_listener);
function traceit(cuePoint:Object):void {
    trace("Cue point name is: " + cuePoint.name);
    trace("Cue point time is: " + cuePoint.time);
    trace("Cue point type is: " + cuePoint.type);
}
```

In the following example, the `ready_listener()` event handler finds cue point `ASpt` and calls the `findNextCuePointWithName()` method to find the next cue point with the same name:

```
import fl.video.*;
my_FLVPlybk.source = "http://www.helpexamples.com/flash/video/cuepoints.flv"
var rtn_obj:Object; //create cue point object
my_FLVPlybk.addASCuePoint(2.02, "ASpt");//add AS cue point
my_FLVPlybk.addASCuePoint(3.4, "ASpt");//add 2nd Aspt
my_FLVPlybk.addEventListener(VideoEvent.READY, ready_listener);
function ready_listener(eventObject:VideoEvent):void {
    rtn_obj = my_FLVPlybk.findCuePoint("ASpt", CuePointType.ACTIONSCRIPT);
    traceit(rtn_obj);
    rtn_obj = my_FLVPlybk.findNextCuePointWithName(rtn_obj);
    traceit(rtn_obj);
}
function traceit(cuePoint:Object):void {
    trace("Cue point name is: " + cuePoint.name);
    trace("Cue point time is: " + cuePoint.time);
    trace("Cue point type is: " + cuePoint.type);
}
```

For more information about finding cue points, see the FLVPlayback.findCuePoint(), FLVPlayback.findNearestCuePoint(), and FLVPlayback.findNextCuePointWithName() methods in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Seek navigation cue points

You can seek a navigation cue point, seek the next navigation cue point from a specified time, and seek the previous navigation cue point from a specified time. The following example plays the FLV file cuepoints.flv and seeks the cue point at 7.748 when the `ready` event occurs. When the `cuePoint` event occurs, the example calls the `seekToPrevNavCuePoint()` method to seek the first cue point. When that `cuePoint` event occurs, the example calls the `seekToNextNavCuePoint()` method to seek the last cue point by adding 10 seconds to `eventObject.info.time`, which is the time of the current cue point.

```
import fl.video.*;

my_FLVPlybk.addEventListener(VideoEvent.READY, ready_listener);
function ready_listener(eventObject:Object):void {
    my_FLVPlybk.seekToNavCuePoint("point2");
}
my_FLVPlybk.addEventListener(MetadataEvent.CUE_POINT, cp_listener);
function cp_listener(eventObject:MetadataEvent):void {
    trace(eventObject.info.time);
    if(eventObject.info.time == 7.748)
        my_FLVPlybk.seekToPrevNavCuePoint(eventObject.info.time - .005);
    else
        my_FLVPlybk.seekToNextNavCuePoint(eventObject.info.time + 10);
}
my_FLVPlybk.source = "http://helpexamples.com/flash/video/cuepoints.flv";
```

For more information, see the FLVPlayback.seekToNavCuePoint(), FLVPlayback.seekToNextNavCuePoint(), and FLVPlayback.seekToPrevNavCuePoint() methods in the *ActionScript 3.0 Reference*.

Enable and disable embedded FLV file cue points

You can enable and disable embedded FLV file cue points using the `setFLVCuePointEnabled()` method. Disabled cue points do not trigger `cuePoint` events and do not work with the `seekToCuePoint()`, `seekToNextNavCuePoint()`, or `seekToPrevNavCuePoint()` methods. You can find disabled cue points, however, with the `findCuePoint()`, `findNearestCuePoint()`, and `findNextCuePointWithName()` methods.

You can test whether an embedded FLV file cue point is enabled using the `isFLVCuePointEnabled()` method. The following example disables the embedded cue points `point2` and `point3` when the video is ready to play. When the first `cuePoint` event occurs, however, the event handler tests to see whether cue point `point3` is disabled and, if so, enables it.

```
import fl.video.*;
my_FLVPlybk.source = "http://www.helpexamples.com/flash/video/cuepoints.flv";
my_FLVPlybk.addEventListener(VideoEvent.READY, ready_listener);
function ready_listener(eventObject:VideoEvent):void {
    my_FLVPlybk.setFLVCuePointEnabled(false, "point2");
    my_FLVPlybk.setFLVCuePointEnabled(false, "point3");
}
my_FLVPlybk.addEventListener(MetadataEvent.CUE_POINT, cp_listener);
function cp_listener(eventObject:MetadataEvent):void {
        trace("Cue point time is: " + eventObject.info.time);
        trace("Cue point name is: " + eventObject.info.name);
        trace("Cue point type is: " + eventObject.info.type);
        if (my_FLVPlybk.isFLVCuePointEnabled("point2") == false) {
            my_FLVPlybk.setFLVCuePointEnabled(true, "point2");
        }
}
```

For more information, see the FLVPlayback.isFLVCuePointEnabled() and FLVPlayback.setFLVCuePointEnabled() methods in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Remove an ActionScript cue point

You can remove an ActionScript cue point using the `removeASCuePoint()` method. The following example removes the cue point `ASpt2` when cue point `ASpt1` occurs:

```
import fl.video.*;
my_FLVPlybk.source = "http://www.helpexamples.com/flash/video/cuepoints.flv"
my_FLVPlybk.addASCuePoint(2.02, "ASpt1");//add AS cue point
my_FLVPlybk.addASCuePoint(3.4, "ASpt2");//add 2nd Aspt
my_FLVPlybk.addEventListener(MetadataEvent.CUE_POINT, cp_listener);
function cp_listener(eventObject:MetadataEvent):void {
        trace("Cue point name is: " + eventObject.info.name);
        if (eventObject.info.name == "ASpt1") {
            my_FLVPlybk.removeASCuePoint("ASpt2");
            trace("Removed cue point ASpt2");
        }
}
```

For more information, see FLVPlayback.removeASCuePoint() in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Play multiple video files

You can play video files sequentially in an FLVPlayback instance simply by loading a new URL in the `source` property when the previous video file finishes playing. For example, the following ActionScript code listens for the `complete` event, which occurs when a video file finishes playing. When this event occurs, the code sets the name and location of a new video file in the `source` property and calls the `play()` method to play the new video.

```
import fl.video.*;
my_FLVPlybk.source = "http://www.helpexamples.com/flash/video/clouds.flv";
my_FLVPlybk.addEventListener(VideoEvent.COMPLETE, complete_listener);
// listen for complete event; play new FLV
function complete_listener(eventObject:VideoEvent):void {
    if (my_FLVPlybk.source == "http://www.helpexamples.com/flash/video/clouds.flv") {
        my_FLVPlybk.play("http://www.helpexamples.com/flash/video/water.flv");
    }
};
```

Use multiple video players

You can also open multiple video players within a single instance of the FLVPlayback component to play multiple videos and switch between them as they play.

You create the initial video player when you drag the FLVPlayback component to the Stage. The component automatically assigns the initial video player the number 0 and makes it the default player. To create an additional video player, simply set the `activeVideoPlayerIndex` property to a new number. Setting the `activeVideoPlayerIndex` property also makes the specified video player the *active* video player, which is the one that will be affected by the properties and methods of the FLVPlayback class. Setting the `activeVideoPlayerIndex` property does not make the video player visible, however. To make the video player visible, set the `visibleVideoPlayerIndex` property to the video player's number. For more information on how these properties interact with the methods and properties of the FLVPlayback class, see the FLVPlayback.activeVideoPlayerIndex and FLVPlayback.visibleVideoPlayerIndex properties in the *ActionScript 3.0 Reference for the Adobe Flash Platform* .

The following ActionScript code loads the `source` property to play a video file in the default video player and adds a cue point for it. When the `ready` event occurs, the event handler opens a second video player by setting the `activeVideoPlayerIndex` property to the number 1. It specifies a FLV file and a cue point for the second video player and then makes the default player (0) the active video player again.

```
/**
    Requires:
- FLVPlayback component on the Stage with an instance name of my_FLVPlybk
*/
// add a cue point to the default player
import fl.video.*;
my_FLVPlybk.source = "http://www.helpexamples.com/flash/video/clouds.flv";
my_FLVPlybk.addASCuePoint(3, "1st_switch");
my_FLVPlybk.addEventListener(VideoEvent.READY, ready_listener);
function ready_listener(eventObject:VideoEvent):void {
    // add a second video player and create a cue point for it
    my_FLVPlybk.activeVideoPlayerIndex = 1;
    my_FLVPlybk.source = "http://www.helpexamples.com/flash/video/water.flv";
    my_FLVPlybk.addASCuePoint(3, "2nd_switch");
    my_FLVPlybk.activeVideoPlayerIndex = 0;
};
```

To switch to another FLV file while one is playing, you must make the switch in your ActionScript code. Cue points allow you to intervene at specific points in the FLV file using a `cuePoint` event. The following code creates a listener for the `cuePoint` event and calls a handler function that pauses the active video player (0), switches to the second player (1), and plays its FLV file:

```
import fl.video.*;
// add listener for a cuePoint event
my_FLVPlybk.addEventListener(MetadataEvent.CUE_POINT, cp_listener);
// add the handler function for the cuePoint event
function cp_listener(eventObject:MetadataEvent):void {
        // display the no. of the video player causing the event
        trace("Hit cuePoint event for player: " + eventObject.vp);
        // test for the video player and switch FLV files accordingly
        if (eventObject.vp == 0) {
            my_FLVPlybk.pause(); //pause the first FLV file
            my_FLVPlybk.activeVideoPlayerIndex = 1; // make the 2nd player active
            my_FLVPlybk.visibleVideoPlayerIndex = 1; // make the 2nd player visible
            my_FLVPlybk.play(); // begin playing the new player/FLV
        } else if (eventObject.vp == 1) {
            my_FLVPlybk.pause(); // pause the 2nd FLV
            my_FLVPlybk.activeVideoPlayerIndex = 0; // make the 1st player active
            my_FLVPlybk.visibleVideoPlayerIndex = 0; // make the 1st player visible
            my_FLVPlybk.play(); // begin playing the 1st player
        }
}
my_FLVPlybk.addEventListener(VideoEvent.COMPLETE, complete_listener);
function complete_listener(eventObject:VideoEvent):void {
        trace("Hit complete event for player: " + eventObject.vp);
        if (eventObject.vp == 0) {
            my_FLVPlybk.activeVideoPlayerIndex = 1;
            my_FLVPlybk.visibleVideoPlayerIndex = 1;
            my_FLVPlybk.play();
        } else {
            my_FLVPlybk.closeVideoPlayer(1);
        }
};
```

When you create a new video player, the FLVPlayback instance sets its properties to the value of the default video player, except for the `source`, `totalTime`, and `isLive` properties, which the FLVPlayback instance always sets to the default values: empty string, 0, and `false`, respectively. It sets the `autoPlay` property, which defaults to `true` for the default video player, to `false`. The `cuePoints` property has no effect, and it has no effect on a subsequent load into the default video player.

The methods and properties that control volume, positioning, dimensions, visibility, and the user interface controls are always global and their behavior is not affected by setting the `activeVideoPlayerIndex` property. For more information on these methods and properties and the effect of setting the `activeVideoPlayerIndex` property, see the FLVPlayback.activeVideoPlayerIndex property in the *ActionScript 3.0 Reference for the Adobe Flash Platform*. The remaining properties and methods target the video player identified by the value of the `activeVideoPlayerIndex` property.

Properties and methods that control dimensions *dointeract* with the `visibleVideoPlayerIndex` property, however. For more information, see the FLVPlayback.visibleVideoPlayerIndex property in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Stream FLV files from Flash Media Server

The requirements for streaming FLV files from Flash Media Server are different depending on whether native bandwidth detection is available from your Flash Video Streaming Service provider. Native bandwidth detection means that the bandwidth detection is built-in to the streaming server and it provides better performance. Check with your provider to determine whether native bandwidth detection is available.

To access your FLV files on Flash Media Server, use a URL such as rtmp://*my_servername/my_application/stream.*flv.

When playing a live stream with Flash Media Server, you need to set the FLVPlayback `isLive` property to `true`. For more information, see the FLVPlayback.isLive property in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

For more information on administering Flash Media Server, including how to set up a live stream, see the Flash Media Server documentation at www.adobe.com/support/documentation/en/flashmediaserver/.

For native bandwidth detection or no bandwidth detection

The NCManagerNative class is a subclass of NCManager that supports native bandwidth detection, which some Flash Video Streaming Service providers may support. When you use NCManagerNative, no special files are needed on the Flash Media Server. NCManagerNative also allows connection to any version of Flash Media Server, without a main.asc file, if bandwidth detection is not required.

To use NCManagerNative instead of the default NCManager class, add the following lines of code in the first frame of your FLA file:

```
import fl.video*;
VideoPlayer.iNCManagerClass = fl.video.NCManagerNative;
```

For non-native bandwidth detection

If native bandwidth detection is not available from your Flash Video Streaming Service provider but you need bandwidth detection, you must add the main.asc file to your Flash Media Server FLV application. You can find the main.asc file online at www.adobe.com/go/learn_fl_samples. It's contained in the Samples.zip file -- within the Samples\ComponentsAS2\FLVPlayback directory.

To set up your Flash Media Server for streaming FLV files:

1 Create a folder in your Flash Media Server application folder, and give it a name such as **my_application**.

2 Copy the main.asc file into the my_application folder.

3 Create a folder named **streams** in the my_application folder.

4 Create a folder named **_definst_** inside the streams folder.

5 Place your FLV files in the **_definst_** folder.

Customize the FLVPlayback component

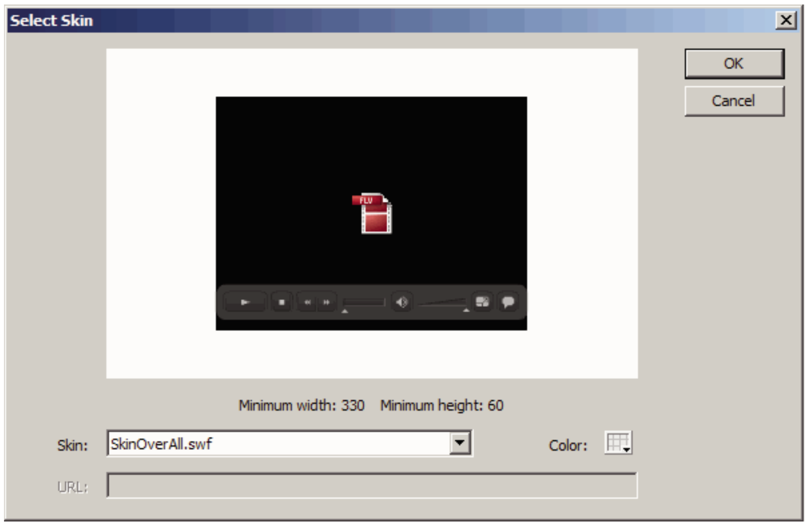
This section explains how to customize the FLVPlayback component. Most of the methods used to customize other components, however, do not work with the FLVPlayback component. To customize the FLVPlayback component, use only the techniques described in this section.

You have the following choices for customizing the FLVPlayback component: select a predesigned skin, skin FLV Playback Custom UI components individually, or create a new skin. You can also use FLVPlayback properties to modify the behavior of a skin.

***Note:** You must upload your skin SWF file to the web server along with your application SWF file for the skin to work with your FLVPlayback component.*

Select a predesigned skin

You can select a skin for the FLVPlayback component by clicking the `value` cell for the `skin` parameter in the Component inspector. Then click the magnifying glass icon to open the following Select Skin dialog box, in which you can select a skin or provide a URL that specifies the location of the skin SWF file.



Minimum width: 330 Minimum height: 60

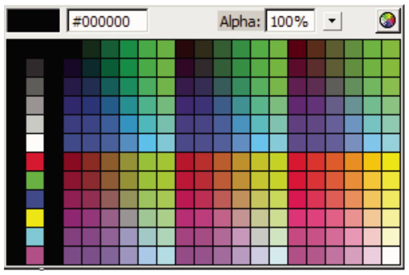
Skin: SkinOverAll.swf

URL:

FLVPlayback Select Skin dialog box

Skins that are listed in the Skin pop-up menu are located in the Flash application folder /Flash Configuration/FLVPlayback Skins/ActionScript 3.0. You can make new skins available to this dialog box by creating them and placing the SWF file in the folder. The name appears in the pop-up menu with a .swf extension. For more information about creating a skin set, see "Create a new skin" on page 155.

For skins that you assign by setting the `skin` property, either by setting the skin parameter during authoring or with ActionScript at run time, you can assign color and alpha (transparency) values independent of choosing the skin. To assign color and alpha values during authoring, open the Color picker in the Skin Select dialog box, as shown here.



Color picker in the Skin Select dialog box

To choose the color, click a swatch in the panel or enter its numeric value in the text box. To choose the alpha value, use the slider or type a percentage in the Alpha text box.

To assign color and alpha values during run time, set the `skinBackgroundColor` and `skinBackgroundAlpha` properties. Set the `skinBackgroundColor` property to a 0xRRGGBB (red, green, blue) value. Set the skinBackgroundAlpha property to a number between 0.0 and 1.0. The following example sets `skinBackgroundColor` to 0xFF0000 (red) and skinBackgroundAlpha to .5.

```
my_FLVPlybk.skinBackgroundColor = 0xFF0000;
my_FLVPlybk.skinBackgroundAlpha = .5;
```

The default values are the last values chosen by the user.

If you want to skin the FLVPlayback component using the FLV Playback Custom UI components, select None from the pop-up menu.

Skin FLV Playback Custom UI components individually

The FLV Playback Custom UI components allow you to customize the appearance of the FLVPlayback controls within your FLA file and allow you to see the results when you preview your web page. These components are not designed to be scaled, however. You should edit a movie clip and its contents to be a specific size. For this reason, it is generally best to have the FLVPlayback component on the Stage at the desired size, with the `scaleMode` set to `exactFit`.

To begin, simply drag the FLV Playback Custom UI components that you want from the Components panel, place them where you want them on the Stage and give them instance names.

These components can work without any ActionScript. If you put them on the same timeline and frame as the FLVPlayback component and there is no skin set in the component, the FLVPlayback component will connect automatically to them. If you have multiple FLVPlayback components on Stage, or if the custom control and the FLVPlayback instance are not on the same Timeline, then Action is needed.

After your components are on the Stage, you edit them as you would any other symbol. After you open the components, you can see that each one is set up a little differently from the others.

Button components

The button components have a similar structure. The buttons include the BackButton, ForwardButton, MuteButton, PauseButton, PlayButton, PlayPauseButton, and StopButton. Most have a single movie clip on Frame 1 with the instance name placeholder_mc. This is usually an instance of the normal state for the button, but not necessarily so. On Frame 2, there are four movie clips on the Stage for each display state: normal, over, down, and disabled. (At run time, the component never actually goes to Frame 2; these movie clips are placed here to make editing more convenient and to force them to load into the SWF file without selecting the Export in First Frame check box in the Symbol Properties dialog box. You must still select the Export for ActionScript option, however.)

To skin the button, you simply edit each of these movie clips. You can change their size as well as their appearance.

Some ActionScript usually appears on Frame 1. You should not need to change this script. It simply stops the playhead on Frame 1 and specifies which movie clips to use for which states.

PlayPauseButton, MuteButton, FullScreenButton, and CaptionButton buttons

The PlayPauseButton, MuteButton, FullScreenButton, and CaptionButton buttons are set up differently than the other buttons; they have only one frame with two layers and no script. On that frame, there are two buttons, one on top of the other—in the case of PlayPauseButton, a Play and a Pause button; in the case of MuteButton, a Mute-on and a Mute-off button; in the case of FullScreenButton, a full-screen-on and a full-screen-off button; in the case of CaptionButton, a caption-on and a caption-off button. To skin these buttons, skin each of these two internal buttons as described in “Skin FLV Playback Custom UI components individually” on page 150; no additional action is required.

The CaptionButton is for the FLVPlaybackCaptioning component and must be attached to that component and not the FLVPlayback component.

BackButton and ForwardButton buttons

The BackButton and ForwardButton buttons are also set up differently than the other buttons. On Frame 2, they have extra movie clips that you can use as a frame around one or both of the buttons. These movie clips are not required and have no special capability; they are provided only as a convenience. To use them, simply drag them on the Stage from your Library panel and place them where you want them. If you don’t want them, either don’t use them or delete them from your Library panel.

Most of the buttons, as supplied, are based on a common set of movie clips so that you can change the appearance of all the buttons at once. You can use this capability, or you can replace those common clips and make every button look different.

BufferingBar component

The buffering bar component is simple: It consists of an animation that is made visible when the component enters the buffering state, and it does not require any special ActionScript to configure it. By default, it is a striped bar moved from left to right with a rectangular mask on it to give it a “barber pole” effect, but there is nothing special about this configuration.

Although the buffering bars in the skin SWF files use 9-slice scaling because they need to be scaled at run time, the BufferingBar FLV Custom UI Component does not and *cannot* use 9-slice scaling because it has nested movie clips. If you want to make the BufferingBar wider or taller, you might want to change its contents rather than scale it.

SeekBar and VolumeBar components

The SeekBar and VolumeBar components are similar, although they have different functions. Each has handles, uses the same handle tracking mechanisms, and has support for clips nested within to track progress and fullness.

There are many places where the ActionScript code in the FLVPlayback component assumes that the registration point (also *origin point* or *zero point*) of your SeekBar or VolumeBar component is at the upper-left corner of the content, so it is important to maintain this convention. Otherwise, you might have problems with handles and with progress and fullness movie clips.

Although the seek bars in the skin SWF files use 9-slice scaling because they need to be scaled at run time, the SeekBar FLV Custom UI component does not and *cannot* use 9-slice scaling because it has nested movie clips. If you want to make the SeekBar wider or taller, you might want to change its contents rather than scale it.

Handle

An instance of the handle movie clip is on Frame 2. As with the BackButton and ForwardButton components, the component never actually goes to Frame 2; these movie clips are placed here to make editing more convenient and as a way to force them to be loaded into the SWF file without selecting the Export in First Frame check box in the Symbol Properties dialog box. You still must select the Export for ActionScript option, however.

You might notice that the handle movie clip has a rectangle in the background with alpha set to 0. This rectangle increases the size of the handle's hit area, making it easier to grab without changing its appearance, similar to the hit state of a button. Because the handle is created dynamically at run time, it must be a movie clip and not a button. This rectangle with alpha set to 0 is not necessary for any other reason and, generally, you can replace the inside of the handle with any image you want. It works best, however, to keep the registration point centered horizontally in the middle of the handle movie clip.

The following ActionScript code is on Frame 1 of the SeekBar component to manage the handle:

```
stop();
handleLinkageID = "SeekBarHandle";
handleLeftMargin = 2;
handleRightMargin = 2;
handleY = 11;
```

The call to the `stop()` function is necessary due to the content of Frame 2.

The second line specifies which symbol to use as the handle, and you should not need to change this if you simply edit the handle movie clip instance on Frame 2. At run time, the FLVPlayback component creates an instance of the specified movie clip on the Stage as a sibling of the Bar component instance, which means that they have the same parent movie clip. So, if your bar is at the root level, your handle must also be at the root level.

The variable `handleLeftMargin` determines the handle's original location (0%), and the variable `handleRightMargin` determines where it is at the end (100%). The numbers give the offsets from the left and right ends of the bar control, with positive numbers marking the limits within the bar and negative numbers marking the limits outside the bar. These offsets specify where the handle can go, based on its registration point. If you put your registration point in the middle of the handle, the handle's far left and right sides will go past the margins. A seek bar movie clip must have its registration point as the upper-left corner of its content to work properly.

The variable `handleY` determines the `y` position of the handle, relative to the bar instance. This is based on the registration points of each movie clip. The registration point in the sample handle is at the tip of the triangle to place it relative to the visible part, disregarding the invisible hit state rectangle. Also, the bar movie clip must keep its registration point as the upper-left corner of its content to work properly.

So, for example, with these limits, if a bar control is set at (100, 100) and it is 100 pixels wide, the handle can range from 102 to 198 horizontally and stay at 111 vertically. If you change the `handleLeftMargin` and `handleRightMargin` to -2 and `handleY` to -11, the handle can range from 98 to 202 horizontally and stay at 89 vertically.

Progress and fullness movie clips

The SeekBar component has a *progress* movie clip and the VolumeBar has a *fullness* movie clip, but in practice, any SeekBar or VolumeBar can have either, neither, or both of these movie clips. They are structurally the same and behave similarly but track different values. A progress movie clip fills up as the FLV file downloads (which is useful for an HTTP download only, because it is always full if streaming from FMS) and a fullness movie clip fills up as the handle moves from left to right.

The FLVPlayback component finds these movie clip instances by looking for a specific instance name, so your progress movie clip instance must have your bar movie clip as its parent and have the instance name progress_mc. The fullness movie clip instance must have the instance name fullness_mc.

You can set the progress and fullness movie clips with or without the fill_mc movie clip instance nested inside. The VolumeBar fullness_mc movie clip shows the method *with* the fill_mc movie clip, and the SeekBar progress_mc movie clip shows the method *without* the fill_mc movie clip.

The method with the fill_mc movie clip nested inside is useful when you want a fill that cannot be scaled without distorting the appearance.

In the VolumeBar fullness_mc movie clip, the nested fill_mc movie clip instance is masked. You can either mask it when you create the movie clip, or a mask will be created dynamically at run time. If you mask it with a movie clip, name the instance **mask_mc** and set it up so that fill_mc appears as it would when percentage is 100%. If you do not mask fill_mc, the dynamically created mask will be rectangular and the same size as fill_mc at 100%.

The fill_mc movie clip is revealed with the mask in one of two ways, depending on whether fill_mc.slideReveal is `true` or `false`.

If fill_mc.slideReveal is `true`, then fill_mc is moved from left to right to expose it through the mask. At 0%, it is all the way to the left, so none of it shows through the mask. As the percentage increases, it moves to the right, until at 100%, it is back where it was created on the Stage.

If fill_mc.slideReveal is `false` or undefined (the default behavior), the mask will be resized from left to right to reveal more of fill_mc. When it is at 0%, the mask will be scaled to 05 horizontally, and as the percentage increases, the `scaleX` increases until, at 100%, it reveals all of fill_mc. This is not necessarily `scaleX` = 100 because mask_mc might have been scaled when it was created.

The method without fill_mc is simpler than the method with fill_mc, but it distorts the fill horizontally. If you do not want that distortion, you must use fill_mc. The SeekBar progress_mc illustrates this method.

The progress or fullness movie clip is scaled horizontally based on the percentage. At 0%, the instance's `scaleX` is set to 0, making it invisible. As the percentage grows, the `scaleX` is adjusted until, at 100%, the clip is the same size it was on the Stage when it was created. Again, this is not necessarily `scaleX` = 100 because the clip instance might have been scaled when it was created.

Connect your FLV Playback Custom UI components

If you put your custom UI components on the same Timeline and frame as the FLVPlayback component and you have not set the `skin` property, FLVPlayback will automatically connect to them without the need for any ActionScript.

If you have multiple FLVPlayback components on Stage or if the custom control and the FLVPlayback are not on the same Timeline, you must write ActionScript code to connect your Custom UI components to your instance of the FLVPlayback component. First, you must assign a name to the FLVPlayback instance and then use ActionScript to assign your FLV Playback Custom UI component instances to the corresponding FLVPlayback properties. In the following example, the FLVPlayback instance is my_FLVPlybk, the FLVPlayback property names follow the periods (.), and the FLV Playback Custom UI control instances are to the right of the equal (=) signs:

```
//FLVPlayback instance = my_FLVPlybk
my_FLVPlybk.playButton = playbtn; // set playButton prop. to playbtn, etc.
my_FLVPlybk.pauseButton = pausebtn;
my_FLVPlybk.playPauseButton = playpausebtn;
my_FLVPlybk.stopButton = stopbtn;
my_FLVPlybk.muteButton = mutebtn;
my_FLVPlybk.backButton = backbtn;
my_FLVPlybk.forwardButton = forbtn;
my_FLVPlybk.volumeBar = volbar;
my_FLVPlybk.seekBar = seekbar;
my_FLVPlybk.bufferingBar = bufbar;
```

The following steps create custom StopButton, PlayPauseButton, MuteButton, and SeekBar controls:

1 Drag the FLVPlayback component to the Stage, and give it an instance name of **my_FLVPlybk**.

2 Set the `source` parameter through the Component inspector to **http://www.helpexamples.com/flash/video/cuepoints.flv**.

3 Set the Skin parameter to None.

4 Drag a StopButton, a PlayPauseButton, and a MuteButton to the Stage, and place them over the FLVPlayback instance, stacking them vertically on the left. Give each button an instance name in the Property inspector (such as **my_stopbttn**, **my_plypausbttn**, and **my_mutebttn**).

5 In the Library panel, open the FLVPlayback Skins folder, and then open the SquareButton folder below it.

6 Select the SquareBgDown movie clip, and double-click it to open it on the Stage.

7 Right-click (Windows) or Control-click (Macintosh), select Select All from the menu, and delete the symbol.

8 Select the oval tool, draw an oval in the same location, and set the fill to blue (**#0033FF**).

9 In the Property inspector, set the width (W:) to **40** and the height (H:) to **20**. Set the x-coordinate (X:) to **0.0** and the y-coordinate (Y:) to **0.0**.

10 Repeat steps 6 to 8 for SquareBgNormal, but change the fill to yellow (**#FFFF00**).

11 Repeat steps 6 to 8 for SquareBgOver, but change the fill to green (**#006600**).

12 Edit the movie clips for the various symbol icons within the buttons (PauseIcon, PlayIcon, MuteOnIcon, MuteOffIcon, and StopIcon). You can find these movie clips in the Library panel under FLV Playback Skins/*Label* Button/Assets, where *Label* is the name of the button, such as Play, Pause, and so on. Follow these steps for each one:

   a Select the Select All option.

   b Change the color to red (**#FF0000**).

   c Scale to 300%.

   d Change the X: location of the content to **7.0** to alter the horizontal placement of the icon in every button state.

   ***Note:** By changing the location this way, you avoid opening every button state and moving the icon movie clip instance.*

13 Click the blue Back arrow above the Timeline to return to Scene 1, Frame 1.

14 Drag a SeekBar component to the Stage, and place it in the lower-right corner of the FLVPlayback instance.

15 In the Library panel, double-click the SeekBar to open it on the Stage.

16 Scale it to 400%.

17 Select the outline, and set the color to red (**#FF0000**).

18 Double-click SeekBarProgress in the FLVPlayback Skins/Seek Bar folder, and set the color to yellow (**#FFFF00**).

19 Double-click SeekBarHandle in the FLVPlayback Skins/Seek Bar folder and set the color to red (**#FF0000**).

20 Click the blue Back arrow above the Timeline to return to Scene 1, Frame 1.

21 Select the SeekBar instance on the Stage, and give it an instance name of **my_seekbar**.

22 In the Actions panel on Frame 1 of the Timeline, add an `import` statement for the video classes, and assign the button and seek bar names to the corresponding FLVPlayback properties, as shown in the following example:

```
import fl.video.*;
my_FLVPlybk.stopButton = my_stopbttn;
my_FLVPlybk.playPauseButton = my_plypausbttn;
my_FLVPlybk.muteButton = my_mutebttn;
my_FLVPlybk.seekBar = my_seekbar;
```

23 Press Control+Enter to test the movie.

Create a new skin

The best way to create a skin SWF file is to copy one of the skin files that come with Flash, and use it as a starting point. You can find the FLA files for these skins in the Flash application folder in Configuration/FLVPlayback Skins/FLA/ActionScript 3.0/. To make your finished skin SWF file available as an option in the Select Skin dialog box, put it in the Configuration/FLVPlayback Skins/ActionScript 3.0 folder, either in the Flash application folder or in a user's local Configuration/FLVPlayback Skins/ActionScript 3.0 folder.

Because you can set the color of a skin independently of choosing the skin, you do not need to edit the FLA file to modify the color. If you create a skin that has a specific color and you do not want it to be editable in the Select Skin dialog box, set `this.border_mc.colorMe = false;` in the skin FLA file ActionScript code. For information on setting a skin's color, see "Select a predesigned skin" on page 149.

When looking at the installed Flash skin FLA files, it might seem that certain things on the Stage are unnecessary, but many of these things are put into guide layers. With live preview using scale 9, you can quickly see what will actually appear in the SWF file.

The following sections cover more complex customizations and changes to the SeekBar, BufferingBar, and VolumeBar movie clips.

Use the skin layout

When you open a Flash skin FLA file, you will find the skin's movie clips laid out on the main Timeline. These clips and the ActionScript code that you find on the same frame define how the controls will be laid out at run time.

Although the Layout layer looks a lot like how the skin will look like at run time, the contents of this layer are not visible at run time. It is used only to calculate where to place the controls. The other controls on the Stage are used at run time.

Within the Layout layer is a placeholder for the FLVPlayback component named video_mc. All the other controls are laid out relative to video_mc. If you start with one of the Flash FLA files and change the size of the controls, you can probably fix the layout by moving these placeholder clips.

Each of the placeholder clips has a specific instance name. The names of the placeholder clips are playpause_mc, play_mc, pause_mc, stop_mc, captionToggle_mc, fullScreenToggle_mc, back_mc, bufferingBar_mc, bufferingBarFill_mc, seekBar_mc, seekBarHandle_mc, seekBarProgress_mc, volumeMute_mc, volumeBar_mc, and volumeBarHandle_mc. The piece that is recolored when you choose a skin color is called border_mc.

Which clip is used for a control is not important. Generally, for buttons the normal state clip is used. For other controls the clip for that control is used, but this is only for convenience. All that is important are the *x* (horizontal) and *y* (vertical) location and the height and the width of the placeholder.

You can also have as many additional clips as you want beside the standard controls. The only requirement for these clips is that their library symbols have Export for ActionScript checked in the Linkage dialog box. The custom clips in the Layout layer can have any instance name, other than the reserved instance names listed above. An instance name is only needed to set ActionScript on the clips to determine the layout.

The border_mc clip is special. If you set the `FlvPlayback.skinAutoHide` property to `true`, the skin shows when the mouse is over the border_mc clip. This is important for skins that appear outside the bounds of the video player. For information on the `skinAutoHide` property, see "Modify skin behavior" on page 159.

In the Flash FLA files, border_mc is used for the chrome and for the border around the Forward and Back buttons.

The border_mc clip is also the part of the skin that has its alpha and color changed by the `skinBackgroundAlpha` and `skinBackgroundColor` properties. To allow customizable color and alpha, the ActionScript in the skin FLA file must include the following:

```
border_mc.colorMe = true;
```

ActionScript and skin layout

The following ActionScript code generally applies to all controls. Some controls have specific ActionScript that defines additional behavior, and that is explained in the section for that control.

The initial ActionScript is a large section that specifies the class names for each state of each component. You can see all of these class names in the SkinOverAll.fla file. The code looks like this for the Pause and Play buttons, for example:

```
this.pauseButtonDisabledState = "fl.video.skin.PauseButtonDisabled";
this.pauseButtonDownState = "fl.video.skin.PauseButtonDown";
this.pauseButtonNormalState = "fl.video.skin.PauseButtonNormal";
this.pauseButtonOverState = "fl.video.skin.PauseButtonOver";
this.playButtonDisabledState = "fl.video.skin.PlayButtonDisabled";
this.playButtonDownState = "fl.video.skin.PlayButtonDown";
this.playButtonNormalState = "fl.video.skin.PlayButtonNormal";
this.playButtonOverState = "fl.video.skin.PlayButtonOver";
```

The class names do not have actual external class files; they are just specified in the Linkage dialog box for all the movie clips in the library.

In the ActionScript 2.0 component, there were movie clips on Stage that were actually used at run time. In the ActionScript 3.0 component, those movie clips are still in the FLA file, but just to make editing convenient. Now, they are all in guide layers and are not exported. All of the skin assets in the library are set to export on the first frame and they are created dynamically with code like this, for example:

```
new fl.video.skin.PauseButtonDisabled();
```

Following that section is ActionScript code that defines the minimum width and height for the skin. The Select Skin dialog box shows these values and they are used at run time to prevent the skin from scaling below its minimum size. If you do not want to specify a minimum size, leave it as undefined or less than or equal to zero.

```
// minimum width and height of video recommended to use this skin,
// leave as undefined or <= 0 if there is no minimum
this.minWidth = 270;
this.minHeight = 60;
```

Each placeholder can have the following properties applied to it:

Property	Description
`anchorLeft`	Boolean. Positions the control relative to the left side of the FLVPlayback instance. Defaults to `true` unless `anchorRight` is explicitly set to `true`, and then it defaults to `false`.
`anchorRight`	Boolean. Positions the control relative to the right side of the FLVPlayback instance. Defaults to `false`.
`anchorBottom`	Boolean. Positions the control relative to the bottom of the FLVPlayback instance. Defaults to `true`, unless `anchorTop` is explicitly set to `true`, and then it defaults to `false`.
`anchorTop`	Boolean. Positions the control relative to the top of the FLVPlayback instance. Defaults to `false`.

If both the `anchorLeft` and `anchorRight` properties are `true`, the control is scaled horizontally at run time. If both the `anchorTop` and `anchorBottom` properties are `true`, the control is scaled vertically at run time.

To see the effects of these properties, see how they are used in the Flash skins. The BufferingBar and SeekBar controls are the only ones that scale, and they are laid on top of one another and have both the `anchorLeft` and `anchorRight` properties set to `true`. All controls to the left of the BufferingBar and SeekBar have `anchorLeft` set to `true`, and all controls to their right have `anchorRight` set to `true`. All controls have `anchorBottom` set to `true`.

You can try editing the movie clips on the Layout layer to make a skin where the controls sit at the top rather than at the bottom. You simply need to move the controls to the top, relative to `video_mc`, and set `anchorTop` equal to `true` for all controls.

Buffering bar

The buffering bar has two movie clips: bufferingBar_mc and bufferingBarFill_mc. Each clip's position on the Stage relative to the other clip is important because this relative positioning is maintained. The buffering bar uses two separate clips because the component scales bufferingBar_mc but not bufferingBarFill_mc.

The bufferingBar_mc clip has 9-slice scaling applied to it, so the borders won't distort when it scales. The bufferingBarFill_mc clip is extremely wide, so that it will always be wide enough without needing to be scaled. It is automatically masked at run time to show only the portion above the stretched bufferingBar_mc. By default, the exact dimensions of the mask will maintain an equal margin on the left and right within the bufferingBar_mc, based on the difference between the *x* (horizontal) positions of bufferingBar_mc and bufferingBarFill_mc. You can customize the positioning with ActionScript code.

If your buffering bar does not need to scale or does not use 9-slice scaling, you could set it up like the FLV Playback Custom UI BufferingBar component. For more information, see "BufferingBar component" on page 151.

The buffering bar has the following additional property:

Property	Description
`fill_mc:MovieClip`	Specifies the instance name of the buffering bar fill. Defaults to bufferingBarFill_mc.

Seek bar and volume bar

The seek bar also has two movie clips: seekBar_mc and seekBarProgess_mc. Each clip's position on the Layout layer relative to the other clip is important because this relative positioning is maintained. Although both clips scale, the seekBarProgress_mc cannot be nested within seekBar_mc because seekBar_mc uses 9-slice scaling, which does not work well with nested movie clips.

The seekBar_mc clip has 9-slice scaling applied to it, so the borders won't distort when it scales. The seekBarProgress_mc clip also scales, but it does distort. It does not use 9-slice scaling because it is a fill, which looks fine when distorted.

The seekBarProgress_mc clip works without a fill_mc, much like the way a progress_mc clip works in FLV Playback Custom UI components. In other words, it is not masked and is scaled horizontally. The exact dimensions of the seekBarProgress_mc at 100% are defined by left and right margins within the seekBarProgress_mc clip. These dimensions are, by default, equal and based on the difference between the *x* (horizontal) positions of seekBar_mc and seekBarProgress_mc. You can customize the dimensions with ActionScript in the seek bar movie clip, as shown in the following example:

```
this.seekBar_mc.progressLeftMargin = 2;
this.seekBar_mc.progressRightMargin = 2;
this.seekBar_mc.progressY = 11;
this.seekBar_mc.fullnessLeftMargin = 2;
this.seekBar_mc.fullnessRightMargin = 2;
this.seekBar_mc.fullnessY = 11;
```

You can put this code either in the SeekBar movie clip Timeline or you could put it with the other ActionScript code on the main Timeline. If you customize with code instead of modifying the layout, the fill doesn't need to be on the Stage. It just needs to be in the library, set to export for ActionScript on Frame 1 with the correct class name.

As with the FLV Playback Custom UI SeekBar component, it is possible to create a fullness movie clip for the seek bar. If your seek bar does not need to scale, or if it does scale but does not use 9-slice scaling, you could set up your progress_mc or fullness_mc using any of the methods used for FLV Playback Custom UI components. For more information, see .

Because the volume bar in the Flash skins does not scale, it is constructed the same way as the VolumeBar FLV Playback Custom UI component. For more information, see "SeekBar and VolumeBar components" on page 151. The exception is that the handle is implemented differently.

Seekbar and VolumeBar handles

The SeekBar and VolumeBar handles are placed on the Layout layer next to the bar. By default, the handle's left margin, right margin, and *y*-axis values are set by its position relative to the bar movie clip. The left margin is set by the difference between the handle's *x* (horizontal) location and the bar's *x* (horizontal) location, and the right margin is equal to the left margin. You can customize these values through ActionScript in the SeekBar or VolumeBar movie clip. The following example is the same ActionScript code that is used with the FLV Playback Custom UI components:

```
this.seekBar_mc.handleLeftMargin = 2;
this.seekBar_mc.handleRightMargin = 2;
this.seekBar_mc.handleY = 11;
```

You can put this code either in the SeekBar movie clip Timeline or you could put it with the other ActionScript code on the main Timeline. If you customize with code instead of modifying the layout, the handle doesn't need to be on the Stage. It just needs to be in the library, set to export for ActionScript on Frame 1 with the correct class name.

Beyond these properties, the handles are simple movie clips, set up the same way as they are in the FLV Playback Custom UI components. Both have rectangle backgrounds with the `alpha` property set to 0. These are present only to increase the hit region and are not required.

Background and foreground clips

The movie clips chrome_mc and forwardBackBorder_mc are implemented as background clips.

Of the ForwardBackBorder, ForwardBorder, and BackBorder movie clips on the Stage and the placeholder Forward and Back buttons, the only one that is *not* on a guide layer is ForwardBackBorder. It is only in the skins that actually use the Forward and Back buttons.

The only requirement for these clips is that they need to be exported for ActionScript on Frame 1 in the library.

Modify skin behavior

The `bufferingBarHidesAndDisablesOthers` property and the `skinAutoHide` property allow you to customize the behavior of your FLVPlayback skin.

Setting the `bufferingBarHidesAndDisablesOthers` property to `true` causes the FLVPlayback component to hide the SeekBar and its handle as well as disable the Play and Pause buttons when the component enters the buffering state. This can be useful when an FLV file is streaming from FMS over a slow connection with a high setting for the `bufferTime` property (10, for example). In this situation, an impatient user might try to start seeking by clicking the Play and Pause buttons, which could delay playing the file even longer. You can prevent this activity by setting `bufferingBarHidesAndDisablesOthers` to `true` and disabling the SeekBar element and the Pause and Play buttons while the component is in the buffering state.

The `skinAutoHide` property affects only predesigned skin SWF files and not controls created from the FLV Playback Custom UI components. If set to `true`, the FLVPlayback component hides the skin when the mouse is not over the viewing area. The default value of this property is `false`.

Use a SMIL file

To handle multiple streams for multiple bandwidths, the VideoPlayer class uses a helper class (NCManager) that supports a subset of SMIL. SMIL is used identify the location of the video stream, the layout (width and height) of the FLV file, and the source FLV files that correspond to the different bandwidths. It can also be used to specify the bit rate and duration of the FLV file.

Use the `source` parameter or the FLVPlayback.source property (ActionScript) to specify the location of a SMIL file. For more information, see "Specify the FLVPlayback source parameter" on page 135 and the `FLVPlayback.source` property in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

The following example shows a SMIL file that streams multiple bandwidth FLV files from a FMS using RTMP:

```
<smil>
<head>
<meta base="rtmp://myserver/myapp/" />
<layout>
<root-layout width="240" height="180" />
</layout>
</head>
<body>
            <switch>
                    <ref src="myvideo_cable.flv" dur="3:00.1"/>
                    <video src="myvideo_isdn.flv" system-bitrate="128000" dur="3:00.1"/>
                    <video src="myvideo_mdm.flv" system-bitrate="56000"dur="3:00.1"/>
            </switch>
</body>
</smil>
```

The `<head>` tag may contain the `<meta>` and `<layout>` tags. The `<meta>` tag supports only the `base` attribute, which is used to specify the URL of the streaming video (RTMP from a FMS).

The `<layout>` tag supports only the `root-layout` element, which is used to set the `height` and `width` attributes, and, therefore, determines the size of the window in which the FLV file is rendered. These attributes accept only pixel values, not percentages.

Within the body of the SMIL file, you can either include a single link to a FLV source file or, if you're streaming multiple files for multiple bandwidths from a FMS (as in the previous example), you can use the `<switch>` tag to list the source files.

The `video` and `ref` tags within the `<switch>` tag are synonymous—they both can use the `src` attribute to specify FLV files. Further, each can use the `region`, `system-bitrate`, and `dur` attributes to specify the region, the minimum bandwidth required, and the duration of the FLV file.

Within the `<body>` tag, only one occurrence of either the `<video>`, `<src>`, or `<switch>` tag is allowed.

The following example shows a progressive download for a single FLV file that does not use bandwidth detection:

```
<smil>
        <head>
            <layout>
                <root-layout width="240" height="180" />
            </layout>
        </head>
        <body>
            <video src=""myvideo.flv" />
        </body>
</smil>
```

<smil>

Availability
Flash Professional 8.

Usage
```
<smil>
...
child tags
...
</smil>
```

Attributes
None.

Child tags
`<head>, <body>`

Parent tag
None.

Description
Top-level tag, which identifies a SMIL file.

Example
The following example shows a SMIL file specifying three FLV files:

```
<smil>
<head>
<meta base="rtmp://myserver/myapp/" />
<layout>
<root-layout width="240" height="180" />
</layout>
</head>
<body>
<switch>
                <ref src="myvideo_cable.flv" dur="3:00.1"/>
                <video src="myvideo_isdn.flv" system-bitrate="128000" dur="3:00.1"/>
                <video src="myvideo_mdm.flv" system-bitrate="56000"dur="3:00.1"/>
            </switch>
</body>
</smil>
```

<head>

Availability

Flash Professional 8.

Usage

```
<head>
...
child tags
...
</head>
```

Attributes

None.

Child tags

`<meta>`, `<layout>`

Parent tag

`<smil>`

Description

Supporting the `<meta>` and `<layout>` tags, specifies the location and default layout (height and width) of the source FLV files.

Example

The following example sets the root layout to 240 pixels by 180 pixels:

```
<head>
    <meta base="rtmp://myserver/myapp/" />
    <layout>
        <root-layout width="240" height="180" />
    </layout>
</head>
```

<meta>

Availability
Flash Professional 8.

Usage
```
<meta/>
```

Attributes
```
base
```

Child tags
```
<layout>
```

Parent tag
None.

Description
Contains the `base` attribute which specifies the location (RTMP URL) of the source FLV files.

Example
The following example shows a meta tag for a base location on `myserver`:

```
<meta base="rtmp://myserver/myapp/" />
```

<layout>

Availability
Flash Professional 8.

Usage
```
<layout>
...
child tags
...
</layout>
```

Attributes
None.

Child tags
```
<root-layout>
```

Parent tag
```
<meta>
```

Description
Specifies the width and height of the FLV file.

Example

The following example specifies the layout of 240 pixels by 180 pixels:

```
<layout>
    <root-layout width="240" height="180" />
</layout>
```

<root-layout>

Availability

Flash Professional 8.

Usage

```
<root-layout...attributes.../>
```

Attributes

Width, height

Child tags

None.

Parent tag

```
<layout>
```

Description

Specifies the width and height of the FLV file.

Example

The following example specifies the layout of 240 pixels by 180 pixels:

```
<root-layout width="240" height="180" />
```

<body>

Availability

Flash Professional 8.

Usage

```
<body>
...
child tags
...
</body>
```

Attributes

None.

Child tags

`<video>, <ref>, <switch>`

Parent tag

`<smil>`

Description

Contains the `<video>`, `<ref>`, and `<switch>` tags, which specify the name of the source FLV file, the minimum bandwidth, and the duration of the FLV file. The `system-bitrate` attribute is supported only when using the `<switch>` tag. Within the `<body>` tag, only one instance of either the `<switch>`, `<video>`, or `<ref>` tag is allowed.

Example

The following example specifies three FLV files, two using the `video` tag, and one using the `ref` tag:

```
<body>
    <switch>
        <ref src="myvideo_cable.flv" dur="3:00.1"/>
        <video src="myvideo_isdn.flv" system-bitrate="128000" dur="3:00.1"/>
        <video src="myvideo_mdm.flv" system-bitrate="56000"dur="3:00.1"/>
    </switch>
</body>
```

<video>

Availability

Flash Professional 8.

Usage

`<video...attributes.../>`

Attributes

`src, system-bitrate, dur`

Child tags

None.

Parent tag

`<body>`

Description

Synonymous with the `<ref>` tag. Supports the `src` and `dur` attributes, which specify the name of the source FLV file and its duration. The `dur` attribute supports the full (00:03:00:01) and partial (03:00:01) time formats.

Example

The following example sets the source and duration for a video:

```
<video src="myvideo_mdm.flv" dur="3:00.1"/>
```

<ref>

Availability
Flash Professional 8.

Usage
```
<ref...attributes.../>
```

Attributes
`src, system-bitrate, dur`

Child tags
None.

Parent tag
`<body>`

Description
Synonymous with `<video>` tag. Supports the `src` and `dur` attributes, which specify the name of the source FLV file and its duration. The `dur` attribute supports the full (00:03:00:01) and partial (03:00:01) time formats.

Example
The following example sets the source and duration for a video:

```
<ref src="myvideo_cable.flv" dur="3:00.1"/>
```

<switch>

Availability
Flash Professional 8.

Usage
```
<switch>
...
child tags
...
<switch/>
```

Attributes
None.

Child tags
`<video>, <ref>`

Parent tag
`<body>`

Description

Used with either the `<video>` or `<ref>` child tag to list the FLV files for multiple bandwidth video streaming. The `<switch>` tag supports the `system-bitrate` attribute, which specifies the minimum bandwidth as well as the `src` and `dur` attributes.

Example

The following example specifies three FLV files, two using the `video` tag, and one using the `ref` tag:

```
<switch>
    <ref src="myvideo_cable.flv" dur="3:00.1"/>
    <video src="myvideo_isdn.flv" system-bitrate="128000" dur="3:00.1"/>
    <video src="myvideo_mdm.flv" system-bitrate="56000"dur="3:00.1" />
</switch>
```

Chapter 7: Using the FLVPlayback Captioning Component

The FLVPlayback component lets you include a video player in your Adobe Flash CS5 Professional application to play downloaded Adobe Flash Video (FLV or F4V) files and streaming FLV or F4V files. For more information on FLVPlayback, see “Using the FLVPlayback Component” on page 131.

The FLVPlaybackCaptioning component allows you to include closed captioning support for your video. The captioning component supports W3C standard XML format Timed Text and includes these features:

Captioning with Embedded Event Cue Points Associate embedded event cue points in an FLV file with XML to provide captioning instead of using a Timed Text XML file.

Multiple FLVPlayback Captioning Create multiple FLVPlayback captioning instances for multiple FLVPlayback instances.

Toggle Button Control Provide user interaction with captioning through a captioning toggle button.

Use the FLVPlaybackCaptioning component

You use the FLVPlaybackCaptioning component with one or more FLVPlayback components. In the simplest scenario, you drag a FLVPlayback component on stage, drag a FLVPlaybackCaptioning component on the same stage, identify your caption’s URL, and set captions to display. In addition, you can also set various parameters to customize your FLVPlayback captioning.

Add captioning to the FLVPlayback component

You can add the FLVPlaybackCaptioning component to any FLVPlayback component. For information on adding FLVPlayback components to your application, see “Create an application with the FLVPlayback component” on page 132.

Add the FLVPlaybackCaptioning component from the Components panel:

1 In the Components panel, open the Video folder.

2 Drag (or double-click) the FLVPlaybackCaptioning component and add it to the same stage as the FLVPlayback component to which you want to add captioning.

   Note: *Adobe provides two files to help you learn the FLVPlaybackCaptioning component quickly: caption_video.flv (a FLVPlayback sample) and caption_video.xml (a captioning sample). Access these files at www.helpexamples.com/flash/video/caption_video.flv and www.helpexamples.com/flash/video/caption_video.xml.*

3 (Optional) Drag the CaptionButton component to the same stage as the FLVPlayback and FLVPlaybackCaptioning components. The CaptionButton component enables a user to turn captioning on and off.

   Note: *To enable the CaptionButton component, you must drag it to the same stage as the FLVPlayback and FLVPlaybackCaptioning components.*

4 With the FLVPlaybackCaptioning component selected on the Stage, on the Parameters tab of the Property inspector, specify the following required information:

- Set `showCaptions` to `true`.
- Specify the `source` of the Timed Text XML file to download.

While working in Flash to test your captions, you should set the `showCaptions` property to `true`. However, if you include the `CaptionButton` component to allow users to turn captioning on and off, you should set the `showCaptions` property to `false`.

Other parameters are available to help you customize the FLVPlaybackCaptioning component. For more information, see "Customize the FLVPlaybackCaptioning component" on page 176 and the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

5 Select Control > Test Movie to start the video.

Create an instance dynamically using ActionScript:

1 Drag the FLVPlayback component from the Component panel to the Library panel (Windows > Library).

2 Drag the FLVPlaybackCaptioning component from the Component panel to the Library panel.

3 Add the following code to the Actions panel on Frame 1 of the Timeline.

```
import fl.video.*;
var my_FLVPlybk = new FLVPlayback();
my_FLVPlybk.x = 100;
my_FLVPlybk.y = 100;
addChild(my_FLVPlybk);
my_FLVPlybk.skin = "install_drive:/Program Files/Adobe/Adobe Flash 
CS5/en/Configuration/FLVPlayback Skins/ActionScript 3.0/SkinUnderPlaySeekCaption.swf";
my_FLVPlybk.source = "http://www.helpexamples.com/flash/video/caption_video.flv";
var my_FLVPlybkcap = new FLVPlaybackCaptioning();
addChild (my_FLVPlybkcap);
my_FLVPlybkcap.source = "http://www.helpexamples.com/flash/video/caption_video.xml";
my_FLVPlybkcap.showCaptions = true;
```

4 Change *install_drive* to the drive on which you installed Flash, and modify the path to reflect the location of the Skins folder for your installation:

***Note:** If you create an FLVPlayback instance with ActionScript, you must also assign a skin to it dynamically by setting the skin property with ActionScript. When you apply a skin with ActionScript, it is not automatically published with the SWF file. Copy the skin SWF file and the application SWF file to your server, or the skin SWF file won't be available when a user executes it.*

Set the FLVPlaybackCaptioning component parameters

For each instance of the FLVPlaybackCaptioning component, you can set the following parameters in the Property inspector or in the Component inspector to further customize the component. The following list identifies and provides a brief explanation of the properties:

`autoLayout` Determines if the FLVPlaybackCaptioning component controls the size of the captioning area. The default is `true`.

`captionTargetName` Identifies the TextField or MovieClip instance name containing captions. The default is auto.

`flvPlaybackName` Identifies the FLVPlayback instance name that you want to caption. The default is auto.

`simpleFormatting` Limits formatting instructions of the Timed Text XML file when set to true. The default is `false`.

`showCaptions` Determines if captions display. The default is `true`.

`source` Identifies the location of the Timed Text XML file.

For more information on all of the FLVPlaybackCaptioning parameters, see the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Specify the source parameter

Use the `source` parameter to specify the name and location of the Timed Text XML file that contains the captions for your movie. Enter the URL path directly in the source cell in the Component inspector.

Display captions

To view captioning, set the `showCaptions` parameter to `true`.

For more information on all of the FLVPlaybackCaptioning component parameters, see the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

In the previous examples, you learned how to create and enable the FLVPlaybackCaptioning component to display captions. There are two sources you can use for your captions: (1) a Timed Text XML file containing your captions, or (2) an XML file with captioning text that you associate with embedded event cue points.

Use Timed Text captions

The FLVPlaybackCaptioning component enables captioning for the associated FLVPlayback component by downloading a Timed Text (TT) XML file. For more information about Timed Text format, review the AudioVideo Timed Text information at www.w3.org.

This section provides an overview of the supported Timed Text tags, the required captioning file tags, and an example of a Timed Text XML file. For detailed information on all the supported Timed Text tags, see "Timed Text tags" on page 170.

The FLVPlaybackCaptioning component supports the following Timed Text tags:

Category	Task
Paragraph formatting support	Align a paragraph right, left, or center
Text formatting support	• Set the size of the text with absolute pixel sizes or delta style (for example, +2, -4) • Set the text color and font • Make text bold and italic • Set text justification
Other formatting support	• Set the background color of the TextField for captions • Set the background color of the TextField for captions to transparent (alpha 0) • Set the word wrap of the TextField for captions (on or off)

The FLVPlaybackCaptioning component matches the time code of the FLV file. Every caption must have a `begin` attribute, which determines when the caption should appear. If the caption does not have a `dur` or `end` attribute, the caption disappears when the next caption appears, or when the FLV file ends.

The following is an example of a Timed Text XML file. This file (caption_video.xml) provides captioning for the caption_video.flv file. Access these files at www.helpexamples.com/flash/video/caption_video.flv and www.helpexamples.com/flash/video/caption_video.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
    <tt xml:lang="en"
xmlns="http://www.w3.org/2006/04/ttaf1"xmlns:tts="http://www.w3.org/2006/04/ttaf1#styling">
<head>
    <styling>
<style id="1" tts:textAlign="right"/>
<style id="2" tts:color="transparent"/>
<style id="3" style="2" tts:backgroundColor="white"/>
<style id="4" style="2 3" tts:fontSize="20"/>
    </styling>
</head>
<body>
    <div xml:lang="en">
<p begin="00:00:00.00" dur="00:00:03.07">I had just joined <span
tts:fontFamily="monospaceSansSerif,proportionalSerif,TheOther"tts:fontSize="+2">Macromedia</
span> in 1996,</p>
<p begin="00:00:03.07" dur="00:00:03.35">and we were trying to figure out what to do about the
internet.</p>
<p begin="00:00:06.42" dur="00:00:03.15">And the company was in dire straights at the time.</p>
<p begin="00:00:09.57" dur="00:00:01.45">We were a CD-ROM authoring company,</p>
<p begin="00:00:11.42" dur="00:00:02.00">and the CD-ROM business was going away.</p>
<p begin="00:00:13.57" dur="00:00:02.50">One of the technologies I remember seeing was
Flash.</p>
<p begin="00:00:16.47" dur="00:00:02.00">At the time, it was called <span
tts:fontWeight="bold" tts:color="#ccc333">FutureSplash</span>.</p>
<p begin="00:00:18.50" dur="00:00:01.20">So this is where Flash got its start.</p>
<p begin="00:00:20.10" dur="00:00:03.00">This is smart sketch running on the <span
tts:fontStyle="italic">EU-pin computer</span>,</p>
<p begin="00:00:23.52" dur="00:00:02.00">which was the first product that FutureWave did.</p>
<p begin="00:00:25.52" dur="00:00:02.00">So our vision for this product was to</p>
<p begin="00:00:27.52" dur="00:00:01.10">make drawing on the computer</p>
<p begin="00:00:29.02" dur="00:00:01.30" style="1">as <span tts:color="#ccc333">easy</span>
as drawing on paper.</p>
</div>
</body>
</tt>
```

Timed Text tags

The FLVPlaybackCaptioning component supports Timed Text tags for captioning XML files. For more information about the Audio Video Timed Text tags, review information at www.w3.org. The following table lists supported and non-supported tags.

Function	Tag/Value	Usage/Description	Example
Ignored tags	metadata	Ignored / allowed at any level of the document	
	set	Ignored / allowed at any level of the document	
	xml:lang	Ignored	

Function	**Tag/Value**	**Usage/Description**	**Example**
	xml:space	Ignored / Behavior overrides to: xml:space="default"	
	layout	Ignored / including any region tags in a layout tag section	
	br tag	All attributes and contents are ignored.	
Media Timing for Captions	begin attributes	Allowed in p tags only. Required for media time deployment of captions.	<p begin="3s">
	dur attributes	Allowed in p tags only. Recommended. If not included, the caption ends with the FLV file or when another caption starts.	
	end attributes	Allowed in p tags only. Recommended. If not included, the caption ends with the FLV file or when another caption starts.	
Clock Timing for Captions	00:03:00.1	Full clock format	
	03:00.1	Partial clock format	
	10	Offset times without units. Offset represents seconds.	
	00:03:00:05 00:03:00:05.1 30f 30t	Not supported. Time formats that include frames or ticks are not supported.	
Body tag	body	Required / Support for only one body tag.	<body><div>...</div></body>
Content tag	div tag	Zero or more allowed. The first tag is used.	
	p tag	Zero or more allowed.	
	span tag	A logical container for a sequence of textual content units. No support for nested spans. Support for attribute style tags.	
	br tag	Denotes an explicit line break.	
Styling Tags (All style tags are used within the p tag)	style	Reference one or more style elements. Can be used as a tag and as an attribute. As a tag, an ID attribute is required (the style can be reused in the document). Support for one or more style tags inside style tag.	
	tts:background Color	Specify a style property that defines the background color of a region. Alpha is ignored unless set to zero (alpha 0) to make the background transparent. The color format is #RRGGBBAA.	

Function	**Tag/Value**	**Usage/Description**	**Example**
	tts:color	Specify a style property that defines the foreground color. Alpha not supported for any colors. Value `transparent` translates to black.	<style id="3" style="2" tts:backgroundColor="white"/> "transparent" = #00000000 "black"=#000000FF "silver"=#C0C0C0FF "grey"=#808080FF "white"=#FFFFFFFF "maroon"=#800000FF "red"=#FF0000FF "purple"=#800080FF "fuchsia"("magenta")= #FF00FFFF "green"=#008000FF "lime"=#00FF00FF "olive"=#808000FF "yellow"=#FFFF00FF "navy"=#000080FF "blue"=#0000FFFF "teal"=#008080FF "aqua"("cyan")=#00FFFFFF
	tts:fontFamily	Specify a style property that defines the font family.	"default" = `_serif` "monospace" = `_typewriter` "sansSerif" = `_sans` "serif" = `_serif` "monospaceSansSerif" =`_typewriter` "monospaceSerif" =`_typewriter` "proportionalSansSerif" = `_sans`
	tts:fontSize	Specify a style property that defines the font size. Only the first (vertical) value is used if two are supplied. Percentage values and units are ignored. Support for absolute pixel (for example, 12) and relative style (for example +2) sizes.	
	tts: fontStyle	Specify a style property that defines the font style.	"normal" "italic" "inherit"* * The default behavior; inherits the style from the enclosing tag.

Function	**Tag/Value**	**Usage/Description**	**Example**
	tts: fontWeight	Specify a style property that defines the font weight.	"normal" "bold" "inherit"* * The default behavior; inherits the style from the enclosing tag.
	tts: textAlign	Specify a style property that defines how inline areas are aligned within a containing block area.	"left" "right" "center" "start" (="left") "end" (="right") "inherit"* *Inherits the style from the enclosing tag. If no textAlign tag is set, the default is "left".
	tts: wrapOption	Specify a style property that defines whether or not automatic line wrapping (breaking) applies within the context of the affected element. This setting affects all paragraphs in the caption element.	"wrap" "noWrap" "inherit"* *Inherits the style from the enclosing tag. If no wrapOption tag is set, the default is "wrap".
Non Supported Attributes	tts: direction tts: display tts: displayAlign tts: dynamicFlow tts: extent tts: lineHeight tts: opacity tts: origin tts: overflow tts: padding tts: showBackground tts: textOutline tts: unicodeBidi tts: visibility tts: writingMode tts: zIndex		

Use cue points with captioning

Cue points allow you to interact with a video; for example, you can affect the playing of an FLV file or display text at specific times in the video. If you don't have a Timed Text XML file to use with an FLV file, you can embed event cue points in an FLV file and then associate those cue points with text. This section provides information on the FLVPlaybackCaptioning component cue points standards and a brief overview of how to associate those cue points with text for captioning. For more information on how to embed event cue points with the Video Import wizard or the Adobe Media Encoder, "Working with Video," in *Using Flash*.

Understanding FLVPlaybackCaptioning cue point standards

Within the FLV file's metadata, a cue point is represented as an object with the following properties: `name`, `time`, `type`, and `parameters`. FLVPlaybackCaptioning ActionScript cue points have the following attributes:

`name` The `name` property is a string that contains the assigned name of the cue point. The `name` property must start with the *fl.video.caption.2.0.* prefix and follow the prefix with a string. The string is a series of positive integers that increment each time to keep each name unique. The prefix includes the version number that also matches the FLVPlayback version number. For Adobe Flash CS4 and later, you must set the version number to `2.0`.

`time` The `time` property is the time when the caption should display.

`type` The `type` property is a string whose value is `"event"`.

`parameters` The `parameters` property is an array that supports the following name-and-value pairs:

- **`text:String`** The HTML-formatted text for the caption. This text is passed to the `TextField.htmlText` property directly. The FLVPlaybackCaptioning component supports an optional `text:`*n* property, which supports the use of multiple language tracks. For more information, see "Support multiple language tracks with embedded cue points" on page 176.

- **`endTime:Number`** The time when the caption should disappear. If you do not specify this property, the FLVPlaybackCaptioning component assumes it is not a number (NaN), and the caption is displayed until the FLV file completes (the FLVPlayback instance dispatches the `VideoEvent.COMPLETE event)`. Specify the `endTime:Number` property in seconds.

- **`backgroundColor:uint`** This parameter sets the `TextField.backgroundColor`. This property is optional.

- **`backgroundColorAlpha:Boolean`** If the backgroundColor has an alpha of 0%, then the parameter sets `TextField.background = !backgroundColor`. This property is optional.

- **`wrapOption:Boolean`** This parameter sets the TextField.wordWrap. This property is optional.

Understanding captioning for event embedded cue points

If you do not have a Timed Text XML file that contains captions for your FLV file, you can create captioning by associating an XML file that contains captioning with event embedded cue points. The XML sample assumes you have performed the following steps to create event embedded cue points in your video:

- Add the event cue points (following the FLVPlaybackCaptioning standards), and encode the video.
- In Flash, drag an FLVPlayback component and an FLVPlaybackCaptioning component to the Stage.
- Set the FLVPlayback and FLVPlaybackCaptioning components' source properties (the location of your FLV file and the location of your XML file).
- Publish.

  The following sample imports XML into the encoder:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<FLVCoreCuePoints>

    <CuePoint>
        <Time>9136</Time>
        <Type>event</Type>
        <Name>fl.video.caption.2.0.index1</Name>
        <Parameters>
            <Parameter>
                <Name>text</Name>
                <Value><![CDATA[Captioning text for the first cue point]]></Value>
            </Parameter>
        </Parameters>
    </CuePoint>

    <CuePoint>
        <Time>19327</Time>
        <Type>event</Type>
        <Name>fl.video.caption.2.0.index2</Name>
        <Parameters>
            <Parameter>
                <Name>text</Name>
                <Value><![CDATA[Captioning text for the second cue point]]></Value>
            </Parameter>
        </Parameters>
    </CuePoint>

    <CuePoint>
        <Time>24247</Time>
        <Type>event</Type>
        <Name>fl.video.caption.2.0.index3</Name>
        <Parameters>
            <Parameter>
                <Name>text</Name>
                <Value><![CDATA[Captioning text for the third cue point]]></Value>
            </Parameter>
        </Parameters>
    </CuePoint>

    <CuePoint>
        <Time>36546</Time>
        <Type>event</Type>
        <Name>fl.video.caption.2.0.index4</Name>
        <Parameters>
            <Parameter>
                <Name>text</Name>
                <Value><![CDATA[Captioning text for the fourth cue point]]></Value>
            </Parameter>
        </Parameters>
    </CuePoint>

</FLVCoreCuePoints>
```

The FLVPlaybackCaptioning component also supports multiple language tracks with embedded cue point. For more information, see “Support multiple language tracks with embedded cue points” on page 176.

Support multiple language tracks with embedded cue points

The FLVPlaybackCaptioning `track` property supports multiple language tracks with embedded cue points, as long as the Timed Text XML file follows the FLVPlaybackCaptioning cue point standards. (For more information, see "Understanding FLVPlaybackCaptioning cue point standards" on page 174.) However, the FLVPlaybackCaptioning component does not support multiple language tracks in separate XML files. To use the `track` property, set the property to a value not equal to 0. For example, if you set the track property to 1 (`track == 1`), the FLVPlaybackCaptioning component will search the cue point parameters. If a match is not found, the text property in the cue point parameters is used. For more information, see the `track` property in the *ActionScript 3.0 Reference for the Adobe Flash Platform*.

Play multiple FLV files with captioning

You can open multiple video players within a single instance of the FLVPlayback component to play multiple videos and switch between them as they play. You can also associate captioning with each video player within the FLVPlayback component. For more information on how to open multiple video players, see "Use multiple video players" on page 146. To use captioning in multiple video players, create one instance of the FLVPlaybackCaptioning component for each `VideoPlayer` and set the FLVPlaybackCaptioning `videoPlayerIndex` to the corresponding index. The `VideoPlayer` index defaults to 0 when only one `VideoPlayer` exists.

The following is an example of code that assigns unique captioning to unique videos. To run this example, the fictional URLs in the example must be replaced with working URLs.

```
captioner0.videoPlayerIndex = 0;
captioner0.source = "http://www.[yourDomain].com/mytimedtext0.xml";
flvPlayback.play("http://www.[yourDomain].com/myvideo0.flv");
captioner1.videoPlayerIndex = 1;
captioner1.source = "http://www.[yourDomain].com/mytimedtext1.xml";
flvPlayback.activeVideoIndex = 1;
flvPlayback.play ("http://www.[yourDomain].com/myvideo1.flv");
```

Customize the FLVPlaybackCaptioning component

To start using the FLVPlaybackCaptioning component quickly, you can choose to use the FLVPlaybackCaptioning defaults which place the captioning directly over the FLVPlayback component. You may want to customize the FLVPlaybackCaptioning component to move the captioning away from the video.

The following code demonstrates how to dynamically create an FLVPlayback object with the toggle captioning button:

1 Place the FLVPlayback component on the stage at 0,0 and provide the instance name **player**.

2 Place the FLVPlaybackCaptioning component on the stage at 0,0 and provide the instance name **captioning**.

3 Place the CaptionButton component on the stage.

4 In the following code example, set the `testVideoPath:String` variable to an FLV file (using an absolute or relative path).

   ***Note:** The code example sets the `testVideoPath` variable to the Flash video sample, `caption_video.flv`. Change this variable to the path of the captioning video component to which you are adding a caption button component.*

5 In the following code example, set the `testCaptioningPath:String` variable to an appropriate Timed Text XML file (using an absolute or relative path).

***Note:** The code example sets the `testCaptioningPath` variable to the Timed Text XML file, `caption_video.xml`. Change this variable to the path of the Timed Text XML file that contains captions for your video.*

6 Save the following code as FLVPlaybackCaptioningExample.as in the same directory as your FLA file.

7 Set the DocumentClass in the FLA file to FLVPlaybackCaptioningExample.

```
package
{
    import flash.display.Sprite;
    import flash.text.TextField;
    import fl.video.FLVPlayback;
    import fl.video.FLVPlaybackCaptioning;

    public class FLVPlaybackCaptioningExample extends Sprite {

        private var testVideoPath:String = 
"http://www.helpexamples.com/flash/video/caption_video.flv";
        private var testCaptioningPath:String = 
"http://www.helpexamples.com/flash/video/caption_video.xml";

        public function FLVPlaybackCaptioningExample() {
            player.source = testVideoPath;
            player.skin = "SkinOverAllNoCaption.swf";
            player.skinBackgroundColor = 0x666666;
            player.skinBackgroundAlpha = 0.5;

            captioning.flvPlayback = player;
            captioning.source = testCaptioningPath;
            captioning.autoLayout = false;
            captioning.addEventListener("captionChange",onCaptionChange);
        }
        private function onCaptionChange(e:*):void {
            var tf:* = e.target.captionTarget;
            var player:FLVPlayback = e.target.flvPlayback;

            // move the caption below the video
            tf.y = 210;
        }
    }
}
```

For more information on all of the FLVPlaybackCaptioning parameters, see the *ActionScript 3.0 Reference for the Adobe Flash Platform*.