



macromedia®
CENTRAL™

Building Central Applications with Components

Trademarks

1 Step RoboPDF, ActiveEdit, ActiveTest, Authorware, Blue Sky Software, Blue Sky, Breeze, Breezo, Captivate, Central, ColdFusion, Contribute, Database Explorer, Director, Dreamweaver, Fireworks, Flash, FlashCast, FlashHelp, Flash Lite, FlashPaper, Flex, Flex Builder, Fontographer, FreeHand, Generator, HomeSite, JRun, MacRecorder, Macromedia, MXML, RoboEngine, RoboHelp, RoboInfo, RoboPDF, Roundtrip, Roundtrip HTML, Shockwave, SoundEdit, Studio MX, UltraDev, and WebHelp are either registered trademarks or trademarks of Macromedia, Inc. and may be registered in the United States or in other jurisdictions including internationally. Other product names, logos, designs, titles, words, or phrases mentioned within this publication may be trademarks, service marks, or trade names of Macromedia, Inc. or other entities and may be registered in certain jurisdictions including internationally.

Third-Party Information

This guide contains links to third-party websites that are not under the control of Macromedia, and Macromedia is not responsible for the content on any linked site. If you access a third-party website mentioned in this guide, then you do so at your own risk. Macromedia provides these links only as a convenience, and the inclusion of the link does not imply that Macromedia endorses or accepts any responsibility for the content on those third-party sites.

Copyright © 1997-2005 Macromedia, Inc. All rights reserved. This manual may not be copied, photocopied, reproduced, translated, or converted to any electronic or machine-readable form in whole or in part without written approval from Macromedia, Inc. Notwithstanding the foregoing, the owner or authorized user of a valid copy of the software with which this manual was provided may print out one copy of this manual from an electronic version of this manual for the sole purpose of such owner or authorized user learning to use such software, provided that no part of this manual may be printed out, reproduced, distributed, resold, or transmitted for any other purposes, including, without limitation, commercial purposes, such as selling copies of this documentation or providing paid-for support services.

Acknowledgments

Project Management: JuLee Burdekin

Writing: Jay Armstrong, Jody Bleyle, Alec Flett, David Jacowitz, Shimi Rahim

Editing Management: Rosana Francescato

Editing: Mary Ferguson, Mary Kraemer, Noreen Maher, Antonio Padial, Lisa Stanziano

Production Management: Patrice O'Neill

Media Design and Production: Adam Barnett, Christopher Basmajian, Aaron Begley, John Francis

Special thanks to the Macromedia Central QA and Development teams, Randy Nielsen, Vijay Shah

Third Edition: January 2005

Macromedia, Inc.
600 Townsend St.
San Francisco, CA 94103

CONTENTS

INTRODUCTION: About This Guide	5
System requirements	5
Installing Macromedia Central components	5
Guide to instructional media	6
CHAPTER 1: Using Macromedia Central Components	7
Component changes in the Macromedia Central SDK	8
Migrating from previous versions of Central	9
Design considerations	9
Macromedia Central artwork	10
Coding considerations	11
About accessibility	11
Testing components in your application	11
Writing event listeners for components	11
CHAPTER 2: Components Reference	15
Accordion component	17
AccordionTab component	36
Alert component	53
Button component	69
CheckBox component	83
CloseButton component	95
ComboBox component	96
DataGrid component	132
DateChooser component	171
DateField component	186
DialogBox component	207
ExpandingPod component	222
FocusManager class	231
IconButton component	242
IconMenu component	246
Label component	271
List component	277
Loader component	311
Menu component	324
MenuBar component	359

NumericStepper component	373
ProgressBar component	385
RadioButton component	402
RoundIconButton component	416
ScrollPane component	421
SimpleButton class	439
TextInput component	445
TextArea component	459
TossButton component	476
Tree component	477
Window component	505
UIComponent class	521
UIEventDispatcher class	530
UIObject class	536
UIScrollBar component	557

INTRODUCTION

About This Guide

The Macromedia Central SDK includes new versions of selected user interface components and several new components to help you create distinctive, elegant interfaces for your rich Internet applications. This guide includes usage scenarios and procedural samples for using the Central components, as well as descriptions of the component methods in alphabetical order.

This guide is intended to be used with the Central SDK and Macromedia Flash MX, Macromedia Flash MX 2004, or Macromedia Flash MX Professional 2004. It helps you create applications that are consistent with the Macromedia Central environment.

This guide assumes that you have installed Flash MX, Flash MX 2004, or Flash MX Professional 2004 and know how to use it and know how to use components. You should already be familiar with developing applications with components in Flash MX or Flash MX 2004, writing ActionScript, and using Flash Player. Before using Macromedia Central components, read *Developing Central Applications*.

For basic information about components, see the help system in your Flash authoring tool. Also, make sure you frequently check the Macromedia Central DevNet website at www.macromedia.com/go/central_dev_center for articles about components and developing for Macromedia Central in general.

System requirements

The system requirements for using Macromedia Central components are the same as for the authoring tool.

You can use Macromedia Central components with Flash MX, Flash MX 2004, or Flash MX Professional 2004, in Windows or on the Macintosh.

Installing Macromedia Central components

Use the Extensions Manager in your Flash authoring tool to download and install the Macromedia Central Components MXP file.

To install Macromedia Central components:

1. Start the Flash authoring tool.
2. Select Help > Manage Extensions.
The Macromedia Extensions Manager opens.
3. In the Macromedia Extensions Manager, select File > Install Extension.
4. Browse to the Central Components MXP file in the Components folder in the Macromedia Central SDK folder and click Install.

After the component is installed, you are asked to exit and restart the Flash authoring tool.

5. Start the Flash authoring tool again.
6. Open the Components panel.
7. Click the options menu icon and select Central Components.

The Central components appear in the Components panel.

Guide to instructional media

The documentation for Macromedia Central includes the following items:

- This book, *Building Central Applications with Components*.
- *Developing Central Applications*, a guide that describes the Central architecture, the application development workflow, and the elements that make up a Central application. It explains how you use those elements, and it includes samples and information about testing and deploying applications and information about security.
- Using Macromedia Central Help, an online help system for users of Central and Central applications. It includes detailed information on the features and functions of the Central interface. You can access the help system from the Help menu.

Typographical conventions

The following typographical conventions are used in this book:

- `Code font` indicates ActionScript statements, XML tag and attribute names, and literal text used in examples.
- *Italic* indicates placeholder elements in code or paths. For example, `\settings\myPrinter\` means that you should specify your own location for *myPrinter*.

Additional resources

The following additional resources are important sources of information before, during, and after development of Macromedia Central applications:

- The Macromedia Central DevNet website at www.macromedia.com/go/central_dev_center is an excellent resource for information about developing for Macromedia Central.
- The Macromedia Central Support Center at www.macromedia.com/go/central_support provides information on new features, contains tech notes documenting technical issues, and lets you search the Support Center database.

CHAPTER 1

Using Macromedia Central Components

The Macromedia Central Software Development Kit (SDK) supports some version 2 components from the Macromedia Flash MX 2004 Authoring tool, as well as several Central-specific UI components.

You can use Flash MX 2004 or Flash MX Professional 2004 to develop Macromedia Central applications.

Although a wide range of developers can create Central applications, all Central applications reside in the Central environment. You can use components shipped with the authoring tool in Central applications, but Macromedia recommends that you use the Central version of a component in your Central applications when a Central version is available. For example, if you want a data grid in your Central application, use the Central DataGrid component rather than the version of the DataGrid component shipped with the authoring tool.

Using the Central components ensures that your application can take advantage of new, powerful functionality and performance optimizations introduced with the Central SDK. Because Central components are linked to <http://download.macromedia.com>, your total application size is smaller when you use the Central components. Finally, using the Central components helps to ensure that your application has a recognizable look and feel that is similar to other applications that run in Central. A standard application design that follows Macromedia guidelines provides consistency for users and helps make Central applications easy to use.

To achieve further consistency, Macromedia encourages developers to follow design guidelines outlined in this book and in the articles on the Macromedia Central DevNet website at www.macromedia.com/go/central_dev_center.

Component changes in the Macromedia Central SDK

The Macromedia Central SDK contains a number of components that are specific to Central. Central 1.5 updates some version 1.0 components, deprecates some Central 1.0 components, and introduces many new *version 2* components.

Updated version 1 components

The following table lists the Central 1.0 components that have been updated for this release, and provides a reference to the documentation later in this book.

Central 1.0 class	Updated component class
MAccordionTab	See the “AccordionTab component” on page 36.
MCloseButton	See the “Button component” on page 69.
MDialogBox	See the “DialogBox component” on page 207.
MExpandingPod	See the “ExpandingPod component” on page 222.
MIconButton	See the “IconButton component” on page 242, which contains new icons for buttons in your application: Close, Print, Toss, Add to Favorites, Remove from Favorites, and Set Notices.
MRoundIconButton	See the “RoundIconButton component” on page 416.
MTossButton	See the “TossButton component” on page 476.

Deprecated component classes

The following table lists the deprecated Central 1.0 components, and provides a reference to an alternative component described later in this book:

Deprecated class	Alternative component class
MCalendar	See the “DateChooser component” on page 171.
MCheckBox	See the “CheckBox component” on page 83.
MComboBox	See the “ComboBox component” on page 96.
MDataGrid	See the “DataGrid component” on page 132.
MListBox	See the “List component” on page 277.
MProgressBar	See the “ProgressBar component” on page 385.
MPushButton	See the “Button component” on page 69.
MRadioButton	See the “RadioButton component” on page 402.
MScrollBar	See the “UIScrollBar component” on page 557.
MScrollPane	See the “ScrollPane component” on page 421.
MTextInputField	See the “TextInput component” on page 445.

Migrating from previous versions of Central

The new version 2 components provided in the Macromedia Central SDK have new methods and properties and class names that are incompatible with previous components. When you update an application to use the new version 2 components, refer to the Component Dictionary to learn about the new methods and properties.

The version 2 components have a fundamentally different architecture and API than previous versions of the components. The following are some of the primary differences:

Direct access to properties Component properties are now directly accessible using properties rather than methods. For example, to set the label property of a button, you write `button.label = "Submit"` rather than `button.setLabel("Submit")`.

New event model Event listeners should be used to handle events such as mouse clicks and keyboard entry. Use the `addEventListener()` method on most Components instead of calling `setChangeHandler()`.

Intrinsic classes ActionScript 2.0 requires intrinsic classes to be installed in order for the compiler to be able to enforce strict types. In previous versions of the components, class names were usually the component name prefixed with the letter M, such as `MCheckBox`. In version 2 of the components, the letter M prefix has been removed from the class names. For example, `MCheckBox` has been replaced by `CheckBox`. Some Central components retained the M prefix.

New component names Many of the components have changed names in addition to the removal of the M prefix. For example, `MPushButton` is now `Button`. Other components have been replaced. The `MCalendar` component has been split into `DateField` and `DateChooser`.

To upgrade a component from previous versions to version 2, perform the following steps for each instance of the component:

1. Select the old component on the stage. Make note of the instance name, the X and Y positions, the width and height, and any properties that you may have changed for your application.
2. Drag the new version of the component onto the stage, near the old component.
3. Set the instance name of the new component to be the same as the old component, and update any properties that you changed in the old component.
4. Delete the old component.
5. Set the X and Y positions, the height, and width of your new component.

After you have updated the component on the stage, search for references to that component in your ActionScript and update API calls and event handling as appropriate.

Design considerations

The Central components provide an updated look and feel and ensure consistency between different Central applications. The Central SDK allows for brand differentiation in other aspects of application design, such as the use of your own logo and colors. In addition to components, new artwork for your applications is included (see [“Macromedia Central artwork” on page 10](#)).

Unlike other Flash user interface components, Central components cannot be reskinned, and the use of styles is not supported. Using the same components across different Central applications makes it easier for users to learn how to use Central applications, and ensures that users have a consistent experience interacting with Central applications.

However, you can customize the look of most components in several ways:

- You can set text font and size.
To provide maximum consistency across Central applications, Macromedia strongly recommends that you use the default font settings, which are Verdana 10-point font, text color #2B333C, and disabled text color #AAB3B3. You can also use Trebuchet MS for titles or headings. The component descriptions include information about exceptions.
- You can customize the size of many components so they fit best in your application.

When you design the interface for your applications, keep in mind that many components have a colored border or outline around them during certain states, to indicate the status for the user. A green border indicates focus, selection, emphasis, or pressed states, and a red border indicates errors. For example, when a user moves the mouse over a PushButton component instance, the green border appears.

Macromedia Central artwork

The Macromedia Central SDK contains artwork that you can use in your applications. Access the artwork in the Components panel or Components inspector, depending on your version of Flash. These images do not support an ActionScript interface.

The Central environment and applications developed by Macromedia use these images for certain functions. You can use these images in your own applications to speed development and to maintain consistency with other Central applications:

- • **Progress animation arrow** An animated circular arrow that shows that a task is in progress. You can display the progress animation arrow when data is loading or when lengthy processing occurs, and remove it when the data has finished loading or the processing task is complete. Your pods can use the progress animation arrow to show progress within a pod, where a progress bar would not fit. You can also use the progress animation arrow anywhere to indicate progress in a limited space.
- ▼ • **Disclosure triangle** A solid triangle used to expand or collapse an area of content. Use this triangle to expand or collapse the pod viewers in the Console. Use the standard icon for the expanded state. Rotate the triangle 90 degrees counterclockwise for the collapsed state.
- • **Favorites symbol** A green bookmark ribbon to indicate a location where cached information is stored. Add this symbol to a Favorites tab or the tab that displays information saved by the user. Variations of this bookmark ribbon can be displayed on the IconButton component. For more information, see [“IconButton component” on page 242](#).
- **Gradients** Various images you can use in your applications to duplicate the subtle color or grayscale gradients that are found in the interface of the Central environment.

Additional images might have been added to the Central Artwork MXP file; for more information, see the Release Notes.

Coding considerations

Component methods do not perform error checking for type as other native ActionScript objects and actions do. Therefore, Macromedia recommends that you validate parameters before passing them to methods.

You do not need to use a constructor to access the methods of components.

Certain tasks cannot be performed during clip initialization (`#initclip`). You should set properties or call methods of components after the `#endinitclip` line in your code.

About accessibility

Central components do not support accessibility features and do not provide support for screen readers. However, users can navigate through the components on a basic level using some keyboard keys. For example, they can use arrow keys to scroll through items in a list, menu, or data grid.

Testing components in your application

Because of an issue with how the Flash authoring tool handles remote shared libraries, you cannot test the Central components in the Flash authoring environment. That is, you cannot publish a test application that uses Central components and view the application outside of Central. You must test Central components in the Central environment.

For more information, see “Testing and debugging your application within Central” in *Developing Central Applications*.

Writing event listeners for components

Each version 2 component has a set of events that occur when a user interacts with it. Each component has a method called `addEventListener` that allows code to be executed when such an event occurs.

You can write event listeners in a variety of ways. It is good coding practice to create just one event listener for each component in your document. This ensures that conflicting actions are not assigned, and makes it easier to update and change the code. An event listener always accepts one parameter, which is an object that represents the event when the instance of the component has changed.

Single-selection forms

In the following example, `listener.click` is an event listener function specified for two `CheckBox` components. The listener function accepts an event target, which holds a reference to the target component in the `target` property. The function uses a series of `if/else if` statements to determine which check box instance is selected, and enables either `listBox1_mc` or `listBox2_mc`, depending on the value of the check box instance.

```
var listener = new Object;
listener.click = function(event)
{
```

```

var component = event.target;
if (component._name=="check1") {
    listBox1_mc.enabled = component.selected;
} else if (component._name=="check2") {
    listBox2_mc.enabled = component.selected;
}
}

// connect the event listener to the component
check1.addEventListener("change", listener);
check2.addEventListener("change", listener);

```

Another way of accomplishing the same thing is to specify an event listener function for each `CheckBox` component, as the following example shows.

For the `check1` instance, add the `listener1` object as the event listener. If the user selects the `check1` instance of the check box, the list box instance `listBox1` is enabled.

```

var listener1 = new Object;
listener1.click = function(event)
{
    var component = event.target;
    listBox1_mc.enabled = component.selected;
}

// connect the event listener to the component
check1.addEventListener("change", listener1);

```

For the check box instance `check2`, add the `listener2` object as the event listener. If the user selects the `check2` instance of the check box, the list box instance `listBox2` is enabled.

```

var listener2 = new Object;
listener2.click = function(event)
{
    var component = event.target;
    listBox2_mc.enabled = component.selected;
}

// connect the event listener to the component
check2.addEventListener("change", listener2);

```

Multiple-selection forms

In a form where the user makes multiple inputs or selections using various components and then submits the completed form, you only need to specify an event listener for the component responsible for submitting the form data and exiting the form. The event listener needs to create an object with properties for storing the data, specify actions for gathering the data from all of the components in the form, and then perform an output, submit, or exit page action.

The following example is a click event listener specified for a Submit button on a form that has a check box, a group of radio buttons, and a list box. The user makes choices before pressing the Submit button to submit the form. The labels of the selected components are written to the Output panel.

```
listener.click =function( event ) {
var component = event.target;
if ( component._name == "submit"){

    // create the object to store values
    formData = new Object();

    // gather the data
    formData.checkValue = checkBox_mc.selected;
    formData.radioValue = radioButton1.selected;
    formData.listValue = listBox_mc.selectedIndex;

    // output the results
    trace(formData.listValue);
    trace(formData.radioValue);
    trace(formData.checkValue);
}
}
```


CHAPTER 2

Components Reference

The following components/classes are included with the Central SDK (for a summary of deprecated components, see [“Component changes in the Macromedia Central SDK”](#) on page 8):

Component/class name	Description
Accordion component	A navigator that contains a sequence of children that it displays one at a time.
AccordionTab component	A tab with an area for content and animation for switching among tabs.
Alert component	Lets you display a window that presents the user with a message and response buttons.
Button component	A resizable rectangular user interface button.
CheckBox component	A box that lets users select a true or false option.
CloseButton component	A close button for use in dialog boxes and pods.
ComboBox component	A text field combined with a pop-up menu that lets users select an option or enter a value.
DataGrid component	A combination of elements used to present tabular data.
DateChooser component	A calendar that allows users to select a date.
DateField component	A pop-up calendar that lets users quickly select dates.
DialogBox component	A simple window.
ExpandingPod component	A box for text and images that presents a partial view or highlight of information and lets users quickly open a larger, detailed view.
FocusManager class	The Focus Manager allows you to specify the order in which components receive focus when a user presses the Tab key, or set a button reaction when the user presses Enter (Windows) or Return (Macintosh).
IconButton component	A button you can customize with an icon of your choice.
IconMenu component	A button you can customize with an icon of your choice, which opens a pop-up menu to let users select an option.
Label component	A label component is a single line of text.

Component/class name	Description
List component	A box that lets users select options from a scrollable list.
Loader component	A container that can display a SWF or JPEG file.
Menu component	A pop-up menu that lets a user select an item.
MenuBar component	A horizontal menu bar with pop-up menus and commands
NumericStepper component	Allows a user to step through an ordered set of numbers
ProgressBar component	A box with a bar that shows the progress of a current activity.
RadioButton component	A round button that lets users select exclusively that option in a group of choices.
RoundIconButton component	A round button that you can customize with an icon of your choice.
ScrollPane component	A container for movie clips, bitmaps, or SWF files, typically used to display large forms or images in a limited area, using horizontal or vertical scroll bars.
SimpleButton class	Allows you to determine the state of a button.
TextInput component	A single-line text component that is a wrapper for the native ActionScript TextField object.
TextArea component	Wraps the native ActionScript TextField object.
TossButton component	A button that you can use in your pod to send data to the parent application.
Tree component	A hierarchical view of data.
UIComponent class	The UIComponent class does not represent a visual component; it contains methods, properties, and events that allow Macromedia components to share some common behavior.
UIEventDispatcher class	The UIEventDispatcher class is mixed in to the UIComponent class and allows components to emit certain events.
UIObject class	UIObject is the base class for all version 2 components; it is not a visual component.
UIScrollBar component	A control that lets users move through a display of information vertically or horizontally when the amount of information exceeds the display area
Window component	A pop-up dialog box that also offers selectable options, which can provide a single interface for multiple related user tasks, such as setting different categories of preferences for an application.

Accordion component

The Accordion component is a navigator that contains a sequence of children that it displays one at a time. The children must be objects that inherit from the `UIObject` class (which includes all components and screens built with version 2 of the Macromedia Component Architecture); most often, children are a subclass of the `View` class. This includes movie clips assigned to the class `mx.core.View`. To maintain tabbing order in an accordion's children, the children must also be instances of the `View` class.

An accordion creates and manages header buttons that a user can click to navigate between the accordion's children. An accordion has a vertical layout with header buttons that span the width of the component. One header is associated with each child, and each header belongs to the accordion—not to the child. When a user clicks a header, the associated child is displayed below that header. The transition to the new child uses a transition animation.

An accordion with children accepts focus, and changes the appearance of its headers to display focus. When a user tabs into an accordion, the selected header displays the focus indicator. An accordion with no children does not accept focus. Clicking components that can take focus within the selected child gives them focus. When an `Accordion` instance has focus, you can use the following keys to control it:

Key	Description
Down Arrow, Right Arrow	Moves focus to the next child header. Focus cycles from last to first without changing the selected child.
Up Arrow, Left Arrow	Moves focus to the previous child header. Focus cycles from first to last without changing the selected child.
End	Selects the last child.
Enter/Space	Selects the child associated with the header that has focus.
Home	Selects the first child.
Page Down	Selects the next child. Selection cycles from the last child to the first child.
Page Up	Selects the previous child. Selection cycles from the first child to the last child.
Shift+Tab	Moves focus to the previous component. This component may be inside the selected child, or outside the accordion; it is never another header in the same accordion.
Tab	Moves focus to the next component. This component may be inside the selected child, or outside the accordion; it is never another header in the same accordion.

The `Accordion` component cannot be made accessible to screen readers.

Using the Accordion component

You can use the Accordion component to present multipart forms. For example, a three-child accordion might present forms where the user fills out her shipping address, billing address, and payment information for an e-commerce transaction. Using an accordion instead of multiple web pages minimizes server traffic and allows the user to maintain a better sense of progress and context in an application.

Accordion parameters

You can set the following authoring parameters for each Accordion component instance in the Property inspector or in the Component inspector:

childSymbols is an array that specifies the linkage identifiers of the library symbols to be used to create the accordion's children. The default value is [] (an empty array).

childNames is an array that specifies the instance names of the accordion's children. The values you enter will be the instance names for the child symbols you specify in the `childSymbols` parameter. The default value is [] (an empty array).

childLabels is an array that specifies the text labels to use on the accordion's headers. The default value is [] (an empty array).

childIcons is an array that specifies the linkage identifiers of the library symbols to be used as the icons on the accordion's headers. The default value is [] (an empty array).

You can write ActionScript to control additional options for the Accordion component using its properties, methods, and events. For more information, see [“Accordion class” on page 26](#).

Creating an application with the Accordion component

In this example, an application developer is building the checkout section of an online store. The design calls for an accordion with three forms in which a user enters a shipping address, a billing address, and payment information. The shipping address and billing address forms are identical.

To use screens to add an Accordion component to an application:

1. In Flash, select File > New and select Flash Form Application.

2. Double-click the text Form1, and enter the name **addressForm**.

Although it doesn't appear in the library, the `addressForm` screen is a symbol of the `Screen` class. Because the `Screen` class is a subclass of the `View` class, an accordion can use it as a child.

3. With the form selected, in the Property inspector, set the form's visible property to `false`.

This hides the contents of the form in the application; the form only appears in the accordion.

4. Drag components such as Label and TextInput from the Components panel onto the form to create a mock address form; arrange them, and set their properties in the Parameters tab of the Component inspector.

Position the form elements in the upper left corner of the form. This corner of the form is placed in the upper left corner of the accordion.

5. Repeat steps 2-4 to create a screen named **checkoutForm**.

6. Create a new screen named **accordionForm**.
7. Drag an Accordion component from the Components panel to the accordionForm form, and name it **myAccordion**.
8. With myAccordion selected, in the Property inspector, do the following:
 - For the childSymbols property, enter **addressForm**, **addressForm**, and **checkoutForm**.
These strings specify the names of the screens used to create the accordion's children.
Note: The first two children are instances of the same screen, because the shipping address form and the billing address form are identical.
 - For the childNames property, enter **shippingAddress**, **billingAddress**, and **checkout**.
These strings are the ActionScript names of the accordion's children.
 - For the childLabels property, enter **Shipping Address**, **Billing Address**, and **Checkout**.
These strings are the text labels on the accordion headers.
9. Select Control > Test Movie.

To add an Accordion component to an application:

1. Select File > New and create a new Flash document.
2. Select Insert > New Symbol and name it **AddressForm**.
3. In the Create New Symbol dialog box, click the Advanced button and select Export for ActionScript. In the AS 2.0 Class field, enter **mx.core.View**.
To maintain tabbing order in an accordion's children, the children must also be instances of the View class.
4. Drag components such as Label and TextInput from the Components panel onto the Stage to create a mock address form; arrange them, and set their properties in the Parameters tab of the Component inspector.
Position the form elements in relation to 0,0 (the middle) on the Stage. The 0,0 coordinate of the movie clip is placed in the upper left corner of the accordion.
5. Select Edit > Edit Document to return to the main Timeline.
6. Repeat steps 2-5 to create a movie clip named **CheckoutForm**.
7. Drag an Accordion component from the Components panel to add it to the Stage on the main Timeline.
8. In the Property inspector, do the following:
 - Enter the instance name **myAccordion**.
 - For the childSymbols property, enter **AddressForm**, **AddressForm**, and **CheckoutForm**.
These strings specify the names of the movie clips used to create the accordion's children.
Note: The first two children are instances of the same movie clip, because the shipping address form and the billing address form are identical.
 - For the childNames property, enter **shippingAddress**, **billingAddress**, and **checkout**.
These strings are the ActionScript names of the accordion's children.

- For the `childLabels` property, enter **Shipping Address**, **Billing Address**, and **Checkout**. These strings are the text labels on the accordion headers.
 - For the `childIcons` property, enter **AddressIcon**, **AddressIcon**, and **CheckoutIcon**. These strings specify the linkage identifiers of the movie clip symbols that are used as the icons on the accordion headers. You must create these movie clip symbols if you want icons in the headers.
9. Select Control > Test Movie.

To use **ActionScript** to add children to an **Accordion** component:

1. Select File > New and create a Flash document.
2. Drag an Accordion component from the Components panel to the Stage.
3. In the Property inspector, enter the instance name **myAccordion**.
4. Drag a TextInput component to the Stage and delete it.

This adds the component to the library so that you can dynamically instantiate it in step 6.

5. In the Actions panel on Frame 1 of the Timeline, enter the following:

```
myAccordion.createChild("View", "shippingAddress", {label: "Shipping
Address"});
myAccordion.createChild("View", "billingAddress", {label: "Billing
Address"});
myAccordion.createChild("View", "payment", {label: "Payment"});
```

This code calls the `createChild()` method to create its child views.

6. In the Actions panel on Frame 1, below the code you entered in step 5, enter the following code:

```
var o = myAccordion.shippingAddress.createChild("TextInput", "firstName");
o.move(20, 38);
o.setSize(116, 20);
o = myAccordion.shippingAddress.createChild("TextInput", "lastName");
o.move(175, 38);
o.setSize(145, 20);
```

This code adds component instances (two `TextInput` components) to the accordion's children.

Customizing the Accordion component

You can transform an Accordion component horizontally and vertically during authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)).

The `setSize()` method and the Transform tool change only the width of the accordion's headers and the width and height of its content area. The height of the headers and the width and height of the children are not affected. Calling the `setSize()` method is the only way to change the bounding rectangle of an accordion.

If the headers are too small to contain their label text, the labels are clipped. If the content area of an accordion is smaller than a child, the child is clipped.

Using styles with the Accordion component

You can set style properties to change the appearance of the border and background of an Accordion component.

If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in Flash Help.

An Accordion component uses the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. This is the only color style that doesn't inherit its value. Possible values are "haloGreen", "haloBlue", and "haloOrange".
<code>backgroundColor</code>	Both	The background color. The default color is white.
<code>border styles</code>	Both	The Accordion component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See “RectBorder class” in Flash Help. The Accordion component's default border style value is "solid".
<code>headerHeight</code>	Both	The height of the header buttons, in pixels. The default value is 22.
<code>color</code>	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for the header labels. The default value is "_sans".
<code>fontSize</code>	Both	The point size for the font of the header labels. The default value is 10.
<code>fontStyle</code>	Both	The font style for the header labels; either "normal" or "italic". The default value is "normal".
<code>fontWeight</code>	Both	The font weight for the header labels; either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return "none".
<code>textDecoration</code>	Both	The text decoration; either "none" or "underline".

Style	Theme	Description
<code>openDuration</code>	Both	The duration, in milliseconds, of the transition animation.
<code>openEasing</code>	Both	A reference to a tweening function that controls the animation. Defaults to sine in/out. For more information, see “Customizing component animations” in Flash Help.

Using skins with the Accordion component

The Accordion component uses skins to represent the visual states of its header buttons. To skin the buttons and title bar while authoring, modify skin symbols in the Flash UI Components 2/ Themes/MMDefault/Accordion Assets skins states folder in the library of one of the themes FLA files. For more information, see “About skinning components” in Flash Help.

An Accordion component is composed of its border, background, header buttons, and children. The border and background are provided by the `RectBorder` class by default. For information see “`RectBorder` class” in Flash Help. You can skin the headers with the skins listed below.

Property	Description	Default value
<code>falseUpSkin</code>	The up (normal) state of the header above all collapsed children.	<code>accordionHeaderSkin</code>
<code>falseDownSkin</code>	The pressed state of the header above all collapsed children.	<code>accordionHeaderSkin</code>
<code>falseOverSkin</code>	The rolled-over state of the header above all collapsed children.	<code>accordionHeaderSkin</code>
<code>falseDisabled</code>	The disabled state of the header above all collapsed children.	<code>accordionHeaderSkin</code>
<code>trueUpSkin</code>	The up (normal) state of the header above the expanded child.	<code>accordionHeaderSkin</code>
<code>trueDownSkin</code>	The pressed state of the header above the expanded child.	<code>accordionHeaderSkin</code>
<code>trueOverSkin</code>	The rolled-over state of the header above the expanded child.	<code>accordionHeaderSkin</code>
<code>trueDisabledSkin</code>	The disabled state of the header above the expanded child.	<code>accordionHeaderSkin</code>

Using ActionScript to draw the Accordion header

The default headers in both the Halo and Sample themes use the same skin element for all states and draw the actual graphics through ActionScript. The Halo implementation uses an extension of the `RectBorder` class and custom drawing API code to draw the states. The Sample implementation uses the same skin and the same ActionScript class as the Button skin.

To create an ActionScript class to use as the skin and provide different states, the skin can read the `borderStyle` style property of the skin to determine the state. The following table shows the border style that is set for each skin:

Property	Border style
<code>falseUpSkin</code>	<code>falseup</code>
<code>falseDownSkin</code>	<code>falsedown</code>
<code>falseOverSkin</code>	<code>falserollover</code>
<code>falseDisabled</code>	<code>falsedisabled</code>
<code>trueUpSkin</code>	<code>trueup</code>
<code>trueDownSkin</code>	<code>truedown</code>
<code>trueOverSkin</code>	<code>truerollover</code>
<code>trueDisabledSkin</code>	<code>truedisabled</code>

To create an ActionScript customized Accordion header skin:

1. Create a new ActionScript class file.

For this example, name the file **RedGreenBlueHeader.as**.

2. Copy the following ActionScript to the file:

```
import mx.skins.RectBorder;
import mx.core.ext.UIObjectExtensions;

class RedGreenBlueHeader extends RectBorder
{
    static var symbolName:String = "RedGreenBlueHeader";
    static var symbolOwner:Object = RedGreenBlueHeader;

    function size():Void
    {
        var c:Number; // color
        var borderStyle:String = getStyle("borderStyle");

        switch (borderStyle) {
            case "falseup":
            case "falserollover":
            case "falsedisabled":
                c = 0x7777FF;
                break;
            case "falsedown":
                c = 0x77FF77;
                break;
            case "trueup":
            case "truedown":
            case "truerollover":
            case "truedisabled":
                c = 0xFF7777;
                break;
        }
    }
}
```

```

        clear();
       LineStyle(0, 0, 100);
        beginFill(c, 100);
        drawRect(0, 0, __width, __height);
        endFill();
    }

    // required for skins
    static function classConstruct():Boolean
    {
        UIObjectExtensions.Extensions();
        _global.skinRegistry["AccordionHeaderSkin"] = true;
        return true;
    }
    static var classConstructed:Boolean = classConstruct();
    static var UIObjectExtensionsDependency = UIObjectExtensions;
}

```

This class creates a square box based on the border style: a blue box for the false up, rollover, and disabled states; a green box for the normal pressed state; and a red box for the expanded child.

3. Save the file.
4. Create a new FLA file.
5. Save the FLA file in the same folder as the AS file.
6. Create a new symbol by selecting Insert > New Symbol.
7. Set the name to `AccordionHeaderSkin`.
8. If the advanced view is not displayed, click the Advanced button.
9. Select Export for ActionScript.

The identifier will be automatically filled out with `AccordionHeaderSkin`.

10. Set the AS 2.0 class to `RedGreenBlueHeader`.
11. Ensure that Export in First Frame is already selected, and click OK.
12. Drag an Accordion component to the Stage.
13. Set the Accordion properties so that they display several children.

For example, set the `childLabels` to an array of `[One,Two,Three]` and `childNames` to an array of `[one,two,three]`.

14. Select Control > Test Movie.

Using movie clips to customize the Accordion header skin

The above example demonstrates how to use an ActionScript class to customize the Accordion header skin, which is the method used by the skins provided in both the Halo and Sample themes. However, because the example uses simple colored boxes, it is simpler in this case to use different movie clip symbols as header skins.

To create movie clip symbols for Accordion header skins:

1. Create a new FLA file.
2. Create a new symbol by selecting Insert > New Symbol.
3. Set the name to `RedAccordionHeaderSkin`.
4. If the advanced view is not displayed, click the Advanced button.
5. Select Export for ActionScript.

The identifier will be automatically filled out with `RedAccordionHeaderSkin`.
6. Leave the AS 2.0 Class text box blank.
7. Ensure that Export in First Frame is already selected, and click OK.
8. Open the new symbol for editing.
9. Use the drawing tools to create a box with a red fill and black line.
10. Set the border style to hairline.
11. Set the box, including the border, so that it is positioned at (0,0) and has a width and height of 100.

The ActionScript code will size the skin as needed.

12. Repeat steps 2-11 and create green and blue skins, named accordingly.
13. Click the Back button to return to the main Timeline.
14. Drag an Accordion component to the stage.
15. Set the Accordion properties so that they display several children.

For example, set `childLabels` to an array of `[One,Two,Three]` and `childNames` to an array of `[one,two,three]`.

16. Copy the following ActionScript code to the Actions panel with the Accordion instance selected:

```
onClipEvent(initialize) {  
    falseUpSkin = "RedAccordionHeaderSkin";  
    falseDownSkin = "GreenAccordionHeaderSkin";  
    falseOverSkin = "RedAccordionHeaderSkin";  
    falseDisabled = "RedAccordionHeaderSkin";  
    trueUpSkin = "BlueAccordionHeaderSkin";  
    trueDownSkin = "BlueAccordionHeaderSkin";  
    trueOverSkin = "BlueAccordionHeaderSkin";  
    trueDisabledSkin = "BlueAccordionHeaderSkin";  
}
```

17. Select Control > Test Movie.

Accordion class

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > View > Accordion

ActionScript Class Name mx.containers.Accordion

An Accordion component contains children that are displayed one at a time. Each child has a corresponding header button that is created when the child is created. A child must be an instance of UIObject.

A movie clip symbol automatically becomes an instance of the UIObject class when it becomes a child of an accordion. However, to maintain tabbing order in an accordion's children, the children must also be instances of the View class. If you use a movie clip symbol as a child, set its AS 2.0 Class field to mx.core.View so that it inherits from the View class.

Setting a property of the Accordion class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

Each component class has a `version` property that is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.containers.Accordion.version);
```

Note: The code `trace(myAccordionInstance.version);` returns `undefined`.

Method summary for the Accordion class

The following table lists methods of the Accordion class.

Method	Description
Accordion.createChild()	Creates a child for an Accordion instance.
Accordion.createSegment()	Creates a child for an Accordion instance. The parameters for this method are different from those of the <code>createChild()</code> method.
Accordion.destroyChildAt()	Destroys a child at a specified index position.
Accordion.getChildAt()	Gets a reference to a child at a specified index position.

Methods inherited from the UIObject class

The following table lists the methods the Accordion class inherits from the UIObject class. When calling these methods from the Accordion object, use the form `accordionInstance.methodName`.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.

Method	Description
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it will be redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Methods inherited from UIComponent class

The following table lists the methods the Accordion class inherits from the UIComponent class. When calling these methods from the Accordion object, use the form *accordionInstance.methodName*.

Method	Description
UIComponent.getFocus()	Returns a reference to the object that has focus.
UIComponent.setFocus()	Sets focus to the component instance.

Property summary for the Accordion class

The following table lists properties of the Accordion class.

Property	Description
Accordion.numChildren	The number of children of an Accordion instance.
Accordion.selectedChild	A reference to the selected child.
Accordion.selectedIndex	The index position of the selected child.

Properties inherited from the UIObject class

The following table lists the properties the Accordion class inherits from the UIObject class. When accessing these properties, use the form *accordionInstance.propertyName*.

Property	Description
UIObject.bottom	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
UIObject.height	The height of the object, in pixels. Read-only.
UIObject.left	The left edge of the object, in pixels. Read-only.
UIObject.right	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
UIObject.scaleX	A number indicating the scaling factor in the x direction of the object, relative to its parent.

Property	Description
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `Accordion` class inherits from the `UIComponent` class. When accessing these properties, use the form `accordionInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the `Accordion` class

The following table lists an event of the `Accordion` class.

Event	Description
<code>Accordion.change</code>	Broadcast to all registered listeners when the <code>selectedIndex</code> and <code>selectedChild</code> properties of an accordion change because of a user's mouse click or keypress.

Events inherited from the `UIObject` class

The following table lists the events the `Accordion` class inherits from the `UIObject` class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the Accordion class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Accordion.change

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.change = function(eventObject){
    // insert your code here
}
myAccordionInstance.addEventListener("change", listenerObject)
```

Description

Event; broadcast to all registered listeners when the `selectedIndex` and `selectedChild` properties of an accordion change. This event is broadcast only when a user's mouse click or keypress changes the value of `selectedChild` or `selectedIndex`—not when the value is changed with ActionScript. This event is broadcast before the transition animation occurs.

Version 2 components use a dispatcher/listener event model. The Accordion component dispatches a `change` event when one of its buttons is clicked and the event is handled by a function (also called a *handler*) on a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it a reference to the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see [“UIEventDispatcher class” on page 530](#).

The Accordion `change` event also contains two unique event object properties:

- `newValue` Number; the index of the child that is about to be selected.
- `prevValue` Number; the index of the child that was previously selected.

Example

In the following example, a handler called `myAccordionListener` is defined and passed to the `myAccordion.addEventListener()` method as the second parameter. The event object is captured by the `change` handler in the `eventObject` parameter. When the `change` event is broadcast, a trace statement is sent to the Output panel.

```
myAccordionListener = new Object();
myAccordionListener.change = function(){
    trace("Changed to different view");
}
myAccordion.addEventListener("change", myAccordionListener);
```

Accordion.createChild()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myAccordion.createChild(classOrSymbolName, instanceName[, initialProperties])
```

Parameters

classOrSymbolName Either the constructor function for the class of the UIObject to be instantiated, or the linkage name (a reference to the symbol to be instantiated). The class must be UIObject or a subclass of UIObject, but most often it is View object or a subclass of View.

instanceName The instance name of the new instance.

initialProperties An optional parameter that specifies initial properties for the new instance. You can use the following properties:

- `label` A string that specifies the text label that the new child instance uses on its header.
- `icon` A string that specifies the linkage identifier of the library symbol that the child uses for the icon on its header.

Returns

A reference to an instance of the UIObject that is the newly created child.

Description

Method (inherited from View); creates a child for the accordion. The newly created child is added to the end of the list of children owned by the accordion. Use this method to place views inside the accordion. The created child is an instance of the class or movie clip symbol specified in the `classOrSymbolName` parameter. You can use the `label` and `icon` properties to specify a text label and an icon for the associated accordion header for each child in the `initialProperties` parameter.

When each child is created, it is assigned an index number in the order of creation and the `numChildren` property is increased by 1.

Example

The following code creates an instance of the `PaymentForm` movie clip symbol named `payment` as the last child of `myAccordion`:

```
var child = myAccordion.createChild("PaymentForm", "payment", {label:
    "Payment", Icon: "payIcon"});
child.cardType.text = "Visa";
child.cardNumber.text = "1234567887654321";
```

The following code creates a child that is an instance of the `View` class:

```
var child = myAccordion.createChild(mx.core.View, "payment", {label:
    "Payment", Icon: "payIcon"});
child.cardType.text = "Visa";
child.cardNumber.text = "1234567887654321";
```

The following code also creates a child that is an instance of the `View` class, but it uses `import` to reference the constructor for the `View` class:

```
import mx.core.View
var child = myAccordion.createChild(View, "payment", {label: "Payment", Icon:
    "payIcon"});
child.cardType.text = "Visa";
child.cardNumber.text = "1234567887654321";
```

Accordion.createSegment()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myAccordion.createSegment(classOrSymbolName, instanceName[, label[, icon]])
```

Parameters

classOrSymbolName Either a reference to the constructor function for the class of the `UIObject` to be instantiated, or the linkage name of the symbol to be instantiated. The class must be `UIObject` or a subclass of `UIObject`, but most often it is `View` or a subclass of `View`.

instanceName The instance name of the new instance.

label A string that specifies the text label that the new child instance uses on its header. This parameter is optional.

icon A string reference to the linkage identifier of the library symbol that the child uses for the icon on its header. This parameter is optional.

Returns

A reference to the newly created `UIObject` instance.

Description

Method; creates a child for the accordion. The newly created child is added to the end of the list of children owned by the accordion. Use this method to place views inside the accordion. The created child is an instance of the class or movie clip symbol specified in the *className* parameter. You can use the *label* and *icon* parameters to specify a text label and an icon for the associated accordion header for each child.

The `createSegment()` method differs from the `addChild()` method in that *label* and *icon* are passed directly as parameters, not as properties of an *initialProperties* parameter.

When each child is created, it is assigned an index number in the order of creation, and the `numChildren` property is increased by 1.

Example

The following example creates an instance of the `PaymentForm` movie clip symbol named `payment` as the last child of `myAccordion`:

```
var child = myAccordion.createSegment("PaymentForm", "payment", "Payment",
    "payIcon");
child.cardType.text = "Visa";
child.cardNumber.text = "1234567887654321";
```

The following code creates a child that is an instance of the `View` class:

```
var child = myAccordion.createSegment(mx.core.View, "payment", {label:
    "Payment", Icon: "payIcon"});
child.cardType.text = "Visa";
child.cardNumber.text = "1234567887654321";
```

The following code also creates a child that is an instance of the `View` class, but it uses `import` to reference the constructor for the `View` class:

```
import mx.core.View
var child = myAccordion.createSegment(View, "payment", {label: "Payment",
    Icon: "payIcon"});
child.cardType.text = "Visa";
child.cardNumber.text = "1234567887654321";
```

Accordion.destroyChildAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myAccordion.destroyChildAt(index)
```

Parameters

index The index number of the accordion child to destroy. Each child of an accordion is assigned a zero-based index number in the order in which it was created.

Returns

Nothing.

Description

Method (inherited from `View`); destroys one of the accordion's children. The child to be destroyed is specified by its index, which is passed to the method in the *index* parameter. Calling this method destroys the corresponding header as well.

If the destroyed child is selected, a new selected child is chosen. If there is a next child, it is selected. If there is no next child, the previous child is selected. If there is no previous child, the selection is undefined.

Note: Calling `destroyChildAt()` decreases the `numChildren` property by 1.

Example

The following code destroys the last child of `myAccordion`:

```
myAccordion.destroyChildAt(myAccordion.numChildren - 1);
```

See also

[Accordion.createChild\(\)](#)

Accordion.getChildAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myAccordion.getChildAt(index)
```

Parameters

index The index number of an accordion child. Each child of an accordion is assigned a zero-based index in the order in which it was created.

Returns

A reference to the instance of the `UIObject` at the specified index.

Description

Method; returns a reference to the child at the specified index. Each accordion child is given an index number for its position. This index number is zero-based, so the first child is 0, the second child is 1, and so on.

Example

The following code gets a reference to the last child of `myAccordion`:

```
var lastChild:UIObject = myAccordion.getChildAt(myAccordion.numChildren - 1);
```

Accordion.numChildren

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myAccordion.numChildren
```

Description

Property (inherited from View); indicates the number of children (of type UIObject) in an Accordion instance. Headers are not counted as children.

Each accordion child is given an index number for its position. This index number is zero-based, so the first child is 0, the second child is 1, and so on. The code `myAccordion.numChild - 1` always refers to the last child added to an accordion. For example, if there were seven children in an accordion, the last child would have the index 6. The `numChildren` property is not zero-based, so the value of `myAccordion.numChildren` would be 7. The result of `7 - 1` is 6, which is the index number of the last child.

Example

The following example selects the last child:

```
myAccordion.selectedIndex = myAccordion.numChildren - 1;
```

Accordion.selectedChild

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myAccordion.selectedChild
```

Description

Property; the selected child (of type UIObject) if one or more children exist; undefined if no children exist.

If the accordion has children, the code `myAccordion.selectedChild` is equivalent to the code `myAccordion.getChildAt(myAccordion.selectedIndex)`.

Setting this property to a child causes the accordion to begin the transition animation to display the specified child.

Changing the value of `selectedChild` also changes the value of `selectedIndex`.

The default value is `myAccordion.getChildAt(0)` if the accordion has children. If the accordion doesn't have children, the default value is `undefined`.

Example

The following example retrieves the label of the selected child view:

```
var selectedLabel = myAccordion.selectedChild.label;
```

The following example sets the payment form to be the selected child view:

```
myAccordion.selectedChild = myAccordion.payment;
```

See also

[Accordion.selectedIndex](#)

Accordion.selectedIndex

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myAccordion.selectedIndex
```

Description

Property; the zero-based index of the selected child in an accordion with one or more children. For an accordion with no child views, the only valid value is `undefined`.

Each accordion child is given an index number for its position. This index number is zero-based, so the first child is 0, the second child is 1, and so on. The valid values of `selectedIndex` are 0, 1, 2, ..., $n - 1$, where n is the number of children.

Setting this property to a child causes the accordion to begin the transition animation to display the specified child.

Changing the value of `selectedIndex` also changes the value of `selectedChild`.

Example

The following example remembers the index of the selected child:

```
var oldSelectedIndex = myAccordion.selectedIndex;
```

The following example selects the last child:

```
myAccordion.selectedIndex = myAccordion.numChildren - 1;
```

See also

[Accordion.numChildren](#), [Accordion.selectedChild](#)

AccordionTab component

The AccordionTab component lets you create multiple tabs, each with an associated area, that allow the display of large amounts of information in a relatively small space. Each tab contains a label and icon, if desired, and an associated area for content. Users click a tab to switch among tabs. The component uses animation to switch from one tab to another, in an accordion-like fashion, thus giving this component its name.

The following image shows an example of how the AccordionTab component can be used to create three tabs: At a Glance, Find a Movie, and Favorites. Note that the Favorites tab contains the Favorites icon.



Using the AccordionTab component

To use the AccordionTab component, specify a label for each tab and a data provider for the content area. The component sends callback functions that you can use to hide or show different data on the screen, depending on the selected index of the component. You cannot put a movie clip into the content area, so instead, you must listen for the tab-switching animation to start and finish and swap in your movie clip. The component has two events that you can listen for: `onAnimationStart` and `onAnimationDone`. For more information, see [“AccordionTab examples” on page 37](#).

The `MAccordionTab.setDataProvider()` method lets you set a data provider for the accordion tabs. A data provider can be an array or an instance of the `DataProvider` class. For more information about the `DataProvider` class, see *Developing Macromedia Central Applications*.

When it is appropriate to your application design, Macromedia recommends that your Central application contain a tab, such as a Favorites tab, to display information that the application has cached for offline viewing. Users add information they want to save to the Favorites tab by clicking a button in your application, such as an Add to Favorites button. Users can then effectively use your application offline. See Macromedia applications in the Macromedia Central SDK for an example of this implementation.

The Favorites tab should contain the Favorites symbol; the Favorites symbol is also on the Add to Favorites button and Remove from Favorites buttons, which you can use to add data to and remove data from the Favorites tab. To set the Favorites icon in a tab in your `AccordionTab` instance, specify the `icon` property for the favorites element in the data provider (`dp`) for the `AccordionTab` instance. The following example uses a data provider named `dp`:

```
dp.addItem({label:"Favorites", data:"FavoritesSymbol",
            icon:"MAccordionTabFavoritesFlag" });
```

For information about the Add to Favorites or Remove from Favorites buttons, see [“Using the IconButton component” on page 242](#).

AccordionTab parameters

You can set the following parameters for the each instance of the `AccordionTab` component:

Labels sets the text for each tab in an array.

Base Color is the base color set for this accordion tab instance. The component automatically selects this base color or appropriate hues of the base color for different elements of the component, such as the color for the tab text in the selected state.

Change Handler is the name of the function that you call when the user selects a tab. You must define this function in the same Timeline as the instance of the `AccordionTab`. This parameter is optional and must be specified only if you want an action to occur when the user selects a tab. For more information, see [“Writing event listeners for components” on page 11](#).

About AccordionTab states

Each tab of the `AccordionTab` component has selected and unselected states and animates between them. In the unselected state, a tab is the color set using the `setBaseColor` method. When a user selects the tab, the tab color changes to white and the tab text color takes on the color set using the `setBaseColor` method. The tab animates so it appears that the content area opens while the other tabs slide over. This component has no border color for the selected state.

AccordionTab examples

The following code provides examples of how to set up an `AccordionTab` component in your application. Both examples create an `AccordionTab` instance for an address book application with three tabs: Add Contacts, Find A Contact, and Favorites. To have a movie clip show in the `AccordionTab` component’s content area, the code listens for `onAnimationStart` and `onAnimationDone`, passes a reference to itself as the only parameter, and swaps out the movie clip that should appear on top of the instance of the `AccordionTab` component, using the `attachMovie()` method.

AccordionTab example one

The following code example is a simple implementation of setting up an `AccordionTab` component. When the user clicks a tab, the tab animates to a selected state, and the previously selected tab animates to an unselected state. When the animation starts, the previous contents of the tab are cleared. When the animation ends, the contents of the new tab are revealed. The following code demonstrates how to listen and respond to the `onAnimationStart` and `onAnimationDone` events:

```
//-----  
// Set up the Tab menu with data  
this.configureMenu = function(menu)  
{  
    this.nextLevel = 100;  
    var dp = new mx.central.data.DataProviderClass();  
    dp.addItem({label:"Add Contacts", data: "AddressBookSymbol"});  
    dp.addItem({label:"Find A Contact", data: "FindAContactSymbol"});  
    dp.addItem({label:"Favorites", data:"FavoritesSymbol",  
        icon:"MAccordionTabFavoritesFlag"});  
    menu.setDataProvider(dp);  
    menu.addListener(this);  
    menu.setSelectedIndex(0);  
}  
//-----  
// Triggered from the Tabs whenever the change is begun  
this.onAnimationStart = function(menu)  
{  
    trace(">> Tab is beginning to change - hide the current section now");  
    this.currentView.deActivate(); // Custom Method in the current view clip...  
}  
//-----  
// Triggered from the Tabs whenever the change is complete -  
this.onAnimationDone = function(menu)  
{  
    var sel = menu.getSelectedItem().data;  
    trace(">> Tab is finished changing - show the new section [" + sel + "]");  
    if(this.currentView instanceof MovieClip) {  
        this.currentView.removeMovieClip();  
    }  
    this[sel] = this.attachMovie(sel, sel+"_mc", this.nextLevel++,  
        {controller:this});  
    this.currentView = this[sel];  
}  
//-----  
// The instance name of the component is tabs_mc.  
this.configureMenu(this.tabs_mc);
```

AccordionTab example two

The following example shows a slightly more complex, but thorough, method of setting up an `AccordionTab` component:

```
//-----  
// Set up the Tab menu with data
```

```

this.configureMenu = function(menu)
{
    this.nextLevel = 100;
    var dp = new mx.central.data.DataProviderClass();
    dp.addItem({label:"At-A-Glance", data: "AtAGlanceSymbol"});
    dp.addItem({label:"Find A Restaurant", data: "FindARestaurantSymbol"});
    dp.addItem({label:"Favorites", data:"FavoritesSymbol",
    icon:"MAccordionTabFavoritesFlag"});
    menu.setDataProvider(dp);
    menu.addListener(this);
    menu.setSelectedIndex(0);
}
//-----
// Triggered from the Tabs whenever the change is begun -
this.onAnimationStart = function(menu)
{
    trace(">> Tab is beginning to change - hide the current section now");
    this.currentView.deActivate(); // Custom Method in the current view clip...
}
//-----
// Triggered from the Tabs whenever the change is complete -
this.onAnimationDone = function(menu)
{
    var sel = menu.getSelectedItem().data;
    trace(">> Tab is finished changing - show the new section [" + sel + "]");
    this.setCurrentView(sel);
}
//-----
// Attach the view if it doesn't yet exist
// call it's custom .activate function if it does...
// This postpones intensive instantiation until absolutely necessary,
// and the use of activate/deActivate prevents additional/unnecessary
// instantiation...

this.setCurrentView = function(str)
{
    // Set the linkage to the library path of your views.
    var linkage = "com.yourDomain.app.views." + str;

    // If the currentView has been previously set,
    // This means that the user is currently viewing
    // something... so we should hide it first...

    if (this.currentView instanceof MovieClip) {
        this.currentView._visible = 0;
    }
    // Determine whether or not the currently selected
    // content movie is already attached-
    // if so - simply show it...
    // if not - attach it...
    // NOTE: Rather than setting a content movie's
    // _visible property - you should call some
    // custom methods - like activate/deActivate...
    // That would not only hide but also clean up
    // any listeners or other cpu-intensive assets...

```

```

// and may also accept params like current height / width data...

if (this[str] == undefined) {
var lvl = this.nextLevel++;
this[str] = this.attachMovie(linkage, str + "_mc", lvl, {controller:this});
} else {
    this[str]._visible = 1;
}

this.currentView = this[str];
this.persistCurrentView(str); // Some Method that will store the current
view for next session...
}
//-----
// Should retrieve the contentWidth prop - which should be managed
// with the onResize event...
this.getContentWidth = function()
{
    return this.contentWidth;
}
//-----
// Should retrieve the contentHeight prop - which should be managed
// with the onResize event...
this.getContentHeight = function()
{
    return this.contentHeight;
}
//-----
// The instance name of the component is tabs_mc.
this.configureMenu(this.tabs_mc);

```

Method summary for the MAccordionTab component

The following table summarizes the methods for the MAccordionTab component:

Method	Description
MAccordionTab.addItem()	Appends an item to the data provider.
MAccordionTab.addItemAt()	Adds an item at the specified index in the data provider.
MAccordionTab.getBaseColor()	Returns the base color of the component in decimal format.
MAccordionTab.getContentBounds()	Returns an object containing four points— <code>xMin</code> , <code>xMax</code> , <code>yMin</code> , and <code>yMax</code> —which represent coordinates in the component's <code>_parent</code> coordinate system. These coordinates are the corners of the white content area of the component.
MAccordionTab.getDataProvider()	Returns the data provider for the component.
MAccordionTab.getItemAt()	Returns an object containing the label and data properties for the tab at the specified index.
MAccordionTab.getLength()	Gets the number of tabs.
MAccordionTab.getSelectedIndex()	Returns the index of the selected tab.

Method	Description
<code>MAccordionTab.getSelectedItem()</code>	Returns an object containing the label and data properties for the selected tab.
<code>MAccordionTab.getValue()</code>	Returns an object representing the currently selected tab. The object contains label and data properties with an optional icon property.
<code>MAccordionTab.removeAll()</code>	Removes all the tabs.
<code>MAccordionTab.removeItemAt()</code>	Removes the specified item in the data provider and in the content area of the tabs.
<code>MAccordionTab.replaceItemAt()</code>	Replaces the tab at the specified index with the new data.
<code>MAccordionTab.setBaseColor()</code>	Assigns the base color for the component.
<code>MAccordionTab.setChangeHandler()</code>	Assigns a change handler that gets called when the user selects a tab with the mouse.
<code>MAccordionTab.setDataProvider()</code>	Sets the data provider for the tabs. The component looks for three specific properties in the data provider elements: a label, data, and an optional icon property that points to an item in the library.
<code>MAccordionTab.setSelectedIndex()</code>	Sets the selected tab for the component.
<code>MAccordionTab.setSize()</code>	Sets the width and height of the component.

MAccordionTab.addItem()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.addItem(label, data)
```

Parameters

label A string for the label for the tab.

data The data provider for the content area. This can be an array or an object of the `DataProvider` class.

Returns

Nothing.

Description

Method; appends a tab to the end of the tab group with a content area. You specify a label and data for the tab. You can use this method or `MAccordionTab.addItemAt()` with the `MAccordionTab.setDataProvider()` method to populate all the tabs.

If you add only one item to the component instance, it doesn't appear as a colored tab and appears as a title in the content area. You need a minimum of two items to have differentiated tabs visible to the user.

Example

The following example creates three tabs, labeled Section 1, Section 2, and Section 3, with content from an object of the `DataProvider` class. The code then gets a reference to the third item and traces its label.

```
myAccordionTab.addItem("Section 1","section1");
myAccordionTab.addItem("Section 2","section2");
myAccordionTab.addItem("Section 3","section3");
var item = myAccordionTab.getValue();
trace(item.label) // traces "Section 3"
```

MAccordionTab.addItemAt()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.addItemAt(index, label, data)
```

Parameters

index An integer indicating the index where the item is added.

label A string for the tab's label.

data The data provider for the content area. This can be an array or an object of the `DataProvider` class.

Returns

Nothing.

Description

Method; adds a tab at the specified index in the tab group. For each tab in the `AccordionTab` component, use either this method or `addItem()`, or use the `setDataProvider` method to populate all the tabs.

Example

The following example creates tabs labeled Section 2 and Section 3, and then adds another tab at the beginning of the tab group:

```
myAccordionTab.addItem("Section 2","section2data");
myAccordionTab.addItem("Section 3","section3data");
myAccordionTab.addItemAt(0, "Section 1","section3");
```

MAccordionTab.getBaseColor()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.getBaseColor()
```

Parameters

None.

Returns

Returns the base color of the component in decimal format.

Description

Method; gets the base color of the component in decimal format.

Example

The following example gets the base color and traces it:

```
var color = myAccordionTab.getBaseColor();  
trace("Color is: "+color);
```

MAccordionTab.getContentBounds()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.getContentBounds()
```

Parameters

None.

Returns

An object containing the coordinates of the content area.

Description

Method; gets an object containing four points: xMin, xMax, yMin, and yMax. These points represent coordinates in the component's parent coordinate system and represent the boundary corners of the content area. This method is useful to get the size of the component's content area.

Example

The following example traces the boundary corners of the `myAccordionTab` instance:

```
trace("bounds");
var bounds = myAccordionTab.getContentBounds();
for (var prop in bounds) {
  trace("bounds[" + prop + "] = " + bounds[prop]);
}
```

MAccordionTab.getDataProvider()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.getDataProvider()
```

Parameters

None.

Returns

The data provider that is used to populate the tabs—either an array or a `DataProvider` object.

Description

Method; gets the array or `DataProvider` object being used to populate the accordion tab. If a `DataProvider` object is returned, it should have the following fields: `label`, `data`, and an optional `icon` field that points to an icon in the library.

Example

The following example gets the data provider for the `myTabs` instance:

```
trace(myTabs.getDataProvider());
```

MAccordionTab.getItemAt()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.getItemAt(index)
```

Parameters

index An integer representing the index of the item.

Returns

Gets an object containing the data and label properties for the tab at the specified index.

Description

Method; gets an object containing the data and label properties for the tab at the specified index.

Example

The following code adds four tabs to the component and then gets the item at index 2 and traces its label:

```
var tabs = ["tab 1","tab 2","tab 3","tab 4"];
myAccordionTab.setDataProvider(tabs);
var item = myAccordionTab.getItemAt(2);
// traces "Item Label: Section 3"
trace("Item Label: "+item.label);
```

MAccordionTab.getLength()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.getLength()
```

Parameters

None.

Returns

Gets the length of the tab.

Description

Method; gets the length of the tab.

Example

The following code gets the length of the tab:

```
myAccordionTab.getLength();
```

MAccordionTab.getSelectedIndex()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

myComponent.getSelectedIndex()

Parameters

None.

Returns

An integer indicating the index of the currently selected tab of the component.

Description

Method; gets an integer indicating the index of the currently selected tab.

Example

The following example adds four tabs to the component, sets the selected index 2, and then retrieves information from the currently selected section:

```
var sections = ["Section 1","Section 2","Section 3","Section 4"];
myAccordionTab.setDataProvider(sections); myAccordionTab.setSelectedIndex(2);
var index = myAccordionTab.getSelectedIndex();
// traces "Index: 2"
trace("Index: "+index);
```

MAccordionTab.getSelectedItem()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

myComponent.getSelectedItem()

Parameters

None.

Returns

Gets an object representing the currently selected item. The object has two properties, label and data, which contain information about the specified tab item.

Description

Method; gets an object representing the currently selected item.

Example

The following example adds four tabs to the AccordionTab component, sets the selected item to the third item, and then traces its label:

```
var sections = ["Tab 1","Tab 2","Tab 3","Tab 4"];
myAccordionTab.setDataProvider(sections);
```

```
myAccordionTab.setSelectedIndex(2);

var item = myAccordionTab.getSelectedItemAt();
// traces "Selected Item: Tab 3"
trace("Selected Item: "+item.label);
```

MAccordionTab.getValue()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.getValue()
```

Parameters

None.

Returns

Gets an object representing the currently selected tab. The object has label and data properties, with an optional icon property.

Description

Method; gets an object representing the currently selected tab. The object has label and data properties, with an optional icon property, that provide information about the tab.

Example

The following example adds four tabs to the component, sets the selected index to 2, and then retrieves the label for the currently selected tab:

```
var sections = ["Tab 1","Tab 2","Tab 3","Tab 4"];
myAccordionTab.setDataProvider(sections);
myAccordionTab.setSelectedIndex(2);
var item = myAccordionTab.getValue();
// traces "Selected Item: Tab 3"
trace("Selected Item: "+item.label);
```

MAccordionTab.removeAll()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.removeAll()
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all the tabs. Any data associated with the tabs is lost.

Example

The following example removes all the tabs for the `myAccordionTab` instance:

```
myAccordionTab.removeAll();
```

MAccordionTab.removeItemAt()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.removeItemAt(index)
```

Parameters

index An integer representing the index of the tab to be removed.

Returns

Nothing.

Description

Method; removes the tab at the specified index. Any data associated with the tab is also removed from the data provider. Any items above the remove index have their index decremented.

Example

The following code adds four tabs to the component, removes the tab at index 2, and then traces the label of the new tab at index 2:

```
var tabs = ["tab 1","tab 2","tab 3","tab 4"];
myAccordionTab.setDataProvider(tabs);
myAccordionTab.removeItemAt(2);
var item = myAccordionTab.getItemAt(2);
// traces "Item Label: Section 4"
trace("Item Label: "+item.label);
```

MAccordionTab.replaceItemAt()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.replaceItemAt(index, data, label)
```

Parameters

index An integer indicating the index of the item.

data The new data for the item.

label The new label for the item.

Returns

Nothing.

Description

Method; replaces the tab at the specified index with new data and label.

Example

The following example replaces the label of the third tab with the text “Favorites”:

```
myAccordionTab.replaceItemAt(2, "dp", "Favorites");
```

MAccordionTab.setBaseColor()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.setBaseColor(color)
```

Parameters

color The color, in hexadecimal format.

Returns

Nothing.

Description

Method; sets the base color for the tab component. The component automatically uses the base color or determines shades of the selected base color for other elements in the component.

The default color for the text labels of the tabs is white in the unselected state and the base color in the selected state. Be sure to set a base color that is dark enough to show white text. For example, do not use a pale color or the white text labels might be unreadable.

This method will not work if you call the method on the instance of the `AccordionTab` component before the first frame has been drawn. To set the base color of your instance of the `AccordionTab` component:

- If the component is placed on the Stage in authoring, set the base color of the component in the Properties pane.
- If the component is added dynamically with ActionScript, pass in the Base Color parameter as an initial parameter.
- Set the base color using the Base Color parameter in the authoring tool instead of in ActionScript.

Example

The following example sets the base color to sky blue (#3399FF):

```
myAccordionTab.setBaseColor(#3399FF);
```

MAccordionTab.setChangeHandler()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.setChangeHandler(handler[, scope])
```

Parameters

handler A string specifying the function that should be called when the selected section of the component changes.

scope An object in which the specified function is called.

Returns

Nothing.

Description

Method; specifies the function to be called when the user clicks the a tab. If *scope* is specified, the function is called within the scope of that object. Otherwise, `_parent` is used.

Example

The following code adds four sections to the component and then defines a change handler that is called when the selected section of the component changes:

```
var tabs = ["Tab 1", "Tab 2", "Tab 3", "Tab 4"];
```

```

myAccordionTab.setDataProvider(tabs);
myAccordionTab.setChangeHandler("onTabSelect");
// traces "Selected Index: 2";
function onTabSelect(tab){
    trace("Selected Index: "+tab.getSelectedIndex());
}

myAccordionTab.setSelectedIndex(2);

```

MAccordionTab.setDataProvider()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.setDataProvider(dataProvider)
```

Parameters

dataProvider A data provider that is used to create and populate tabs. It can be an array or an object of the `DataProvider` class.

Returns

Nothing.

Description

Method; sets up the data provider for the tabs. The `MAccordionTab` component looks for three specific properties in the data provider elements: label, data, and an optional icon property that points to an item in the library.

To have the Favorites bookmark show up on a tab in the `AccordionTab` component, specify the `icon` property for the Favorites element, as shown in the following example:

```
dp.addItem({label:"Favorites", data:"FavoritesSymbol",
    icon:"MAccordionTabFavoritesFlag"} );
```

The `DataProvider` class included with the Macromedia Central SDK contains new methods. For more information about the `DataProvider` class and its methods, see “`Central.DataProviderClass` object” in *Developing Central Applications*.

Example

The following code creates a data provider instance that is an array and uses it to configure and populate the component:

```

var tabs = ["tab 1","tab 2","tab 3","tab 4"];
myAccordionTab.setDataProvider(tabs);

var item = myAccordionTab.getItemAt(2);
// traces "Item Label: Section 3"
trace("Item Label: "+item.label);

```

MAccordionTab.setSelectedIndex()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.setSelectedIndex(index)
```

Parameters

index The index of the tab to set as selected.

Returns

Nothing.

Description

Method; sets tab specified in *index* as selected.

Example

The following example adds four tabs to the AccordionTab component, sets the selected index to the third item, and then traces its label:

```
var sections = ["Tab 1","Tab 2","Tab 3","Tab 4"];
myAccordionTab.setDataProvider(sections);
myAccordionTab.setSelectedIndex(2);

var index = myAccordionTab.getSelectedIndex();
//traces "Selected index: 2"
trace("Selected index: "+index);
```

MAccordionTab.setSize()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.setSize(width, height)
```

Parameters

width The new width in pixels.

height The new height in pixels.

Returns

Nothing.

Description

Method; lets you specify the width and height (in pixels) of the component's content area.

Example

The following example sets the size to 200 pixels wide and 400 pixels high:

```
myAccordionTab.setSize(200,400);
```

Alert component

The Alert component lets you display a window that presents the user with a message and response buttons. The window has a title bar that you can fill with text, a message that you can customize, and buttons whose labels you can change. An Alert window can have any combination of Yes, No, OK, and Cancel buttons, and you can change the button labels by using the [Alert.yesLabel](#), [Alert.click](#), [Alert.okLabel](#), and [Alert.cancelLabel](#) properties. You cannot change the order of the buttons in an Alert window; the button order is always OK, Yes, No, Cancel. An Alert window closes when a user clicks any of its buttons.

To display an Alert window, call the [Alert.show\(\)](#) method. In order to call the method successfully, the Alert component must be in the library. By dragging the Alert component from the Components panel to the Stage and then deleting the component, you add the component to the library without making it visible in the document.

The live preview for the Alert component is an empty window.

When you add an Alert component to an application, you can use the Accessibility panel to make the component's text and buttons accessible to screen readers. First, add the following line of code to enable accessibility:

```
mx.accessibility.AlertAccImpl.enableAccessibility();
```

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

Using the Alert component

You can use an Alert component whenever you want to announce something to a user. For example, you could display an alert when a user doesn't fill out a form properly, when a stock hits a certain price, or when a user quits an application without saving the session.

Alert parameters

The Alert component has no authoring parameters. You must call the ActionScript [Alert.show\(\)](#) method to display an Alert window. You can use other ActionScript properties to modify the Alert window in an application. For more information, see [“Alert class” on page 58](#).

Creating an application with the Alert component

The following procedure explains how to add an Alert component to an application while authoring. In this example, the Alert component appears when a stock hits a certain price.

To create an application with the Alert component:

1. Double-click the Alert component in the Components panel to add it to the Stage.
2. Press Backspace (Windows) or Delete (Macintosh) to delete the component from the Stage.

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

3. In the Actions panel, enter the following code on Frame 1 of the Timeline to define an event handler for the `click` event:

```
import mx.controls.Alert;
myClickHandler = function (evt){
    if (evt.detail == Alert.OK){
        trace("start stock app");
        // startStockApplication();
    }
}
Alert.show("Launch Stock Application?", "Stock Price Alert", Alert.OK |
Alert.CANCEL, this, myClickHandler, "stockIcon", Alert.OK);
```

This code creates an Alert window with OK and Cancel buttons. When the user clicks either button, Flash calls the `myClickHandler` function. But when the user clicks the OK button, Flash calls the `startStockApplication()` function.

Note: The `Alert.show()` method includes an optional parameter that displays an icon in the Alert window (in this example, an icon with the linkage identifier "stockIcon"). To include this icon in your test example, create a symbol named `stockIcon` and set it to Export for ActionScript in the Linkage Properties dialog box or the Create New Symbol dialog box.

4. Select Control > Test Movie.

Customizing the Alert component

The Alert component positions itself in the center of the component that was passed as its *parent* parameter. The parent must be a `UIComponent` object. If it is a movie clip, you can register the clip as `mx.core.View` so that it inherits from `UIComponent`.

The Alert window automatically stretches horizontally to fit the message text or any buttons that are displayed. If you want to display large amounts of text, include line breaks in the text.

The Alert component does not respond to the `setSize()` method.

Using styles with the Alert component

You can set style properties to change the appearance of an Alert component. If the name of a style property ends in "Color", it is a color style property and behaves differently than noncolor style properties.

An Alert component supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>backgroundColor</code>	Both	The background color. The default color is white for the Halo theme and 0xEFEBEF (light gray) for the Sample theme.
<i>border styles</i>	Both	The Alert component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See "RectBorder class" in Flash Help. The Alert component has a component-specific <code>borderStyle</code> setting of "alert" with the Halo theme and "outset" with the Sample theme.
<code>color</code>	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is "_sans".
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either "normal" or "italic". The default value is "normal".
<code>fontWeight</code>	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return "none".
<code>textAlign</code>	Both	The text alignment: either "left", "right", or "center". The default value is "left".
<code>textDecoration</code>	Both	The text decoration: either "none" or "underline". The default value is "none".
<code>textIndent</code>	Both	A number indicating the text indent. The default value is 0.

The Alert component includes three different categories of text. Setting the text properties for the Alert component itself provides default values for all three categories, as shown here:

```
import mx.controls.Alert;
_global.styles.Alert.setStyle("color", 0x000099);
Alert.show("This is a test alert", "Title");
```

To set the text styles for one category individually, the Alert component provides static properties that are references to a CSSStyleDeclaration instance.

Static property	Text affected
<code>buttonStyleDeclaration</code>	Button
<code>messageStyleDeclaration</code>	Message
<code>titleStyleDeclaration</code>	Title

The following example demonstrates how to set the title of an Alert component to be italicized:

```
import mx.controls.Alert;
import mx.styles.CSSStyleDeclaration;

var titleStyles = new CSSStyleDeclaration();
titleStyles.setStyle("fontWeight", "bold");
titleStyles.setStyle("fontStyle", "italic");

Alert.titleStyleDeclaration = titleStyles;

Alert.show("Name is a required field", "Validation Error");
```

The default title style declarations set `fontWeight` to "bold". When you override the `titleStyleDeclaration` property, this default is also overridden, so you must explicitly set `fontWeight` to "bold" if that setting is desired.

Note: Text styles set on an Alert component provide default text styles to its components through style inheritance.

Using skins with the Alert component

The Alert component extends the Window component and uses its title background skin for the title background, a `RectBorder` class instance for its border, and Button skins for the visual states of its buttons. To skin the buttons and title bar while authoring, modify the Flash UI Components 2/Themes/MMDefault/Window Assets/Elements/TitleBackground and Flash UI Components 2/Themes/MMDefault/Button Assets/ButtonSkin symbols. For more information, see "About skinning components" in Flash Help. The border and background are provided by the `RectBorder` class by default. For information on skinning the `RectBorder` class, see "RectBorder class" in Flash Help.

An Alert component uses the following skin properties to dynamically skin the buttons and title bar:

Property	Description	Default value
<code>buttonUp</code>	The up state of the buttons.	<code>ButtonSkin</code>
<code>buttonUpEmphasized</code>	The up state of the default button.	<code>ButtonSkin</code>
<code>buttonDown</code>	The pressed state of the buttons.	<code>ButtonSkin</code>
<code>buttonDownEmphasized</code>	The pressed state of the default button.	<code>ButtonSkin</code>
<code>buttonOver</code>	The rolled-over state of the buttons.	<code>ButtonSkin</code>

Property	Description	Default value
<code>buttonOverEmphasized</code>	The rolled-over state of the default button.	<code>ButtonSkin</code>
<code>titleBackground</code>	The window title bar.	<code>TitleBackground</code>

To set the title of an Alert component to a custom movie clip symbol:

1. Create a new FLA file.
2. Create a new symbol by selecting Insert > New Symbol.
3. Set the name to `TitleBackground`.
4. If the advanced view is not displayed, click the Advanced button.
5. Select Export for ActionScript.
6. The identifier will be automatically filled out with `TitleBackground`.
7. Set the AS 2.0 class to `mx.skins.SkinElement`.

`SkinElement` is a simple class that can be used for all skin elements that don't provide their own ActionScript implementation. It provides movement and sizing functionality required by the version 2 component framework.
8. Ensure that Export in First Frame is already selected.
9. Click OK.
10. Open the new symbol for editing.
11. Use the drawing tools to create a box with a red fill and black line.
12. Set the border style to hairline.
13. Set the box, including the border, so that is positioned at (0,0) and has a width of 100 and height of 22.

The Alert component sets the proper width of the skin as needed, but it uses the existing height as the height of the title.
14. Click the Back button to return to the main Timeline.
15. Drag an Alert component to the Stage and delete it.

This will add the Alert component to the library and available at runtime.
16. Add ActionScript code to the main Timeline to create a sample Alert instance.

```
import mx.controls.Alert;
Alert.show("This is a skinned Alert component","Title");
```
17. Select Control > Test Movie.

Alert class

Inheritance [MovieClip](#) > [UIObject class](#) > [UIComponent class](#) > [View](#) > [ScrollView](#) > [Window component](#) > [Alert](#)

ActionScript Class Name `mx.controls.Alert`

To use the Alert component, you drag an Alert component to the Stage and delete it so that the component is in the document library but not visible in the application. Then you call `Alert.show()` to display an Alert window. You can pass parameters to `Alert.show()` that add a message, a title bar, and buttons to the Alert window.

Because ActionScript is asynchronous, the Alert component is not blocking, which means that the lines of ActionScript code that follow the call to `Alert.show()` run immediately. You must add listeners to handle the `click` events that are broadcast when a user clicks a button and then continue your code after the event is broadcast.

Note: In operating environments that are blocking (for example, Microsoft Windows), a call to `Alert.show()` does not return until the user has taken an action, such as clicking a button.

To understand more about the Alert class, see [“Window component” on page 505](#) and [“PopUpManager class”](#) in Flash Help.

Method summary for the Alert class

The following table lists the method of the Alert class.

Method	Description
Alert.show()	Creates an Alert window with optional parameters.

Methods inherited from the UIObject class

The following table lists the methods the Alert class inherits from the UIObject class.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it will be redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.

Method	Description
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the `UIComponent` class

The following table lists the methods the `Alert` class inherits from the `UIComponent` class.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Methods inherited from the `Window` class

The following table lists the methods the `Alert` class inherits from the `Window` class.

Method	Description
<code>Window.deletePopup()</code>	Removes a window instance created by <code>PopupManager.createPopup()</code> .

Property summary for the `Alert` class

The following table lists properties of the `Alert` class.

Property	Description
<code>Alert.buttonHeight</code>	The height of each button, in pixels. The default value is 22.
<code>Alert.buttonWidth</code>	The width of each button, in pixels. The default value is 100.
<code>Alert.CANCEL</code>	A constant hexadecimal value indicating whether a Cancel button should be displayed in the <code>Alert</code> window.
<code>Alert.cancelLabel</code>	The label text for the Cancel button.
<code>Alert.click</code>	The label text for the No button.
<code>Alert.NO</code>	A constant hexadecimal value indicating whether a No button should be displayed in the <code>Alert</code> window.
<code>Alert.OK</code>	A constant hexadecimal value indicating whether an OK button should be displayed in the <code>Alert</code> window.
<code>Alert.okLabel</code>	The label text for the OK button.
<code>Alert.YES</code>	A constant hexadecimal value indicating whether a Yes button should be displayed in the <code>Alert</code> window.
<code>Alert.yesLabel</code>	The label text for the Yes button.

Properties inherited from the `UIObject` class

The following table lists the properties the `Alert` class inherits from the `UIObject` class. When calling these properties from the `Alert` object, use the form `Alert.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `Alert` class inherits from the `UIComponent` class. When calling these properties from the `Alert` object, use the form `Alert.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Properties inherited from the `Window` class

The following table lists the properties the `Alert` class inherits from the `Window` class.

Property	Description
<code>Window.closeButton</code>	Indicates whether a close button is (<code>true</code>) or is not (<code>false</code>) included on the title bar.
<code>Window.content</code>	A reference to the content (root movie clip) of the window.
<code>Window.contentPath</code>	Sets the name of the content to display in the window.

Property	Description
<code>Window.title</code>	The text that appears in the title bar.
<code>Window.titleStyleDeclaration</code>	The style declaration that formats the text in the title bar.

Event summary for the Alert class

The following table lists an event of the Alert class.

Event	Description
<code>Alert.click</code>	Broadcast when a button in an Alert window is clicked.

Events inherited from the UIObject class

The following table lists the events the Alert class inherits from the UIObject class. When calling these events from the Alert object, use the form `Alert.eventName`.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the Alert class inherits from the UIComponent class. When calling these events from the Alert object, use the form `Alert.eventName`.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Events inherited from the Window class

The following table lists the events the Alert class inherits from the Window class.

Event	Description
<code>Window.click</code>	Broadcast when the close button is clicked (released).

Event	Description
Window.complete	Broadcast when a window is created.
Window.mouseDownOutside	Broadcast when the mouse is clicked (released) outside the modal window.

Alert.buttonHeight

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

`Alert.buttonHeight`

Description

Property (class); a class (static) property that changes the height of the buttons. The default value is 22.

See also

[Alert.buttonWidth](#)

Alert.buttonWidth

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

`Alert.buttonWidth`

Description

Property (class); a class (static) property that changes the width of the buttons. The default value is 100.

See also

[Alert.buttonHeight](#)

Alert.CANCEL

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Alert.CANCEL

Description

Property (constant); a property with the constant hexadecimal value 0x8. This property can be used for the *flags* or *defaultButton* parameter of the `Alert.show()` method. When used as a value for the *flags* parameter, this property indicates that a Cancel button should be displayed in the Alert window. When used as a value for the *defaultButton* parameter, the Cancel button has initial focus and is triggered when the user presses Enter (Windows) or Return (Macintosh). If the user tabs to another button, that button is triggered when the user presses Enter.

Example

The following example uses `Alert.CANCEL` and `Alert.OK` as values for the *flags* parameter and displays an Alert component with an OK button and a Cancel button:

```
import mx.controls.Alert;
Alert.show("This is a generic Alert window", "Alert Test", Alert.OK |
    Alert.CANCEL, this);
```

Alert.cancelLabel

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Alert.cancelLabel

Description

Property (class); a class (static) property that indicates the label text on the Cancel button.

Example

The following example sets the Cancel button's label to "cancellation":

```
Alert.cancelLabel = "cancellation";
```

Alert.click

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
clickHandler = function(eventObject){
    // insert code here
}
Alert.show(message[, title[, flags[, parent[, clickHandler[, icon[,
    defaultButton]]]]]])
```

Description

Event; broadcast to the registered listener when the OK, Yes, No, or Cancel button is clicked.

Version 2 components use a dispatcher/listener event model. The Alert component dispatches a `click` event when one of its buttons is clicked and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You call the `Alert.show()` method and pass it the name of the handler as a parameter. When a button in the Alert window is clicked, the listener is called.

When the event occurs, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Alert.click` event's event object has an additional `detail` property whose value is `Alert.OK`, `Alert.CANCEL`, `Alert.YES`, or `Alert.NO`, depending on which button was clicked. For more information, see [“UIEventDispatcher class” on page 530](#).

Example

In the following example, a handler called `myClickHandler` is defined and passed to the `Alert.show()` method as the fifth parameter. The event object is captured by `myClickHandler` in the `evt` parameter. The `detail` property of the event object is then used in a `trace` statement to send the name of the button that was clicked (`Alert.OK` or `Alert.CANCEL`) to the Output panel.

```
import mx.controls.Alert;
myClickHandler = function(evt){
    if(evt.detail == Alert.OK){
        trace(Alert.okLabel);
    }else if (evt.detail == Alert.CANCEL){
        trace(Alert.cancelLabel);
    }
}
Alert.show("This is a test of errors", "Error", Alert.OK | Alert.CANCEL, this,
    myClickHandler);
```

Alert.NO

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Alert.NO

Description

Property (constant); a property with the constant hexadecimal value 0x2. This property can be used for the *flags* or *defaultButton* parameter of the `Alert.show()` method. When used as a value for the *flags* parameter, this property indicates that a No button should be displayed in the Alert window. When used as a value for the *defaultButton* parameter, the Cancel button has initial focus and is triggered when the user presses Enter (Windows) or Return (Macintosh). If the user tabs to another button, that button is triggered when the user presses Enter.

Example

The following example uses `Alert.NO` and `Alert.YES` as values for the *flags* parameter and displays an Alert component with a No button and a Yes button:

```
import mx.controls.Alert;
Alert.show("This is a generic Alert window", "Alert Test", Alert.NO |
    Alert.YES, this);
```

Alert.noLabel

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Alert.noLabel

Description

Property (class); a class (static) property that indicates the label text on the No button.

Example

The following example sets the No button's label to "nyet":

```
Alert.noLabel = "nyet";
```

Alert.OK

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Alert.OK

Description

Property (constant); a property with the constant hexadecimal value 0x4. This property can be used for the *flags* or *defaultButton* parameter of the `Alert.show()` method. When used as a value for the *flags* parameter, this property indicates that an OK button should be displayed in the Alert window. When used as a value for the *defaultButton* parameter, the OK button has initial focus and is triggered when the user presses Enter (Windows) or Return (Macintosh). If the user tabs to another button, that button is triggered when the user presses Enter.

Example

The following example uses `Alert.OK` and `Alert.CANCEL` as values for the *flags* parameter and displays an Alert component with an OK button and a Cancel button:

```
import mx.controls.Alert;
Alert.show("This is a generic Alert window", "Alert Test", Alert.OK |
    Alert.CANCEL, this);
```

Alert.okLabel

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Alert.okLabel

Description

Property (class); a class (static) property that indicates the label text on the OK button.

Example

The following example sets the OK button's label to "okay":

```
Alert.okLabel = "okay";
```

Alert.show()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
Alert.show(message[, title[, flags[, parent[, clickHandler[, icon[,  
    defaultButton]]]]]])
```

Parameters

message The message to display.

title The text in the Alert title bar. This parameter is optional; if you omit it, the title bar is blank.

flags An optional parameter that indicates the buttons to display in the Alert window. The default value is `Alert.OK`, which displays an OK button. When you use more than one value, separate the values with a `|` character. Use one or more of the following values: `Alert.OK`, `Alert.CANCEL`, `Alert.YES`, `Alert.NO`.

You can also use `Alert.NONMODAL` to indicate that the Alert window is nonmodal. A nonmodal window allows a user to interact with other windows in the application.

parent The parent window for the Alert component. The Alert window centers itself in the parent window. Use the value `null` or `undefined` to specify the `_root` Timeline. The parent window must inherit from the `UIComponent` class. You can register the parent window with `mx.core.View` to cause it to inherit from `UIComponent`. This parameter is optional.

clickHandler A handler for the `click` events broadcast when the buttons are clicked. In addition to the standard click event object properties, there is an additional `detail` property, which contains the flag value of the button that was clicked (`Alert.OK`, `Alert.CANCEL`, `Alert.YES`, `Alert.NO`). This handler can be a function or an object. For more information, see “Using listeners to handle events” in Flash Help.

icon A string that is the linkage identifier of a symbol in the library; this symbol is used as an icon displayed to the left of the alert text. This parameter is optional.

defaultButton Indicates which button has initial focus and is clicked when a user presses Enter (Windows) or Return (Macintosh). If a user tabs to another button, that button is triggered when the Enter key is pressed.

This parameter can be one of the following values: `Alert.OK`, `Alert.CANCEL`, `Alert.YES`, `Alert.NO`.

Returns

The Alert instance that is created.

Description

Method (class); a class (static) method that displays an Alert window with a message, an optional title, optional buttons, and an optional icon. The title of the alert appears at the top of the window and is left-aligned. The icon appears to the left of the message text. The buttons are centered below the message text and the icon.

Example

The following code is a simple example of a modal Alert window with an OK button:

```
mx.controls.Alert.show("Hello, world!");
```

The following code defines a click handler that sends a message to the Output panel about which button was clicked:

```
import mx.controls.Alert;
myClickHandler = function(evt){
    trace ("button " + evt.detail + " was clicked");
}
Alert.show("This is a test of errors", "Error", Alert.OK | Alert.CANCEL, this,
    myClickHandler);
```

The event object's `detail` property returns a number to represent each button. The OK button is 4, the Cancel button is 8, the Yes button is 1, and the No button is 2.

Note: You must have an Alert component in the library for this code to display an alert. To add the component to the library, drag it to the Stage and then delete it.

Alert.YES

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
Alert.YES
```

Description

Property (constant); a property with the constant hexadecimal value 0x1. This property can be used for the `flags` or `defaultButton` parameter of the `Alert.show()` method. When used as a value for the `flags` parameter, this property indicates that a Yes button should be displayed in the Alert window. When used as a value for the `defaultButton` parameter, the Yes button has initial focus and is triggered when the user presses Enter (Windows) or Return (Macintosh). If the user tabs to another button, that button is triggered when the user presses Enter.

Example

The following example uses `Alert.NO` and `Alert.YES` as values for the *flags* parameter and displays an Alert component with a No button and a Yes button:

```
import mx.controls.Alert;
Alert.show("This is a generic Alert window", "Alert Test", Alert.NO |
    Alert.YES, this);
```

Alert.yesLabel

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
Alert.yesLabel
```

Description

Property (class); a class (static) property that indicates the label text on the Yes button.

Example

The following example sets the OK button's label to "da":

```
Alert.yesLabel = "da";
```

Button component

The Button component is a resizable rectangular user interface button. 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 can be enabled or disabled 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 component and triggers the <code>click</code> event.
Tab	Moves focus to the next object.

For more information about controlling focus, see "Creating custom focus navigation" in Flash Help or "[FocusManager class](#)" on page 231.

A live preview of each Button instance reflects changes made to parameters in the Property inspector or Component inspector during authoring. However, in the live preview a custom icon is represented on the Stage by a gray square.

When you add the Button component to an application, you can use the Accessibility panel to make it accessible to screen readers. First, you must add the following line of code:

```
mx.accessibility.ButtonAccImpl.enableAccessibility();
```

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

Using the Button component

A button is a fundamental part of any form or web application. 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.

To add an icon to a button, you need to select or create a movie clip or graphic symbol to use as the icon. The symbol should be registered at 0,0 for appropriate layout on the button. Select the icon symbol in the Library panel, open the Linkage dialog box from the Library options menu, and enter a linkage identifier. This is the value to enter for the icon parameter in the Property inspector or Component inspector. You can also enter this value for the `Button.icon` ActionScript property.

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), use `FocusManager.defaultPushButton`.

Button parameters

You can set the following authoring parameters for each Button component instance in the Property inspector or in the Component inspector:

label sets the value of the text on the button; the default value is `Button`.

icon adds a custom icon to the button. The value is the linkage identifier of a movie clip or graphic symbol in the library; there is no default value.

toggle turns the button into a toggle switch. If `true`, the button remains in the down state when clicked and returns to the up state when clicked again. If `false`, the button behaves like a normal push button; the default value is `false`.

selected if the toggle parameter is `true`, this parameter specifies whether the button is pressed (`true`) or released (`false`). The default value is `false`.

labelPlacement orients the label text on the button in relation to the icon. This parameter can be one of four values: `left`, `right`, `top`, or `bottom`; the default value is `right`. For more information, see `Button.labelPlacement`.

You can write ActionScript to control these and additional options for the Button component using its properties, methods, and events. For more information, see “Button class” on page 78.

Creating 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 is a Help button with a custom icon that opens a Help system when a user clicks it.

To create an application with the Button component:

1. Drag a Button component from the Components panel to the Stage.
2. In the Property inspector, enter the instance name **helpBtn**.
3. In the Property inspector, do the following:
 - Enter **Help** for the label parameter.
 - Enter **HelpIcon** for the icon parameter.

To use an icon, there must be a movie clip or graphic symbol in the library with a linkage identifier to use as the icon parameter. In this example, the linkage identifier is HelpIcon.
 - Set the **toggle** property to **true**.
4. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code:

```
function click(evt){
    clippyHelper.enabled = evt.target.selected;
}
helpBtn.addEventListener("click", this);
```

The last line of code adds a `click` event handler to the `helpBtn` instance. The handler enables and disables the `clippyHelper` instance, which could be a Help panel of some sort.

Customizing the Button component

You can transform a Button component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the **Modify > Transform** commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)) or any applicable properties and methods of the Button class (see “[Button class](#)” on page 78). Resizing the button does not change the size of the icon or label.

The bounding box of a Button 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.

If an icon is larger than the button, the icon extends beyond the button’s borders.

Using styles with the Button component

You can set style properties to change the appearance of a button instance. If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in Flash Help.

A Button component supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>backgroundColor</code>	Sample	The background color. The default value is 0xEFEBEF (light gray). The Halo theme uses 0xF8F8F8 (very light gray) for the button background color when the button is up and <code>themeColor</code> when the button is pressed. You can only modify the up background color in the Halo theme by skinning the button. See "Using skins with the Button component" on page 73 .
<code>border styles</code>	Sample	The Button component uses a <code>RectBorder</code> instance as its border in the Sample theme and responds to the styles defined on that class. See "RectBorder class" in Flash Help. With the Halo theme, the Button component uses a custom rounded border whose colors cannot be modified except for <code>themeColor</code> .
<code>color</code>	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is "_sans".
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either "normal" or "italic". The default value is "normal".
<code>fontWeight</code>	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return "none".
<code>textDecoration</code>	Both	The text decoration: either "none" or "underline". The default value is "none".

Using skins with the Button component

The Button component includes 32 different skins that can be customized to correspond to the border and icon in 16 different states. To skin the Button component while authoring, create new movie clip symbols with the desired graphics and set the symbol linkage identifiers using ActionScript. (See “Using ActionScript to draw Button skins” on page 75.)

The default implementation of the Button skins provided with both the Halo and Sample themes uses the ActionScript drawing API to draw the button states, and uses a single movie clip symbol associated with one ActionScript class to provide all skins for the Button component.

The Button component has many skins because a button has so many states, and a border and icon for each state. The state of a Button instance is controlled by four properties and user interaction. The properties that affect skins include the following:

Property	Description
<code>emphasized</code>	Provides two different looks for Button instances and is typically used to highlight one button, such as the default button in a form.
<code>enabled</code>	Shows whether the button allows user interaction.
<code>toggle</code>	Toggle buttons provide a selected and unselected value and use different skins to demonstrate the current value. For a Button instance whose <code>toggle</code> property is set to <code>false</code> , the <code>false</code> skins are used. When the <code>toggle</code> property is <code>true</code> , the skin used depends on the <code>selected</code> property.
<code>selected</code>	When the <code>toggle</code> property is set to <code>true</code> , this property determines if the Button is selected (<code>true</code> or <code>false</code>). Different skins are used to identify the value and by default are the only way this value is depicted on screen.

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 and it retains input focus. If the `toggle` parameter is set to `true`, the state of the button does not change until the mouse is released over it.

If a button is disabled, it displays its disabled state, regardless of user interaction.

A Button component supports the following skin properties:

Property	Description
<code>falseUpSkin</code>	The up (normal) state.
<code>falseDownSkin</code>	The pressed state.
<code>falseOverSkin</code>	The over state.
<code>falseDisabledSkin</code>	The disabled state.
<code>trueUpSkin</code>	The toggled state.
<code>trueDownSkin</code>	The pressed-toggled state.

Property	Description
<code>trueOverSkin</code>	The over-toggled state.
<code>trueDisabledSkin</code>	The disabled-toggled state.
<code>falseUpSkinEmphasized</code>	The up (normal) state of an emphasized button.
<code>falseDownSkinEmphasized</code>	The pressed state of an emphasized button.
<code>falseOverSkinEmphasized</code>	The over state of an emphasized button.
<code>falseDisabledSkinEmphasized</code>	The disabled state of an emphasized button.
<code>trueUpSkinEmphasized</code>	The toggled state of an emphasized button.
<code>trueDownSkinEmphasized</code>	The pressed-toggled state of an emphasized button.
<code>trueOverSkinEmphasized</code>	The over-toggled state of an emphasized button.
<code>trueDisabledSkinEmphasized</code>	The disabled-toggled state of an emphasized button.
<code>falseUpIcon</code>	The icon up state.
<code>falseDownIcon</code>	The icon pressed state.
<code>falseOverIcon</code>	The icon over state.
<code>falseDisabledIcon</code>	The icon disabled state.
<code>trueUpIcon</code>	The icon toggled state.
<code>trueOverIcon</code>	The icon over-toggled state.
<code>trueDownIcon</code>	The icon pressed-toggled state.
<code>trueDisabledIcon</code>	The icon disabled-toggled state.
<code>falseUpIconEmphasized</code>	The icon up state of an emphasized button.
<code>falseDownIconEmphasized</code>	The icon pressed state of an emphasized button.
<code>falseOverIconEmphasized</code>	The icon over state of an emphasized button.
<code>falseDisabledIconEmphasized</code>	The icon disabled state of an emphasized button.
<code>trueUpIconEmphasized</code>	The icon toggled state of an emphasized button.
<code>trueOverIconEmphasized</code>	The icon over-toggled state of an emphasized button.
<code>trueDownIconEmphasized</code>	The icon pressed-toggled state of an emphasized button.
<code>trueDisabledIconEmphasized</code>	The icon disabled-toggled state of an emphasized button.

The default value for all skin properties ending in “Skin” is `ButtonSkin`, and the default for all “Icon” properties is `undefined`. The properties with the “Skin” suffix provide a background and border, whereas those with the “Icon” suffix provide a small icon.

In addition to the icon skins, the `Button` component also supports a standard `icon` property. The difference between the standard property and style property is that through the style property you can set icons for the individual states, whereas with the standard property only one icon can be set and it applies to all states. If a `Button` instance has both the `icon` property and `icon style` properties set, the instance may not behave as anticipated.

To see an interactive movie demonstrating when each skin is used, see *Using Components* in Flash Help.

Using ActionScript to draw Button skins

The default skins in both the Halo and Sample themes use the same skin element for all states and draw the actual graphics through ActionScript. The Halo implementation uses an extension of the `RectBorder` class and custom drawing API code to draw the states. The Sample implementation uses the same skin and the same ActionScript class as the Button skin.

To create an ActionScript class to use as the skin and provide different states, the skin can read the `borderStyle` style property of the skin and `emphasized` property of the parent to determine the state. The following table shows the border style that is set for each skin:

Property	Border style
<code>falseUpSkin</code>	<code>falseup</code>
<code>falseDownSkin</code>	<code>falsedown</code>
<code>falseOverSkin</code>	<code>falserollover</code>
<code>falseDisabled</code>	<code>falsedisabled</code>
<code>trueUpSkin</code>	<code>trueup</code>
<code>trueDownSkin</code>	<code>trueedown</code>
<code>trueOverSkin</code>	<code>trueerollover</code>
<code>trueDisabledSkin</code>	<code>trueedisable</code>

To create an ActionScript customized Button skin:

1. Create a new ActionScript class file.

For this example, name the file **RedGreenBlueSkin.as**.

2. Copy the following ActionScript to the file:

```
import mx.skins.RectBorder;
import mx.core.ext.UIObjectExtensions;

class RedGreenBlueSkin extends RectBorder
{
    static var symbolName:String = "RedGreenBlueSkin";
    static var symbolOwner:Object = RedGreenBlueSkin;

    function size():Void
    {
        var c:Number; // color
        var borderStyle:String = getStyle("borderStyle");

        switch (borderStyle) {
            case "falseup":
            case "falserollover":
            case "falsedisabled":
                c = 0x7777FF;
                break;
```

```

        case "falsedown":
            c = 0x77FF77;
            break;
        case "trueup":
        case "truedown":
        case "truerollover":
        case "truedisabled":
            c = 0xFF7777;
            break;
    }

    clear();
    var thickness = _parent.emphasized ? 2 : 0;
   LineStyle(thickness, 0, 100);
    beginFill(c, 100);
    drawRect(0, 0, __width, __height);
    endFill();
}

// required for skins
static function classConstruct():Boolean
{
    UIObjectExtensions.Extensions();
    _global.skinRegistry["ButtonSkin"] = true;
    return true;
}
static var classConstructed:Boolean = classConstruct();
static var UIObjectExtensionsDependency = UIObjectExtensions;
}

```

This class creates a square box based on the border style: a blue box for the false up, rollover, and disabled states; a green box for the normal pressed state; and a red box for the expanded child. It draws a hairline border in the normal case and a thick border if the button is emphasized.

3. Save the file.
4. Create a new FLA file.
5. Save the FLA file in the same folder as the AS file.
6. Create a new symbol by selecting Insert > New Symbol.
7. Set the name to `ButtonSkin`.
8. If the advanced view is not displayed, click the Advanced button.
9. Select Export for ActionScript.

The identifier will be automatically filled out with `ButtonSkin`.
10. Set the AS 2.0 class to `RedGreenBlueSkin`.
11. Ensure that Export in First Frame is already selected, and click OK.
12. Drag a Button component to the Stage.
13. Select Control > Test Movie.

Using movie clips to customize Button skins

The above example demonstrates how to use an ActionScript class to customize the Button skin, which is the method used by the skins provided in both the Halo and Sample themes. However, because the example uses simple colored boxes, it is simpler in this case to use different movie clip symbols as the skins.

To create movie clip symbols for Button skins:

1. Create a new FLA file.
2. Create a new symbol by selecting Insert > New Symbol.
3. Set the name to `RedButtonSkin`.
4. If the advanced view is not displayed, click the Advanced button.
5. Select Export for ActionScript.

The identifier will be automatically filled out with `RedButtonSkin`.
6. Set the AS 2.0 class to `mx.skins.SkinElement`.
7. Ensure that Export in First Frame is already selected, and click OK.
8. Open the new symbol for editing.
9. Use the drawing tools to create a box with a red fill and black line.
10. Set the border style to hairline.
11. Set the box, including the border, so that it is positioned at (0,0) and has a width and height of 100.

The `SkinElement` class resizes the content as appropriate.
12. Repeat steps 2-11 and create green and blue skins, named accordingly.
13. Click the Back button to return to the main Timeline.
14. Drag a Button component to the Stage.
15. Set the `toggled` property value to `true` to see all three skins.
16. Copy the following ActionScript code to the Actions panel with the Button instance selected.

```
onClipEvent(initialize) {
    falseUpSkin = "BlueButtonSkin";
    falseDownSkin = "GreenButtonSkin";
    falseOverSkin = "BlueButtonSkin";
    falseDisabledSkin = "BlueButtonSkin";
    trueUpSkin = "RedButtonSkin";
    trueDownSkin = "RedButtonSkin";
    trueOverSkin = "RedButtonSkin";
    trueDisabledSkin = "RedButtonSkin";
}
```
17. Select Control > Test Movie.

Button class

Inheritance [MovieClip](#) > [UIObject class](#) > [UIComponent class](#) > [Button component](#) > Button

ActionScript Class Name `mx.controls.Button`

The properties of the Button class let you do the following at runtime: add an icon to a button, create a text label, and indicate whether the button acts as a push button or as a toggle switch.

Setting a property of the Button class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

The Button component uses the Focus Manager to override the default Flash Player focus rectangle and draw a custom focus rectangle with rounded corners. For more information, see “Creating custom focus navigation” in Flash Help.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.Button.version);
```

Note: The code `trace(myButtonInstance.version);` returns `undefined`.

The Button component class is different from the built-in ActionScript Button object.

Method summary for the Button class

There are no methods exclusive to the Button class.

Methods inherited from the UIObject class

The following table lists the methods the Button class inherits from the UIObject class. When calling these methods from the Button object, use the form `buttonInstance.methodName`.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it will be redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Methods inherited from the `UIComponent` class

The following table lists the methods the `Button` class inherits from the `UIComponent` class. When calling these methods from the `Button` object, use the form *buttonInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the `Button` class

The following table lists properties of the `Button` class.

Property	Description
<code>Button.icon</code>	Specifies an icon for a button instance.
<code>Button.label</code>	Specifies the text that appears in a button.
<code>Button.labelPlacement</code>	Specifies the orientation of the label text in relation to an icon.

Properties inherited from the `SimpleButton` class

The following table lists the properties the `Button` class inherits from the `SimpleButton` class. When accessing these properties, use the form *buttonInstance.propertyName*.

Property	Description
<code>SimpleButton.emphasized</code>	Indicates whether a button has the look of a default push button.
<code>SimpleButton.emphasizedStyleDeclaration</code>	The style declaration when the <code>emphasized</code> property is set to <code>true</code> .
<code>SimpleButton.selected</code>	A Boolean value indicating whether the button is selected (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .
<code>SimpleButton.toggle</code>	A Boolean value indicating whether the button behaves as a toggle switch (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .

Properties inherited from the `UIObject` class

The following table lists the properties the `Button` class inherits from the `UIObject` class. When accessing these properties from the `Button` object, use the form *buttonInstance.propertyName*.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.

Property	Description
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `Button` class inherits from the `UIComponent` class. When accessing these properties from the `Button` object, use the form `buttonInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the `Button` class

There are no events exclusive to the `Button` class.

Events inherited from the `SimpleButton` class

The following table lists the events the `Button` class inherits from the `SimpleButton` class.

Property	Description
<code>SimpleButton.click</code>	Broadcast when a button is clicked.

Events inherited from the `UIObject` class

The following table lists the events the `Button` class inherits from the `UIObject` class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.

Event	Description
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the `UIComponent` class

The following table lists the events the `Button` class inherits from the `UIComponent` class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Button.icon

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

`buttonInstance.icon`

Description

Property; a string that specifies the linkage identifier of a symbol in the library to be used as an icon for a button instance. The icon can be a movie clip symbol or a graphic symbol with an upper left registration point. You must resize the button if the icon is too large to fit; neither the button nor the icon resizes automatically. If an icon is larger than a button, the icon extends over the borders of the button.

To create a custom icon, create a movie clip or graphic symbol. Select the symbol on the Stage in symbol-editing mode and enter 0 in both the X and Y boxes in the Property inspector. In the Library panel, select the movie clip and select `Linkage` from the Library options menu. Select `Export for ActionScript`, and enter an identifier in the Identifier text box.

The default value is an empty string (""), which indicates that there is no icon.

Use the `labelPlacement` property to set the position of the icon in relation to the button.

Note: The icon does not appear on the Stage in Flash. You must choose `Control > Test Movie` to see the icon.

Example

The following code assigns the movie clip from the Library panel with the linkage identifier `happiness` to the `Button` instance as an icon:

```
myButton.icon = "happiness"
```

See also

[Button.labelPlacement](#)

Button.label

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
buttonInstance.label
```

Description

Property; specifies the text label for a button instance. By default, the label appears centered on the button. Calling this method overrides the label authoring parameter specified in the Property inspector or the Component inspector. The default value is `"Button"`.

Example

The following code sets the label to “Remove from list”:

```
buttonInstance.label = "Remove from list";
```

See also

[Button.labelPlacement](#)

Button.labelPlacement

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
buttonInstance.labelPlacement
```

Description

Property; sets the position of the label in relation to the icon. The default value is "right". The following are the four possible values; the icon and label are always centered vertically and horizontally within the bounding area of the button:

- "right" The label is set to the right of the icon.
- "left" The label is set to the left of the icon.
- "bottom" The label is set below the icon.
- "top" The label is set above the icon.

Example

The following code sets the label to the left of the icon. The second line of the code sends the value of the `labelPlacement` property to the Output panel:

```
iconInstance.labelPlacement = "left";  
trace(iconInstance.labelPlacement);
```

CheckBox component

A check box 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 check box and place it to the left, right, top, or bottom.

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

If a check box is disabled, it displays its disabled appearance, regardless of user interaction. In the disabled state, a button 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 <code>click</code> event.
Tab	Moves focus to the next element.

For more information about controlling focus, see "Creating custom focus navigation" in Flash Help or "[FocusManager class](#)" on page 231.

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 use the Accessibility panel to make it accessible to screen readers. First, you must add the following line of code to enable accessibility:

```
mx.accessibility.CheckBoxAccImpl.enableAccessibility();
```

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

Using the `CheckBox` component

A check box is a fundamental part of any form or web application. You can use check boxes wherever you need to gather a set of `true` or `false` values that aren't mutually exclusive. For example, a form collecting personal information about a customer could have a list of hobbies for the customer to select; each hobby would have a check box beside it.

CheckBox parameters

You can set the following authoring parameters for each `CheckBox` component instance in the Property inspector or in the Component inspector:

label sets the value of the text on the check box; the default value is `defaultValue`.

selected sets the initial value of the check box to checked (`true`) or unchecked (`false`).

labelPlacement orients the label text on the check box. This parameter can be one of four values: `left`, `right`, `top`, or `bottom`; the default value is `right`. For more information, see [CheckBox.labelPlacement](#).

You can write ActionScript to control these and additional options for the `CheckBox` component using its properties, methods, and events. For more information, see [“CheckBox class” on page 87](#).

Creating an application with the `CheckBox` component

The following procedure explains how to add a `CheckBox` component to an application while authoring. The following example is a form for an online dating application. The form is a query that searches for possible dating matches for the customer. The query form must have a check box labeled `Restrict Age` that permits customers to restrict their search to a specified age group. When the `Restrict Age` check box is selected, the customer can then enter the minimum and maximum ages into two text fields. (These text fields are enabled only when the check box is selected.)

To create an application with the `CheckBox` component:

1. Drag two `TextInput` components from the Components panel to the Stage.
2. In the Property inspector, enter the instance names `minimumAge` and `maximumAge`.
3. Drag a `CheckBox` component from the Components panel to the Stage.
4. In the Property inspector, do the following:
 - Enter `restrictAge` for the instance name.
 - Enter `Restrict Age` for the label parameter.

5. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code:

```
restrictAgeListener = new Object();
restrictAgeListener.click = function (evt){
    minimumAge.enabled = evt.target.selected;
    maximumAge.enabled = evt.target.selected;
}
restrictAge.addEventListener("click", restrictAgeListener);
```

This code creates a `click` event handler that enables and disables the `minimumAge` and `maximumAge` text field components, which have already been placed on Stage. For more information, see [CheckBox.click](#) and “[TextInput component](#)” on page 445.

Customizing the CheckBox component

You can transform a `CheckBox` component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the `Modify > Transform` commands. At runtime, use the `setSize()` method (`UIObject.setSize()`) or any applicable properties and methods of the [CheckBox class](#).

Resizing the check box 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.

Using styles with the CheckBox component

You can set style properties to change the appearance of a `CheckBox` instance. If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in Flash Help.

A `CheckBox` component supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are <code>haloGreen</code> , <code>haloBlue</code> , and <code>haloOrange</code> . The default value is <code>haloGreen</code> .
<code>color</code>	Both	The text color. The default value is <code>0x0B333C</code> for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is <code>0x848384</code> (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .

Style	Theme	Description
fontFamily	Both	The font name for text. The default value is "_sans".
fontSize	Both	The point size for the font. The default value is 10.
fontStyle	Both	The font style: either "normal" or "italic". The default value is "normal".
fontWeight	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return "none".
textDecoration	Both	The text decoration: either "none" or "underline". The default value is "none".
symbolBackgroundColor	Sample	The background color of the check box. The default value is 0xFFFFFFFF (white).
symbolBackgroundDisabledColor	Sample	The background color of the check box when disabled. The default value is 0xEFEEEE (light gray).
symbolBackgroundPressedColor	Sample	The background color of the check box when pressed. The default value is 0xFFFFFFFF (white).
symbolColor	Sample	The color of the check mark. The default value is 0x000000 (black).
symbolDisabledColor	Sample	The color of the disabled check mark. The default value is 0x848384 (dark gray).

Using skins with the CheckBox component

The CheckBox component uses symbols in the library to represent the button states. To skin the CheckBox component while authoring, modify symbols in the Library panel. The CheckBox component skins are located in the Flash UI Components 2/Themes/MMDefault/CheckBox Assets/states folder in the library of either the HaloTheme.fla file or the SampleTheme.fla file. For more information, see “About skinning components” in Flash Help.

A CheckBox component uses the following skin properties:

Property	Description
falseUpSkin	The up (normal) unchecked state. The default is <code>CheckFalseUp</code> .
falseDownSkin	The pressed unchecked state. The default is <code>CheckFalseDown</code> .
falseOverSkin	The over unchecked state. The default is <code>CheckFalseOver</code> .
falseDisabledSkin	The disabled unchecked state. The default is <code>CheckFalseDisabled</code> .
trueUpSkin	The toggled checked state. The default is <code>CheckTrueUp</code> .
trueDownSkin	The pressed checked state. The default is <code>CheckTrueDown</code> .
trueOverSkin	The over checked state. The default is <code>CheckTrueOver</code> .
trueDisabledSkin	The disabled checked state. The default is <code>CheckTrueDisabled</code> .

Each of these skins corresponds to the icon indicating the CheckBox state. The CheckBox component does not have a border or background.

To create movie clip symbols for CheckBox skins:

1. Create a new FLA file.
2. Select File > Import > Open External Library, and select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in Flash Help.
3. In the theme’s Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the CheckBox Assets folder to the library for your document.
4. Expand the CheckBox Assets/States folder in the library of your document.
5. Open the symbols you want to customize for editing.
For example, open the CheckFalseDisabled symbol.
6. Customize the symbol as desired.
For example, change the inner white square to a light gray.
7. Repeat steps 5-6 for all symbols you want to customize.
For example, repeat the color change for the inner box of the CheckTrueDisabled symbol.
8. Click the Back button to return to the main Timeline.
9. Drag a CheckBox component to the Stage.
For this example, drag two instances to show the two new skin symbols.
10. Set the CheckBox instance properties as desired.
For this example, set one CheckBox instance to `true`, and use ActionScript to set both CheckBox instances to disabled.
11. Select Control > Test Movie.

CheckBox class

Inheritance MovieClip > UIObject class > UIComponent class > Button component > Button component > CheckBox

ActionScript Class Name mx.controls.CheckBox

The properties of the CheckBox class let you create a text label and position it to the left, right, top, or bottom of a check box at runtime.

Setting a property of the CheckBox class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

The CheckBox component uses the Focus Manager to override the default Flash Player focus rectangle and draw a custom focus rectangle with rounded corners. For more information, see “Creating custom focus navigation” in Flash Help.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.CheckBox.version);
```

Note: The code `trace(myCheckBoxInstance.version);` returns `undefined`.

Method summary for the `CheckBox` class

There are no methods exclusive to the `CheckBox` class.

Methods inherited from the `UIObject` class

The following table lists the methods the `CheckBox` class inherits from the `UIObject` class. When calling these methods from the `CheckBox` object, use the form `checkBoxInstance.methodName`.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it will be redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the `UIComponent` class

The following table lists the methods the `CheckBox` class inherits from the `UIComponent` class. When calling these methods from the `CheckBox` object, use the form `checkBoxInstance.methodName`.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the CheckBox class

The following table lists properties of the CheckBox class.

Property	Description
<code>CheckBox.label</code>	Specifies the text that appears next to a check box.
<code>CheckBox.labelPlacement</code>	Specifies the orientation of the label text in relation to a check box.
<code>CheckBox.selected</code>	Specifies whether the check box is selected (<code>true</code>) or deselected (<code>false</code>).

Properties inherited from the UIObject class

The following table lists the properties the CheckBox class inherits from the UIObject class.

When accessing these properties from the CheckBox object, use the form

`checkBoxInstance.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `CheckBox` class inherits from the `UIComponent` class. When accessing these properties from the `CheckBox` object, use the form `checkBoxInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Properties inherited from the `SimpleButton` class

The following table lists the properties the `CheckBox` class inherits from the `SimpleButton` class. When accessing these properties from the `CheckBox` object, use the form `checkBoxInstance.propertyName`.

Property	Description
<code>SimpleButton.emphasized</code>	Indicates whether a button has the appearance of a default push button.
<code>SimpleButton.emphasizedStyleDeclaration</code>	The style declaration when the <code>emphasized</code> property is set to <code>true</code> .
<code>SimpleButton.selected</code>	A Boolean value indicating whether the button is selected (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .
<code>SimpleButton.toggle</code>	A Boolean value indicating whether the button behaves as a toggle switch (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .

Properties inherited from the `Button` class

The following table lists the properties the `CheckBox` class inherits from the `Button` class. When accessing these properties from the `CheckBox` object, use the form `checkBoxInstance.propertyName`.

Property	Description
<code>Button.label</code>	Specifies the text that appears in a button.
<code>Button.labelPlacement</code>	Specifies the orientation of the label text in relation to an icon.

Event summary for the `CheckBox` class

The following table lists an event of the `CheckBox` class.

Event	Description
<code>CheckBox.click</code>	Triggered when the mouse is clicked (released) over the check box, or if the check box has focus and the Spacebar is pressed.

Events inherited from the UIObject class

The following table lists the events the CheckBox class inherits from the UIObject class.

Event	Description
UIObject.draw	Broadcast when an object is about to draw its graphics.
UIObject.hide	Broadcast when an object's state changes from visible to invisible.
UIObject.load	Broadcast when subobjects are being created.
UIObject.move	Broadcast when the object has moved.
UIObject.resize	Broadcast when an object has been resized.
UIObject.reveal	Broadcast when an object's state changes from invisible to visible.
UIObject.unload	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the CheckBox class inherits from the UIComponent class.

Event	Description
UIComponent.focusIn	Broadcast when an object receives focus.
UIComponent.focusOut	Broadcast when an object loses focus.
UIComponent.keyDown	Broadcast when a key is pressed.
UIComponent.keyUp	Broadcast when a key is released.

Events inherited from the SimpleButton class

The following table lists the event the CheckBox class inherits from the SimpleButton class.

Event	Description
SimpleButton.click	Broadcast when a button is clicked.

CheckBox.click

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(click){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();
listenerObject.click = function(eventObject){
    ...
}
checkBoxInstance.addEventListener("click", listenerObject)
```

Description

Event; broadcast to all registered listeners when the mouse is clicked (released) over the check box, or if the check box has focus and the Spacebar is pressed.

The first usage example uses an `on()` handler and must be attached directly to a `CheckBox` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the check box `myCheckBox`, sends “_level0.myCheckBox” to the Output panel:

```
on(click){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (`checkBoxInstance`) dispatches an event (in this case, `click`), and the event is handled by a function, also called a *handler*, on a listener object (`listenerObject`) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (`eventObject`) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `addEventListener()` method (see `EventDispatcher.addEventListener()` in Flash Help) on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

This example, written on a frame of the Timeline, sends a message to the Output panel when a button called `checkBoxInstance` is clicked. The first line of code creates a listener object called `form`. The second line defines a function for the `click` event on the listener object. Inside the function is a `trace()` statement that uses the event object that is automatically passed to the function (in this example, `eventObj`) to generate a message. The `target` property of an event object is the component that generated the event (in this example, `checkBoxInstance`). The `CheckBox.selected` property is accessed from the event object’s `target` property. The last line calls `addEventListener()` from `checkBoxInstance` and passes it the `click` event and the `form` listener object as parameters.

```
form = new Object();
form.click = function(eventObj){
    trace("The selected property has changed to " + eventObj.target.selected);
}
checkBoxInstance.addEventListener("click", form);
```

The following code also sends a message to the Output panel when `checkBoxInstance` is clicked. The `on()` handler must be attached directly to `checkBoxInstance`:

```
on(click){
    trace("check box component was clicked");
}
```

See also

[EventDispatcher.addEventListener\(\)](#) in Flash Help

CheckBox.label

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

checkBoxInstance.label

Description

Property; indicates the text label for the check box. By default, the label appears to the right of the check box. Setting this property overrides the label parameter specified in the Parameters tab of the Component Inspector panel.

Example

The following code sets the text that appears beside the CheckBox component and sends the value to the Output panel:

```
checkBox.label = "Remove from list";
trace(checkBox.label)
```

See also

[CheckBox.labelPlacement](#)

CheckBox.labelPlacement

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

checkBoxInstance.labelPlacement

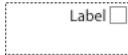
Description

Property; a string that indicates the position of the label in relation to the check box. The following are the four possible values (the dotted lines represent the bounding area of the component; they are invisible in a document):

- "right" The check box is pinned to the upper left corner of the bounding area. The label is set to the right of the check box. This is the default value.



- "left" The check box is pinned to the upper right corner of the bounding area. The label is set to the left of the check box.



- "bottom" The label is set below the check box. The check box and label are centered horizontally and vertically.



- "top" The label is placed below the check box. The check box and label are centered horizontally and vertically.



You can change the bounding area of a component while authoring by using the Transform command or at runtime using the `UIObject.setSize()` property. For more information, see [“Customizing the CheckBox component” on page 85](#).

Example

The following example sets the placement of the label to the left of the check box:

```
checkBox_mc.labelPlacement = "left";
```

See also

[CheckBox.label](#)

CheckBox.selected

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
checkBoxInstance.selected
```

Description

Property; a Boolean value that selects (`true`) or deselects (`false`) the check box.

Example

The following example selects the instance `checkbox1`:

```
checkbox1.selected = true;
```

CloseButton component

The `CloseButton` component is a small, round button with an *x* in it. Use this button in your pods and notices to close notices and pod viewers for your application in the Console. The button changes state when you move a mouse pointer over it.



CloseButton component in a collapsed pod viewer

Using the CloseButton component

The `CloseButton` component has the same methods, properties, and events as the `Button` object. For detailed information on the `Button` object, see the help system in your Flash authoring tool. However, for most developers, use of the `CloseButton` component will simply require the `onRelease` event handler, which is documented in this book.

MCloseButton.onRelease

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myCloseButton.onRelease()
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the button is released. You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onRelease` method that sends a trace action to the Output panel:

```
myCloseButton.onRelease = function () {  
    trace ("onRelease called");  
};
```

ComboBox component

A combo box allows a user to make a single selection from a drop-down list. A combo box can be static or editable. An editable combo box allows a user to enter text directly into a text field at the top of the list, as well as selecting an item from a drop-down list. If the drop-down list hits the bottom of the document, it opens up instead of down. The combo box is composed of three subcomponents: a Button component, a TextInput component, and a List component.

When a selection is made in the list, the label of the selection is copied to the text field at the top of the combo box. It doesn't matter if the selection is made with the mouse or the keyboard.

A ComboBox component receives focus if you click the text box or the button. When a ComboBox component has focus and is editable, all keystrokes go to the text box and are handled according to the rules of the TextInput component (see [“TextInput component” on page 445](#)), with the exception of the following keys:

Key	Description
Control+Down Arrow	Opens the drop-down list and gives it focus.
Shift+Tab	Moves focus to the previous object.
Tab	Moves focus to the next object.

When a ComboBox component has focus and is static, alphanumeric keystrokes move the selection up and down the drop-down list to the next item with the same first character. You can also use the following keys to control a static combo box:

Key	Description
Control+Down Arrow	Opens the drop-down list and gives it focus.
Control+Up Arrow	Closes the drop-down list, if open in the stand-alone and browser versions of Flash Player.
Down Arrow	Moves the selection down one item.
End	Selection moves to the bottom of the list.
Escape	Closes the drop-down list and returns focus to the combo box in Test Movie mode.
Enter	Closes the drop-down list and returns focus to the combo box.
Home	Moves the selection to the top of the list.
Page Down	Moves the selection down one page.

Key	Description
Page Up	Moves the selection up one page.
Shift+Tab	Moves focus to the previous object.
Tab	Moves focus to the next object.

When the drop-down list of a combo box has focus, alphanumeric keystrokes move the selection up and down the drop-down list to the next item with the same first character. You can also use the following keys to control a drop-down list:

Key	Description
Control+Up Arrow	If the drop-down list is open, focus returns to the text box and the drop-down list closes in the stand-alone and browser versions of Flash Player.
Down Arrow	Moves the selection down one item.
End	Moves the insertion point to the end of the text box.
Enter	If the drop-down list is open, focus returns to the text box and the drop-down list closes.
Escape	If the drop-down list is open, focus returns to the text box and the drop-down list closes in Test Movie mode.
Home	Moves the insertion point to the beginning of the text box.
Page Down	Moves the selection down one page.
Page Up	Moves the selection up one page.
Tab	Moves focus to the next object.
Shift+End	Selects the text from the insertion point to the End position.
Shift+Home	Selects the text from the insertion point to the Home position.
Shift+Tab	Moves focus to the previous object.
Up Arrow	Moves the selection up one item.

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 will show items 0-9, 9-18, 18-27, and so on, with one item overlapping per page.

For more information about controlling focus, see “Creating custom focus navigation” in Flash Help or [“FocusManager class” on page 231](#).

A live preview of each ComboBox component instance on the Stage reflects changes made to parameters in the Property inspector or Component inspector during authoring. However, the drop-down list does not open in the live preview, and the first item displays as the selected item.

When you add the ComboBox component to an application, you can use the Accessibility panel to make it accessible to screen readers. First, you must add the following line of code to enable accessibility:

```
mx.accessibility.ComboBoxAccImpl.enableAccessibility();
```

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

Using 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 combo box for more complex scenarios. For example, in an application that provides driving directions, you could use an editable combo box for a user to enter her origin and destination addresses. The drop-down list would contain her previously entered addresses.

ComboBox parameters

You can set the following authoring parameters for each ComboBox component instance in the Property inspector or in the Component inspector:

editable determines if the ComboBox component is editable (`true`) or only selectable (`false`). The default value is `false`.

labels populates the ComboBox component with an array of text values.

data associates a data value with each item in the ComboBox component. The data parameter is an array.

rowCount sets the maximum number of items that can be displayed in the list. The default value is 5.

You can write ActionScript to set additional options for ComboBox instances using the methods, properties, and events of the ComboBox class. For more information, see [“ComboBox class” on page 102](#).

Creating an application with the ComboBox component

The following procedure explains how to add a ComboBox component to an application while authoring. In this example, the combo box presents a list of cities to select from in its drop-down list.

To create an application with the ComboBox component:

1. Drag a ComboBox component from the Components panel to the Stage.
2. Select the Transform tool and resize the component on the Stage.

The combo box can only be resized on the Stage during authoring. Typically, you would only change the width of a combo box to fit its entries.

3. Select the combo box and, in the Property inspector, enter the instance name **comboBox**.

4. In the Component inspector or Property inspector, do the following:
 - Enter **Minneapolis**, **Portland**, and **Keene** for the label parameter. Double-click the label parameter field to open the Values dialog box. Then click the plus sign to add items.
 - Enter **MN.swf**, **OR.swf**, and **NH.swf** for the data parameter.
These are imaginary SWF files that, for example, you could load when a user selects a city from the combo box.
5. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code:

```
function change(evt){  
    trace(evt.target.selectedItem.label);  
}  
comboBox.addEventListener("change", this);
```

The last line of code adds a `change` event handler to the `ComboBox` instance. For more information, see [ComboBox.change](#).

Customizing the ComboBox component

You can transform a `ComboBox` component horizontally and vertically while authoring. While authoring, select the component on the Stage and use the Free Transform tool or any of the `Modify > Transform` commands.

If text is too long to fit in the combo box, the text is clipped to fit. You must resize the combo box while authoring to fit the label text.

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

Using styles with the ComboBox component

You can set style properties to change the appearance of a `ComboBox` component. If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in Flash Help.

The combo box has two unique styles: `openDuration` and `openEasing`. Other styles are passed to the button, text box, and drop-down list of the combo box through those individual components, as follows:

- The button is a `Button` instance and uses its styles. (See “Using styles with the Button component” on page 71.)
- The text is a `TextInput` instance and uses its styles. (See “Using styles with the TextInput component” on page 447.)
- The drop-down list is an `List` instance and uses its styles. (See “Using styles with the List component” on page 281.)

A ComboBox component uses the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>backgroundColor</code>	Both	The background color. The default color is white.
<code>border styles</code>	Both	The Button subcomponent uses two <code>RectBorder</code> instances for its borders and responds to the styles defined on that class. See "RectBorder class" in Flash Help. In the Halo theme, the ComboBox component uses a custom rounded border for the collapsed portion of the ComboBox. The colors of this portion of the ComboBox can be modified only through skinning. See "Using skins with the ComboBox component" on page 101 .
<code>color</code>	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
<code>embedFonts</code>	Both	Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is "_sans".
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either "normal" or "italic". The default value is "normal".
<code>fontWeight</code>	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return "none".
<code>textAlign</code>	Both	The text alignment: either "left", "right", or "center". The default value is "left".
<code>textDecoration</code>	Both	The text decoration: either "none" or "underline". The default value is "none".
<code>openDuration</code>	Both	The duration, in milliseconds, of the transition animation. The default value is 250.
<code>openEasing</code>	Both	A reference to a tweening function that controls the animation. Defaults to sine in/out. For more information, see "Customizing component animations" in Flash Help.

The following example demonstrates how to use List styles to control the behavior of the drop-down portion of a ComboBox component.

```
// comboBox is an instance of the ComboBox component on Stage  
comboBox.setStyle("alternatingRowColors", [0xFFFFFFFF, 0xBFBBFB]);
```

Using skins with the ComboBox component

The ComboBox component uses symbols in the library to represent the button states and has skin variables for the down arrow. These skins are located in the Flash UI Components 2/Themes/MMDefault/ComboBox Assets/States folder of the HaloTheme.fla and SampleTheme.fla files. The information below describes these skins and provides steps for customizing them.

The ComboBox component also uses scroll bar skins for the drop-down list's scroll bar and two RectBorder class instances for the border around the text input and drop-down list. For information on customizing these skins, see [“Using skins with the ScrollPane component” on page 424](#) and “RectBorder class” in Flash Help. For more information on the methods available to skin components, see “About skinning components” in Flash Help.

A ComboBox component uses the following skin properties:

Property	Description
ComboDownArrowDisabledName	The down arrow's disabled state. The default is <code>ComboDownArrowDisabled</code> .
ComboDownArrowDownName	The down arrow's down state. The default is <code>ComboDownArrowDown</code> .
ComboDownArrowUpName	The down arrow's up state. The default is <code>ComboDownArrowOver</code> .
ComboDownArrowOverName	The down arrow's over state. The default is <code>ComboDownArrowUp</code> .

To create movie clip symbols for ComboBox skins:

1. Create a new FLA file.
2. Select File > Import > Open External Library, and select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in Flash Help.
3. In the theme's Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the ComboBox Assets folder to the library for your document.
4. Expand the ComboBox Assets/States folder in the library of your document.
5. Open the symbols you want to customize for editing.
For example, open the ComboDownArrowDisabled symbol.
6. Customize the symbol as desired.
For example, change the inner white square to a light gray.
7. Repeat steps 5-6 for all symbols you want to customize.
8. Click the Back button to return to the main Timeline.
9. Drag a ComboBox component to the Stage.

10. Set the `ComboBox` instance properties as desired.

For this example, use `ActionScript` to set the `ComboBox` to disabled.

11. Select `Control > Test Movie`.

ComboBox class

Inheritance `MovieClip` > [UIObject class](#) > [UIComponent class](#) > `ComboBase` > `ComboBox`

ActionScript Class Name `mx.controls.ComboBox`

The `ComboBox` component combines three separate subcomponents: `Button`, `TextInput`, and `List`. Most of the methods, properties, and events of each subcomponent are available directly from the `ComboBox` component and are listed in the summary tables for the `ComboBox` class.

The drop-down list in a combo box is provided either as an array or as a data provider. If you use a data provider, the list changes at runtime. You can change the source of the `ComboBox` data dynamically by switching to a new array or data provider.

Items in a combo box list are indexed by position, starting with the number 0. An item can be one of the following:

- A primitive data type.
- An object that contains a `label` property and a `data` property

Note: An object may use the `ComboBox.labelFunction` or `ComboBox.labelField` property to determine the `label` property.

If the item is a primitive data type other than `String`, it is converted to a string. If an item is an object, the `label` property must be a string and the `data` property can be any `ActionScript` value.

`ComboBox` methods to which you supply items have two parameters, `label` and `data`, that refer to the properties above. Methods that return an item return it as an object.

A combo box defers the instantiation of its drop-down list until a user interacts with it. Therefore, a combo box may appear to respond slowly on first use.

Use the following code to programmatically access the `ComboBox` component's drop-down list and override the delay:

```
var foo = myComboBox.dropdown;
```

Accessing the drop-down list may cause a pause in the application. This may occur when the user first interacts with the combo box, or when the above code runs.

Method summary for the ComboBox class

The following table lists methods of the `ComboBox` class.

Method	Description
<code>ComboBox.addItem()</code>	Adds an item to the end of the list.
<code>ComboBox.addItemAt()</code>	Adds an item to the end of the list at the specified index.
<code>ComboBox.close()</code>	Closes the drop-down list.

Method	Description
<code>ComboBox.getItemAt()</code>	Returns the item at the specified index.
<code>ComboBox.open()</code>	Opens the drop-down list.
<code>ComboBox.removeAll()</code>	Removes all items in the list.
<code>ComboBox.removeItemAt()</code>	Removes an item from the list at the specified location.
<code>ComboBox.replaceItemAt()</code>	Replaces the content of the item at the specified index.
<code>ComboBox.sortItems()</code>	Sorts the list using a compare function.
<code>ComboBox.sortItemsBy()</code>	Sorts the list using a field of each item.

Methods inherited from the UIObject class

The following table lists the methods the ComboBox class inherits from the UIObject class. When calling these methods from the ComboBox object, use the form *comboBoxInstance.methodName*.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it will be redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the ComboBox class inherits from the UIComponent class. When calling these methods from the ComboBox object, use the form *comboBoxInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the ComboBox class

The following table lists properties of the ComboBox class.

Property	Description
<code>ComboBox.dataProvider</code>	The data model for the items in the list.
<code>ComboBox.dropdown</code>	Returns a reference to the List component contained by the combo box.
<code>ComboBox.dropdownWidth</code>	The width of the drop-down list, in pixels.
<code>ComboBox.editable</code>	Indicates whether a combo box is editable.
<code>ComboBox.labelField</code>	Indicates which data field to use as the label for the drop-down list.
<code>ComboBox.labelFunction</code>	Specifies a function to compute the label field for the drop-down list.
<code>ComboBox.length</code>	Read-only; the length of the drop-down list.
<code>ComboBox.restrict</code>	The set of characters that a user can enter in the text field of a combo box.
<code>ComboBox.rowCount</code>	The maximum number of list items to display at one time.
<code>ComboBox.selectedIndex</code>	The index of the selected item in the drop-down list.
<code>ComboBox.selectedItem</code>	The value of the selected item in the drop-down list.
<code>ComboBox.text</code>	The string of text in the text box.
<code>ComboBox.textField</code>	A reference to the TextInput component in the combo box.
<code>ComboBox.value</code>	The value of the text box (editable) or drop-down list (static).

Properties inherited from the UIObject class

The following table lists the properties the ComboBox class inherits from the UIObject class. When accessing these properties from the ComboBox object, use the form *comboBoxInstance.propertyName*.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.

Property	Description
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `ComboBox` class inherits from the `UIComponent` class. When accessing these properties from the `ComboBox` object, use the form `comboBoxInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the `ComboBox` class

The following table lists events of the `ComboBox` class.

Event	Description
<code>ComboBox.change</code>	Broadcast when the value of the combo box changes as a result of user interaction.
<code>ComboBox.close</code>	Broadcast when the list of the combo box begins to retract.
<code>ComboBox.enter</code>	Broadcast when the Enter key is pressed.
<code>ComboBox.itemRollOut</code>	Broadcast when the pointer rolls off a drop-down list item.
<code>ComboBox.itemRollOver</code>	Broadcast when a drop-down list item is rolled over.
<code>ComboBox.open</code>	Broadcast when the drop-down list begins to open.
<code>ComboBox.scroll</code>	Broadcast when the drop-down list is scrolled.

Events inherited from the `UIObject` class

The following table lists the events the `ComboBox` class inherits from the `UIObject` class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.

Event	Description
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the ComboBox class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

ComboBox.addItem()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```

comboBoxInstance.addItem(label[, data])
comboBoxInstance.addItem({label:label[, data:data])
comboBoxInstance.addItem(obj);

```

Parameters

label A string that indicates the label for the new item.

data The data for the item; it can be of any data type. This parameter is optional.

obj An object with a `label` property and an optional `data` property.

Returns

The index at which the item was added.

Description

Method; adds a new item to the end of the list.

Example

The following code adds an item to the `myComboBox` instance:

```
myComboBox.addItem("this is an Item");
```

ComboBox.addItemAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
comboBoxInstance.addItemAt(index, label[, data])
comboBoxInstance.addItemAt(index, {label:label[, data:data]})
comboBoxInstance.addItemAt(index, obj);
```

Parameters

index A number 0 or greater that indicates the position at which to insert the item (the index of the new item).

label A string that indicates the label for the new item.

data The data for the item; it can be of any data type. This parameter is optional.

obj An object with `label` and `data` properties.

Returns

The index at which the item was added.

Description

Method; adds a new item to the end of the list at the index specified by the *index* parameter. Indices greater than `ComboBox.length` are ignored.

Example

The following code inserts an item at index 3, which is the fourth position in the combo box list (0 is the first position):

```
myBox.addItemAt(3, "this is the fourth Item");
```

ComboBox.change

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(change){
    // your code here
}
```

Usage 2:

```
listenerObject = new Object();
listenerObject.change = function(eventObject){
    // your code here
}
comboBoxInstance.addEventListener("change", listenerObject)
```

Description

Event; broadcast to all registered listeners when the `ComboBox.selectedIndex` or `ComboBox.selectedItem` property changes as a result of user interaction.

The first usage example uses an `on()` handler and must be attached directly to a `ComboBox` instance. The keyword `this`, used in an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `ComboBox` instance `myBox`, sends “_level0.myBox” to the Output panel:

```
on(change){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (`comboBoxInstance`) dispatches an event (in this case, `change`) and the event is handled by a function, also called a *handler*, on a listener object (`listenerObject`) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (`eventObject`) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call `addEventListener()` (see `EventDispatcher.addEventListener()` in Flash Help) on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following example sends the instance name of the component that generated the `change` event to the Output panel:

```
form.change = function(eventObj){
    trace("Value changed to " + eventObj.target.value);
}
myCombo.addEventListener("change", form);
```

See also

`EventDispatcher.addEventListener()` in Flash Help

ComboBox.close()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
myComboBox.close()
```

Parameters

None.

Returns

Nothing.

Description

Method; closes the drop-down list.

Example

The following example closes the drop-down list of the `myBox` combo box:

```
myBox.close();
```

See also

[ComboBox.open\(\)](#)

ComboBox.close

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(close){  
    // your code here  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.close = function(eventObject){  
    // your code here  
}  
comboBoxInstance.addEventListener("close", listenerObject)
```

Description

Event; broadcast to all registered listeners when the drop-down list of the combo box is fully retracted.

The first usage example uses an `on()` handler and must be attached directly to a `ComboBox` instance. The keyword `this`, used in an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `ComboBox` instance `myBox`, sends “_level0.myBox” to the Output panel:

```
on(close){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*comboBoxInstance*) dispatches an event (in this case, `close`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following example sends a message to the Output panel when the drop-down list begins to close:

```
form.close = function(){
    trace("The combo box has closed");
}
myCombo.addEventListener("close", form);
```

See also

`EventDispatcher.addEventListener()` in Flash Help

ComboBox.dataProvider

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

comboBoxInstance.dataProvider

Description

Property; the data model for items viewed in a list. The value of this property can be an array or any object that implements the `DataProvider` API. The default value is `[]`. The `List` component and the `ComboBox` component share the `dataProvider` property, and changes to this property are immediately available to both components.

The `List` component, like other data-aware components, adds methods to the `Array` object's prototype so that they conform to the `DataProvider` API (see `DataProvider.as` for details). Therefore, any array that exists at the same time as a list automatically has all the methods (`addItem()`, `getItemAt()`, and so on) needed for it to be the model of a list, and can be used to broadcast model changes to multiple components.

If the array contains objects, the `labelField` or `labelFunction` property is accessed to determine what parts of the item to display. The default value is `"label"`, so if such a field exists, it is chosen for display; if not, a comma-separated list of all fields is displayed.

Note: If the array contains strings at each index, and not objects, the list is not able to sort the items and maintain the selection state. Any sorting will cause the selection to be lost.

Any instance that implements the `DataProvider` API is eligible as a data provider for a `List` component. This includes `Flash Remoting RecordSet` objects, `Firefly DataSet` components, and so on.

Example

This example uses an array of strings to populate the drop-down list:

```
comboBox.dataProvider = ["Ground Shipping", "2nd Day Air", "Next Day Air"];
```

This example creates a data provider array and assigns it to the `dataProvider` property:

```
myDP = new Array();
list.dataProvider = myDP;

for (var i=0; i<accounts.length; i++) {
    // these changes to the DataProvider will be broadcast to the list
    myDP.addItem({label: accounts[i].name,
                  data: accounts[i].accountID});
}
```

ComboBox.dropdown

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
myComboBox.dropdown
```

Description

Property (read-only); returns a reference to the list contained by the combo box. The List subcomponent isn't instantiated in the combo box until it needs to be displayed. However, when you access the `dropdown` property, the list is created.

See also

[ComboBox.dropdownWidth](#)

ComboBox.dropdownWidth

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
myComboBox.dropdownWidth
```

Description

Property; the width limit of the drop-down list, in pixels. The default value is the width of the ComboBox component (the TextInput instance plus the SimpleButton instance).

Example

The following code sets `dropdownWidth` to 150 pixels:

```
myComboBox.dropdownWidth = 150;
```

See also

[ComboBox.dropdown](#)

ComboBox.editable

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
myComboBox.editable
```

Description

Property; indicates whether the combo box is editable (`true`) or not (`false`). In an editable combo box, a user can enter values into the text box that do not appear in the drop-down list. If a combo box is not editable, you cannot enter text into the text box. The text box displays the text of the item in the list. The default value is `false`.

Making a combo box editable clears the combo box text field. It also sets the selected index (and item) to undefined. To make a combo box editable and still retain the selected item, use the following code:

```
var ix = myComboBox.selectedIndex;
myComboBox.editable = true; // clears the text field
myComboBox.selectedIndex = ix; // copies the label back into the text field
```

Example

The following code makes `myComboBox` editable:

```
myComboBox.editable = true;
```

ComboBox.enter

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(enter){
    // your code here
}
```

Usage 2:

```
listenerObject = new Object();
listenerObject.enter = function(eventObject){
    // your code here
}
comboBoxInstance.addEventListener("enter", listenerObject)
```

Description

Event; broadcast to all registered listeners when the user presses the Enter key in the text box. This event is a `TextInput` event that is broadcast only from editable combo boxes. For more information, see [TextInput.enter](#).

The first usage example uses an `on()` handler and must be attached directly to a `ComboBox` instance. The keyword `this`, used in an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `ComboBox` instance `myBox`, sends “`_level0.myBox`” to the Output panel:

```
on(enter){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*comboBoxInstance*) dispatches an event (in this case, *enter*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following example sends a message to the Output panel when the drop-down list begins to close:

```
form.enter = function(){
    trace("The combo box enter event was triggered");
}
myCombo.addEventListener("enter", form);
```

See also

`EventDispatcher.addEventListener()` in Flash Help

ComboBox.getItemAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
comboBoxInstance.getItemAt(index)
```

Parameters

index The index of the item to retrieve. The index must be a number greater than or equal to 0, and less than the value of `ComboBox.length`.

Returns

The indexed item object or value. The value is undefined if the index is out of range.

Description

Method; retrieves the item at a specified index.

Example

The following code displays the item at index position 4:

```
trace(myBox.getItemAt(4).label);
```

ComboBox.itemRollOut

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(itemRollOut){  
    // your code here  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.itemRollOut = function(eventObject){  
    // your code here  
}  
comboBoxInstance.addEventListener("itemRollOut", listenerObject)
```

Event object

In addition to the standard properties of the event object, the `itemRollOut` event has an `index` property. The index is the number of the item that the pointer rolled out of.

Description

Event; broadcast to all registered listeners when the pointer rolls out of drop-down list items. This is a List event that is broadcast from a combo box. For more information, see [List.itemRollOut](#).

The first usage example uses an `on()` handler and must be attached directly to a `ComboBox` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `ComboBox` instance `myBox`, sends “_level0.myBox” to the Output panel:

```
on(itemRollOut){  
    trace(this);  
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*comboBoxInstance*) dispatches an event (in this case, *itemRollOut*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see “EventDispatcher class” in Flash Help.

Finally, you call the `addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

Example

The following example sends a message to the Output panel that indicates the index of the item that the pointer rolled out of:

```
form.itemRollOut = function (eventObj) {
    trace("Item #" + eventObj.index + " has been rolled out of.");
}
myCombo.addEventListener("itemRollOut", form);
```

See also

[ComboBox.itemRollOver](#), [EventDispatcher.addEventListener\(\)](#) in Flash Help

ComboBox.itemRollOver

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(itemRollOver){
    // your code here
}
```

Usage 2:

```
listenerObject = new Object();
listenerObject.itemRollOver = function(eventObject){
    // your code here
}
comboBoxInstance.addEventListener("itemRollOver", listenerObject)
```

Event object

In addition to the standard properties of the event object, the *itemRollOver* event has an *index* property. The index is the number of the item that the pointer rolled over.

Description

Event; broadcast to all registered listeners when the mouse pointer rolls over drop-down list items. This is a List event that is broadcast from a combo box. For more information, see [List.itemRollOver](#).

The first usage example uses an `on()` handler and must be attached directly to a `ComboBox` instance. The keyword `this`, used in an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `ComboBox` instance `myBox`, sends “_level0.myBox” to the Output panel:

```
on(itemRollOver){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*comboBoxInstance*) dispatches an event (in this case, *itemRollOver*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see “EventDispatcher class” in Flash Help.

Finally, you call the `addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

Example

The following example sends a message to the Output panel that indicates the index of the item that the pointer rolled over:

```
form.itemRollOver = function (eventObj) {
    trace("Item #" + eventObj.index + " has been rolled over.");
}
myCombo.addEventListener("itemRollOver", form);
```

See also

[ComboBox.itemRollOut](#), [EventDispatcher.addEventListener\(\)](#) in Flash Help

ComboBox.labelField

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
myComboBox.labelField
```

Description

Property; the name of the field in `dataProvider` array objects to use as the label field. This is a property of the List component that is available from a ComboBox component instance. For more information, see [List.labelField](#).

The default value is undefined.

Example

The following example sets the `dataProvider` property to an array of strings and sets the `labelField` property to indicate that the `name` field should be used as the label for the drop-down list:

```
myComboBox.dataProvider = [
    {name:"Gary", gender:"male"},
    {name:"Susan", gender:"female"} ];

myComboBox.labelField = "name";
```

See also

[List.labelFunction](#)

ComboBox.labelFunction

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

myComboBox.labelFunction

Description

Property; a function that computes the label of a data provider item. You must define the function. The default value is undefined.

Example

The following example creates a data provider and then defines a function to specify what to use as the label in the drop-down list:

```
myComboBox.dataProvider = [
    {firstName:"Nigel", lastName:"Pegg", age:"really young"},
    {firstName:"Gary", lastName:"Grossman", age:"young"},
    {firstName:"Chris", lastName:"Walcott", age:"old"},
    {firstName:"Greg", lastName:"Yachuk", age:"really old"} ];

myComboBox.labelFunction = function(itemObj){
    return (itemObj.lastName + ", " + itemObj.firstName);
}
```

See also

[List.labelField](#)

ComboBox.length

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
myComboBox.length
```

Description

Property (read-only); the length of the drop-down list. This is a property of the List component that is available from a ComboBox instance. For more information, see [List.length](#). The default value is 0.

Example

The following example stores the value of `length` to a variable:

```
dropdownItemCount = myBox.length;
```

ComboBox.open()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
myComboBox.open()
```

Parameters

None.

Returns

Nothing.

Description

Method; opens the drop-down list.

Example

The following code opens the drop-down list for the `combo1` instance:

```
combo1.open();
```

See also

`ComboBox.close()`

ComboBox.open

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(open){
    // your code here
}
```

Usage 2:

```
listenerObject = new Object();
listenerObject.open = function(eventObject){
    // your code here
}
comboBoxInstance.addEventListener("open", listenerObject)
```

Description

Event; broadcast to all registered listeners when the drop-down list is completely open.

The first usage example uses an `on()` handler and must be attached directly to a `ComboBox` instance. The keyword `this`, used in an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `ComboBox` instance `myBox`, sends “_level0.myBox” to the Output panel:

```
on(open){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*comboBoxInstance*) dispatches an event (in this case, `open`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see “EventDispatcher class” in Flash Help.

Finally, you call the `addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

Example

The following example sends a message to the Output panel:

```
function open(evt) {  
    trace("The combo box has opened with text " + evt.target.text);  
}  
myBox.addEventListener("open", this);
```

See also

[ComboBox.close](#), [EventDispatcher.addEventListener\(\)](#) in Flash Help

ComboBox.removeAll()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
comboBoxInstance.removeAll();
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all items in the list. This is a method of the List component that is available from an instance of the ComboBox component.

Example

The following code clears the list:

```
myCombo.removeAll();
```

See also

[ComboBox.removeItemAt\(\)](#), [ComboBox.replaceItemAt\(\)](#)

ComboBox.removeItemAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.removeItemAt(index)
```

Parameters

index A number that indicates the position of the item to remove. The index is zero-based.

Returns

An object; the removed item (undefined if no item exists).

Description

Method; removes the item at the specified index position. The list indices after the index indicated by the *index* parameter collapse by one. This is a method of the List component that is available from an instance of the ComboBox component.

Example

The following code removes the item at index position 3:

```
myCombo.removeItemAt(3);
```

See also

[ComboBox.removeAll\(\)](#), [ComboBox.replaceItemAt\(\)](#)

ComboBox.replaceItemAt()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
comboBoxInstance.replaceItemAt(index, label[, data])  
comboBoxInstance.replaceItemAt(index, {label:label[, data:data]})  
comboBoxInstance.replaceItemAt(index, obj);
```

Parameters

index A number 0 or greater that indicates the position at which to insert the item (the index of the new item).

label A string that indicates the label for the new item.

data The data for the item. This parameter is optional.

obj An object with `label` and `data` properties.

Returns

Nothing.

Description

Method; replaces the content of the item at the specified index. This is a method of the List component that is available from the ComboBox component.

Example

The following example changes the third index position:

```
myCombo.replaceItemAt(3, "new label");
```

See also

[ComboBox.removeAll\(\)](#), [ComboBox.removeItemAt\(\)](#)

ComboBox.restrict

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
comboBoxInstance.restrict
```

Description

Property; indicates the set of characters that a user can enter in the text field of a combo box. The default value is undefined. If this property is null or an empty string (""), a user can enter any character. If this property is a string of characters, the user can enter only characters in the string; the string is scanned from left to right. You can specify a range by using a dash (-).

If the string begins with a caret (^), all characters that follow the caret are considered unacceptable characters. If the string does not begin with a caret, the characters in the string are considered acceptable.

You can use the backslash (\) to enter a hyphen (-), caret (^), or backslash (\) character, as shown here:

```
\^  
\-  
\\
```

When you enter a backslash in the Actions panel within double quotation marks, it has a special meaning for the Actions panel's double-quote interpreter. It signifies that the character following the backslash should be treated "as is." For example, you could use the following code to enter a single quotation mark:

```
var leftQuote = "'";
```

The Actions panel's restrict interpreter also uses the backslash as an escape character. Therefore, you may think that the following should work:

```
myText.restrict = "0-9\\-\\^\\\"";
```

However, since this expression is surrounded by double quotation marks, the value `0-9-^\` is sent to the restrict interpreter, and the restrict interpreter doesn't understand this value.

Because you must enter this expression within double quotation marks, you must not only provide the expression for the restrict interpreter, but you must also escape the expression so that it will be read correctly by the Actions panel's built-in interpreter for double quotation marks. To send the value `0-9\-\^\` to the restrict interpreter, you must enter the following code:

```
myCombo.restrict = "0-9\-\^\^\^\^\";
```

The `restrict` property restricts only user interaction; a script may put any text into the text field. This property does not synchronize with the Embed Font Outlines check boxes in the Property inspector.

Example

In the following example, the first line of code limits the text field to uppercase letters, numbers, and spaces. The second line of code allows all characters except lowercase letters.

```
my_combo.restrict = "A-Z 0-9";  
my_combo.restrict = "^a-z";
```

The following code allows a user to enter the characters `"0 1 2 3 4 5 6 7 8 9 - ^ \"` in the instance `myCombo`. You must use a double backslash to escape the characters `-`, `^`, and `\`. The first `\` escapes the double quotation marks, and the second `\` tells the interpreter that the next character should not be treated as a special character.

```
myCombo.restrict = "0-9\-\^\^\^\^\";
```

ComboBox.rowCount

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
myComboBox.rowCount
```

Description

Property; the maximum number of rows visible in the drop-down list. The default value is 5.

If the number of items in the drop-down list is greater than the `rowCount` property, the list resizes and a scroll bar is displayed if necessary. If the drop-down list contains fewer items than the `rowCount` property, it resizes to the number of items in the list.

This behavior differs from the List component, which always shows the number of rows specified by its `rowCount` property, even if some empty space is shown.

If the value is negative or fractional, the behavior is undefined.

Example

The following example specifies that the combo box should have 20 or fewer rows visible:

```
myComboBox.rowCount = 20;
```

ComboBox.scroll

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(scroll){  
    // your code here  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.scroll = function(eventObject){  
    // your code here  
}  
comboBoxInstance.addEventListener("scroll", listenerObject)
```

Event object

Along with the standard event object properties, the scroll event has one additional property, `direction`. It is a string with two possible values, "horizontal" or "vertical". For a `ComboBox` scroll event, the value is always "vertical".

Description

Event; broadcast to all registered listeners when the drop-down list is scrolled. This is a List component event that is available to the `ComboBox` component.

The first usage example uses an `on()` handler and must be attached directly to a `ComboBox` instance. The keyword `this`, used in an `on()` handler attached to a component, refers to the instance. For example, the following code, attached to the `ComboBox` component instance `myBox`, sends “_level0.myBox” to the Output panel:

```
on(scroll){  
    trace(this);  
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*comboBoxInstance*) dispatches an event (in this case, `scroll`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see “EventDispatcher class” in Flash Help.

Finally, you call the `addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

Example

The following example sends a message to the Output panel that indicates the index of the item that the list scrolled to:

```
form.scroll = function (eventObj) {
    trace("The list had been scrolled to item # " + eventObj.target.vPosition);
}
myCombo.addEventListener("scroll", form);
```

See also

`EventDispatcher.addEventListener()` in Flash Help

ComboBox.selectedIndex

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

myComboBox.selectedIndex

Description

Property; the index number of the selected item in the drop-down list. The default value is 0. Assigning this property clears the current selection, selects the indicated item, and displays the label of that item in the combo box's text box.

If you assign an out-of-range value to this property, Flash ignores it. Entering text into the text field of an editable combo box sets `selectedIndex` to `undefined`.

Example

The following code selects the last item in the list:

```
myComboBox.selectedIndex = myComboBox.length-1;
```

See also

[ComboBox.selectedItem](#)

ComboBox.selectedItem

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
myComboBox.selectedItem
```

Description

Property; the value of the selected item in the drop-down list.

If the combo box is editable, `selectedItem` returns `undefined` if the user enters any text in the text box. The property only has a value if you select an item from the drop-down list or set the value using `ActionScript`. If the combo box is static, the value of `selectedItem` is always valid; it returns `undefined` if there are no items in the list.

Example

The following example shows `selectedItem` if the data provider contains primitive types:

```
var item = myComboBox.selectedItem;  
trace("You selected the item " + item);
```

The following example shows `selectedItem` if the data provider contains objects with `label` and `data` properties:

```
var obj = myComboBox.selectedItem;  
trace("You have selected the color named: " + obj.label);  
trace("The hex value of this color is: " + obj.data);
```

See also

[ComboBox.dataProvider](#), [ComboBox.selectedIndex](#)

ComboBox.sortItems()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myComboBox.sortItems([compareFunc], [optionsFlag])
```

Parameters

compareFunc A reference to a function that compares two items to determine their sort order. For details, see `Array.sort()` in *Flash ActionScript Language Reference*. This parameter is optional.

optionsFlag Lets you perform multiple sorts of different types on a single array without having to replicate the entire array or re-sort it repeatedly. This parameter is optional.

The following are possible values for *optionsFlag*:

- `Array.DESENDING`, which sorts highest to lowest.
- `Array.CASEINSENSITIVE`, which sorts without regard to case.
- `Array.NUMERIC`, which sorts numerically if the two elements being compared are numbers. If they aren't numbers, use a string comparison (which can be case-insensitive if that flag is specified).
- `Array.UNIQUESORT`, which returns an error code (0) instead of a sorted array if two objects in the array are identical or have identical sort fields.
- `Array.RETURNINDEXEDARRAY`, which returns an integer index array that is the result of the sort. For example, the following array would return the second line of code and the array would remain unchanged:

```
["a", "d", "c", "b"]  
[0, 3, 2, 1]
```

You can combine these options into one value. For example, the following code combines options 3 and 1:

```
array.sort (Array.NUMERIC | Array.DESENDING)
```

Returns

Nothing.

Description

Method; sorts the items in the combo box according to the specified compare function or according to the specified sort options.

Example

This example sorts according to uppercase labels. The items `a` and `b` are passed to the function and contain `label` and `data` fields:

```
myComboBox.sortItems(upperCaseFunc);  
function upperCaseFunc(a,b){  
    return a.label.toUpperCase() > b.label.toUpperCase();  
}
```

The following example uses the `upperCaseFunc()` function defined above, along with the *optionsFlag* parameter to sort the elements of a `ComboBox` instance named `myComboBox`:

```
myComboBox.addItem("Mercury");  
myComboBox.addItem("Venus");  
myComboBox.addItem("Earth");  
myComboBox.addItem("planet");
```

```
myComboBox.sortItems(upperCaseFunc, Array.DESENDING);  
// The resulting sort order of myComboBox will be:  
// Venus  
// planet  
// Mercury  
// Earth
```

ComboBox.sortItemsBy()

Availability

Flash Player 7.

Edition

Flash MX Professional 2004.

Usage

```
myComboBox.sortItemsBy(fieldName, order [optionsFlag])
```

Parameters

fieldName A string that specifies the name of the field to use for sorting. This value is usually "label" or "data".

order A string that specifies whether to sort the items in ascending order ("ASC") or descending order ("DESC").

optionsFlag Lets you perform multiple sorts of different types on a single array without having to replicate the entire array or re-sort it repeatedly. This parameter is optional, but if used, should replace the *order* parameter.

The following are possible values for *optionsFlag*:

- `Array.DESENDING`, which sorts highest to lowest.
- `Array.CASEINSENSITIVE`, which sorts without regard to case.
- `Array.NUMERIC`, which sorts numerically if the two elements being compared are numbers. If they aren't numbers, use a string comparison (which can be case-insensitive if that flag is specified).
- `Array.UNIQUESORT`, which returns an error code (0) instead of a sorted array if two objects in the array are identical or have identical sort fields.
- `Array.RETURNINDEXEDARRAY`, which returns an integer index array that is the result of the sort. For example, the following array would return the second line of code and the array would remain unchanged:

```
["a", "d", "c", "b"]  
[0, 3, 2, 1]
```

You can combine these options into one value. For example, the following code combines options 3 and 1:

```
array.sort (Array.NUMERIC | Array.DESENDING)
```

Returns

Nothing.

Description

Method; sorts the items in the combo box alphabetically or numerically, in the specified order, using the specified field name. If the *fieldName* items are a combination of text strings and integers, the integer items are listed first. The *fieldName* parameter is usually "label" or "data", but advanced programmers may specify any primitive value. If you want, you can use the *optionsFlag* parameter to specify a sorting style.

Example

The following examples are based on a ComboBox instance named `myComboBox`, which contains four elements labeled "Apples", "Bananas", "cherries", and "Grapes":

```
// First, populate the ComboBox with the elements.
myComboBox.addItem("Bananas");
myComboBox.addItem("Apples");
myComboBox.addItem("cherries");
myComboBox.addItem("Grapes");

// The following statement sorts using the order parameter set to "ASC",
// and results in a sort that places "cherries" at the bottom of the list
// because the sort is case-sensitive.
myDP.sortItemsBy("label", "ASC");
// resulting order: Apples, Bananas, Grapes, cherries

// The following statement sorts using the order parameter set to "DESC",
// and results in a sort that places "cherries" at the top of the list
// because the sort is case-sensitive.
myComboBox.sortItemsBy("label", "DESC");
// resulting order: cherries, Grapes, Bananas, Apples

// The following statement sorts using the optionsFlag parameter set to
// Array.CASEINSENSITIVE. Note that an ascending sort is the default setting.
myComboBox.sortItemsBy("label", Array.CASEINSENSITIVE);
// resulting order: Apples, Bananas, cherries, Grapes

// The following statement sorts using the optionsFlag parameter set to
// Array.CASEINSENSITIVE | Array.DESCEDING.
myComboBox.sortItemsBy("label", Array.CASEINSENSITIVE | Array.DESCEDING);
// resulting order: Grapes, cherries, Bananas, Apples
```

ComboBox.text

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

myComboBox.text

Description

Property; the text of the text box. You can get and set this value for editable combo boxes. For static combo boxes, the value is read-only.

Example

The following example sets the current `text` value of an editable combo box:

```
myComboBox.text = "California";
```

ComboBox.textField**Availability**

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

myComboBox.textField

Description

Property (read-only); a reference to the `TextInput` component contained by the `ComboBox` component.

This property lets you access the underlying `TextInput` component so that you can manipulate it. For example, you might want to change the selection of the text box or restrict the characters that can be entered in it.

Example

The following code restricts the text box of `myComboBox` so that it only accept numbers:

```
myComboBox.textField.restrict = "0-9";
```

ComboBox.value**Availability**

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

myComboBox.value

Description

Property (read-only); if the combo box is editable, `value` returns the value of the text box. If the combo box is static, `value` returns the value of the drop-down list. The value of the drop-down list is the `data` field, or, if the `data` field doesn't exist, the `label` field.

Example

The following example puts the data into the combo box by setting the `dataProvider` property. It then displays the `value` in the Output panel. Finally, it selects "California" and displays it in the text box.

```
cb.dataProvider = [
    {label:"Alaska", data:"AZ"},
    {label:"California", data:"CA"},
    {label:"Washington", data:"WA"}];
cb.editable = true;
cb.selectedIndex = 1;
trace('Editable value is "California": '+ cb.value);
cb.editable = false;
cb.selectedIndex = 1;
trace('Non-editable value is "CA": '+ cb.value);
```

DataGrid component

The DataGrid component lets you create powerful data-enabled displays and applications. You can use the DataGrid component to instantiate a recordset (retrieved from a database query in Macromedia ColdFusion, Java, or .Net) using Macromedia Flash Remoting and display it in columns. You can also use data from a data set or from an array to fill a DataGrid component. The version 2 DataGrid component has been improved to include horizontal scrolling, better event support (including event support for editable cells), enhanced sorting capabilities, and performance optimizations.

You can resize and customize characteristics such as the font, color, and 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 use scroll bars to move through data in a grid; you can also turn off scroll bars and use the DataGrid methods to create a page view style display.

When you add the DataGrid component to an application, you can use the Accessibility panel to make the component accessible to screen readers. First, you must add the following line of code to enable accessibility for the DataGrid component:

```
mx.accessibility.DataGridAccImpl.enableAccessibility();
```

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

Interacting with the DataGrid component

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

If `DataGrid.sortableColumns` and `DataGridColumn.sortOnHeaderRelease` are both `true`, clicking in a column header causes the grid to sort based on the column's cell values.

If `DataGrid.resizableColumns` is `true`, clicking in the area between columns lets you resize columns.

Clicking in an editable cell sends focus to that cell; clicking a non-editable 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`.

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.
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.
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	Moves focus to the previous item. When the Tab key is pressed, focus cycles from the last column in the grid to the first column on the next line. When Shift+Tab is pressed, cycling is reversed. All the text in the focused cell is selected.

Using the `DataGrid` component

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 a database query (or other 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

Understanding the design of the DataGrid component

The DataGrid component extends the [List component](#). When you design an application with the DataGrid component, it is helpful to understand how the List class underlying it was designed. The following are some fundamental assumptions and requirements that Macromedia used when developing the List class:

- Keep it small, fast, and simple.

Don't make something more complicated than absolutely necessary. This was the prime design directive. Most of the requirements listed below are based on this directive.

- Lists have uniform row heights.

Every row must be the same height; the height can be set during authoring or at runtime.

- Lists must scale to thousands of records.

- Lists don't measure text.

This creates a horizontal scrolling issue for List and Tree components; for more information, see ["Understanding the design of the List component" on page 278](#). The DataGrid component, however, supports "auto" as an `hScrollPolicy` value, because it measures columns (which are the same width per item), not text.

The fact that lists don't measure text explains why lists have uniform row heights. Sizing individual rows to fit text would require intensive measuring. For example, if you wanted to accurately show the scroll bars on a list with nonuniform row height, you'd need to premeasure every row.

- Lists perform worse as a function of their visible rows.

Although lists can display 5000 records, they can't render 5000 records at once. The more visible rows (specified by the `rowCount` property) you have on the Stage, the more work the list must do to render. Limiting the number of visible rows, if at all possible, is the best solution.

- Lists aren't tables.

DataGrid components are intended to provide an interface for many records. They're not designed to display complete information; they're designed to display enough information so that users can drill down to see more. The message view in Microsoft Outlook is a prime example. You don't read the entire e-mail in the grid; the message would be difficult to read and the client would perform terribly. Outlook displays enough information so that a user can drill into the post to see the details.

Understanding the DataGrid component: data model and view

Conceptually, the DataGrid component is composed of a data model and a view that displays the data. The data model consists of three main parts:

- **DataProvider**

This is a list of items with which to fill the data grid. Any array in the same frame as a DataGrid component is automatically given methods (from the DataProvider API) that let you manipulate data and broadcast changes to multiple views. Any object that implements the DataProvider API can be assigned to the `DataGrid.dataProvider` property (including recordsets, data sets, and so on). The following code creates a data provider called `myDP`:

```
myDP = new Array({name:"Chris", price:"Priceless"}, {name:"Nigel",  
    price:"Cheap"});
```

- **Item**

This is an ActionScript object used for storing the units of information in the cells of a column. A data grid is really a list that can display more than one column of data. A list can be thought of as an array; each indexed space of the list is an item. For the DataGrid component, each item consists of fields. In the following code, the content between curly braces (`{}`) is an item:

```
myDP = new Array({name:"Chris", price:"Priceless"}, {name:"Nigel",  
    price:"Cheap"});
```

- **Field**

Identifiers that indicate the names of the columns within the items. This corresponds to the `columnNames` property in the columns list. In the List component, the fields are usually `label` and `data`, but in the DataGrid component the fields can be any identifier. In the following code, the fields are `name` and `price`:

```
myDP = new Array({name:"Chris", price:"Priceless"}, {name:"Nigel",  
    price:"Cheap"});
```

The view consists of three main parts:

- **Row**

This is a view object responsible for rendering the items of the grid by laying out cells. Each row is laid out horizontally below the previous one.

- **Column**

Columns are fields that are displayed in the grid; the fields each correspond to the `columnName` property of each column.

Each column is a view object (an instance of the `DataGridColumn` class) responsible for displaying each column—for example, width, color, size, and so on.

There are three ways to add columns to a data grid: assign a `DataProvider` object to `DataGrid.dataProvider` (this automatically generates a column for each field in the first item), set `DataGrid.columnNames` to specify which fields will be displayed, or use the constructor for the `DataGridColumn` class to create columns and call `DataGrid.addColumn()` to add them to the grid.

To format columns, either set up style properties for the entire data grid, or define `DataGridColumn` objects, set up their style formats individually, and add them to the data grid.

- Cell

This is a view object responsible for rendering the individual fields of each item. To communicate with the data grid, these components must implement the `CellRenderer` API (see “CellRenderer API” in Flash Help). For a basic data grid, a cell is a built-in `ActionScript` `TextField` object.

DataGrid parameters

You can set the following authoring parameters for each `DataGrid` component instance in the Property inspector or in the Component inspector:

multipleSelection is a Boolean value that indicates whether multiple items can be selected (`true`) or not (`false`). The default value is `false`.

rowHeight indicates the height of each row, in pixels. Changing the font size does not change the row height. The default value is 20.

editable is a Boolean value that indicates whether the grid is editable (`true`) or not (`false`). The default value is `false`.

You can write `ActionScript` to control these and additional options for the `DataGrid` component using its properties, methods, and events. For more information, see “[DataGrid class](#)” on page 139.

Creating an application with the DataGrid component

To create an application with the `DataGrid` component, you must first determine where your data is coming from. The data for a grid can come from a recordset that is fed from a database query in Macromedia ColdFusion, Java, or .Net using Flash Remoting. Data can also come from a data set or an array. To pull the data into a grid, you set the `DataGrid.dataProvider` property to the recordset, data set, or array. You can also use the methods of the `DataGrid` and `DataGridColumn` classes to create data locally. Any `Array` object in the same frame as a `DataGrid` component copies the methods, properties, and events of the `DataProvider` API.

To use Flash Remoting to add a DataGrid component to an application:

1. In Flash, select `File > New` and select `Flash Document`.
2. In the Components panel, double-click the `DataGrid` component to add it to the Stage.
3. In the Property inspector, enter the instance name **myDataGrid**.
4. In the Actions panel on Frame 1, enter the following code:

```
myDataGrid.dataProvider = recordSetInstance;
```

The Flash Remoting recordset `recordSetInstance` is assigned to the `dataProvider` property of `myDataGrid`.

5. Select `Control > Test Movie`.

To use a local data provider to add a DataGrid component to an application:

1. In Flash, select File > New and select Flash Document.
2. In the Components panel, double-click the DataGrid component to add it to the Stage.
3. In the Property inspector, enter the instance name **myDataGrid**.
4. In the Actions panel on Frame 1, enter the following code:

```
myDP = new Array({name:"Chris", price:"Priceless"}, {name:"Nigel",  
    price:"Cheap"});  
myDataGrid.dataProvider = myDP;
```

The name and price fields are used as the column headings, and their values fill the cells in each row.

5. Select Control > Test Movie.

Customizing the DataGrid component

You can transform a DataGrid component horizontally and vertically during authoring and runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)). 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.

Using 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 [“Using styles with the List component” on page 281.](#)) The DataGrid component also supports the following styles:

Style	Theme	Description
<code>backgroundColor</code>	Both	The background color, which can be set for the whole grid or for each column.
<code>backgroundDisabledColor</code>	Both	The background color when the component's <code>enabled</code> property is set to <code>false</code> . The default value is <code>0xDDDDDD</code> (medium gray).
<i>border styles</i>	Both	The DataGrid component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See “RectBorder class” in Flash Help. The default border style value is <code>inset</code> .
<code>headerColor</code>	Both	The color of the column headers. The default value is <code>0xEAEAEA</code> (light gray)
<code>headerStyle</code>	Both	A CSS style declaration for the column header that can be applied to a grid or column to customize the header styles.
<code>color</code>	Both	The text color. The default value is <code>0x0B333C</code> for the Halo theme and blank for the Sample theme.

Style	Theme	Description
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is <code>0x848384</code> (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is <code>10</code> .
<code>fontStyle</code>	Both	The font style: either <code>"normal"</code> or <code>"italic"</code> . The default value is <code>"normal"</code> .
<code>fontWeight</code>	Both	The font weight: either <code>"none"</code> or <code>"bold"</code> . The default value is <code>"none"</code> . All components can also accept the value <code>"normal"</code> in place of <code>"none"</code> during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return <code>"none"</code> .
<code>textAlign</code>	Both	The text alignment: either <code>"left"</code> , <code>"right"</code> , or <code>"center"</code> . The default value is <code>"left"</code> .
<code>textDecoration</code>	Both	The text decoration: either <code>"none"</code> or <code>"underline"</code> . The default value is <code>"none"</code> .
<code>vGridLines</code>	Both	A Boolean value that indicates whether to show vertical grid lines (<code>true</code>) or not (<code>false</code>). The default value is <code>true</code> .
<code>hGridLines</code>	Both	A Boolean value that indicates whether to show horizontal grid lines (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .
<code>vGridLineColor</code>	Both	The color of the vertical grid lines. The default value is <code>0x666666</code> (medium gray).
<code>hGridLineColor</code>	Both	The color of the horizontal grid lines. The default value is <code>0x666666</code> (medium gray).

Setting styles for an individual column

Color and text styles can be set for the grid as a whole or for a column. You can use the following syntax to set a style for a particular column:

```
grid.getColumnAt(3).setStyle("backgroundColor", 0xFF00AA);
```

Setting header styles

You can set header styles through `headerStyle`, which is a style property itself. To do this, you create an instance of `CSSStyleDeclaration`, set the appropriate properties on that instance for the header, and then assign the `CSSStyleDeclaration` to the `headerStyle` property, as shown in the following example.

```
import mx.styles.CSSStyleDeclaration;
var headerStyles = new CSSStyleDeclaration();
```

```
headerStyles.setStyle("fontStyle", "italic");
grid.setStyle("headerStyle", headerStyles);
```

Setting styles for all DataGrid components in a document

The `DataGrid` class inherits from the `List` class, which inherits from the `ScrollSelectList` class. The default class-level style properties are defined on the `ScrollSelectList` class, which the `Menu` component and all `List`-based components extend. You can set new default style values on this class directly, and these new settings will be reflected in all affected components.

```
_global.styles.ScrollSelectList.setStyle("backgroundColor", 0xFF00AA);
```

To set a style property on the `DataGrid` components only, you can create a new instance of `CSSStyleDeclaration` and store it in `_global.styles.DataGrid`.

```
import mx.styles.CSSStyleDeclaration;
if (_global.styles.DataGrid == undefined) {
    _global.styles.DataGrid = new CSSStyleDeclaration();
}
_global.styles.DataGrid.setStyle("backgroundColor", 0xFF00AA);
```

When creating a new class-level style declaration, you will lose all default values provided by the `ScrollSelectList` declaration, including `backgroundColor`, which is required for supporting mouse events. To create a class-level style declaration and preserve defaults, use a `for...in` loop to copy the old settings to the new declaration.

```
var source = _global.styles.ScrollSelectList;
var target = _global.styles.DataGrid;
for (var style in source) {
    target.setStyle(style, source.getStyle(style));
}
```

For more information about class-level styles, see “Setting styles for a component class” in Flash Help.

Using skins with the DataGrid component

The skins that the `DataGrid` component uses to represent its visual states are included in the subcomponents that constitute the data grid (scroll bars and `RectBorder`). For information about their skins, see “Using skins with the `ScrollPane` component” on page 424 and “`RectBorder` class” in Flash Help.

DataGrid class

Inheritance `MovieClip` > `UIObject class` > `UIComponent class` > `View` > `ScrollView` > `ScrollSelectList` > [List component](#) > `DataGrid`

ActionScript Class Name `mx.controls.DataGrid`

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.DataGrid.version);
```

Note: The code `trace(myDataGridInstance.version);` returns `undefined`.

Method summary for the DataGrid class

The following table lists methods of the DataGrid class.

Method	Description
DataGrid.addColumn()	Adds a column to the data grid.
DataGrid.addColumnAt()	Adds a column to the data grid at a specified location.
DataGrid.addItem()	Adds an item to the data grid.
DataGrid.addItemAt()	Adds an item to the data grid at a specified location.
DataGrid.editField()	Replaces the cell data at a specified location.
DataGrid.getColumnAt()	Gets a reference to a column at a specified location.
DataGrid.getColumnIndex()	Gets a reference to the DataGridColumn object at the specified index.
DataGrid.removeAllColumns()	Removes all columns from a data grid.
DataGrid.removeColumnAt()	Removes a column from a data grid at a specified location.
DataGrid.replaceItemAt()	Replaces an item at a specified location with another item.
DataGrid.spaceColumnsEqually()	Spaces all columns equally.

Methods inherited from the UIObject class

The following table lists the methods the DataGrid class inherits from the UIObject class. When calling these methods, use the form *dataGridInstance.methodName*.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it will be redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Methods inherited from the `UIComponent` class

The following table lists the methods the `DataGrid` class inherits from the `UIComponent` class. When calling these methods, use the form `dataGridInstance.methodName`.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Methods inherited from the `List` class

The following table lists the methods the `DataGrid` class inherits from the `List` class. When calling these methods, use the form `dataGridInstance.methodName`.

Method	Description
<code>List.addItem()</code>	Adds an item to the end of the list.
<code>List.addItemAt()</code>	Adds an item to the list at the specified index.
<code>List.getItemAt()</code>	Returns the item at the specified index.
<code>List.removeAll()</code>	Removes all items from the list.
<code>List.removeItemAt()</code>	Removes the item at the specified index.
<code>List.replaceItemAt()</code>	Replaces the item at the specified index with another item.
<code>List.setPropertiesAt()</code>	Applies the specified properties to the specified item.
<code>List.sortItems()</code>	Sorts the items in the list according to the specified compare function.
<code>List.sortItemsBy()</code>	Sorts the items in the list according to a specified property.

Property summary for the `DataGrid` class

The following table lists the properties of the `DataGrid` class.

Property	Description
<code>DataGrid.columnCount</code>	Read-only; the number of columns that are displayed.
<code>DataGrid.columnNames</code>	An array of field names within each item that are displayed as columns.
<code>DataGrid.dataProvider</code>	The data model for a data grid.
<code>DataGrid.editable</code>	A Boolean value that indicates whether the data grid is editable (<code>true</code>) or not (<code>false</code>).
<code>DataGrid.focusedCell</code>	Defines the cell that has focus.
<code>DataGrid.headerHeight</code>	The height of the column headers, in pixels.
<code>DataGrid.hScrollPolicy</code>	Indicates whether a horizontal scroll bar is present (<code>"on"</code>), not present (<code>"off"</code>), or appears when necessary (<code>"auto"</code>).

Property	Description
<code>DataGrid.resizableColumns</code>	A Boolean value that indicates whether the columns are resizable (<code>true</code>) or not (<code>false</code>).
<code>DataGrid.selectable</code>	A Boolean value that indicates whether the data grid is selectable (<code>true</code>) or not (<code>false</code>).
<code>DataGrid.showHeaders</code>	A Boolean value that indicates whether the column headers are visible (<code>true</code>) or not (<code>false</code>).
<code>DataGrid.sortableColumns</code>	A Boolean value that indicates whether the columns are sortable (<code>true</code>) or not (<code>false</code>).

Properties inherited from the `UIObject` class

The following table lists the properties the `DataGrid` class inherits from the `UIObject` class. When accessing these properties from the `DataGrid` object, use the form `dataGridInstance.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `DataGrid` class inherits from the `UIComponent` class. When accessing these properties from the `DataGrid` object, use the form

`dataGridInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Properties inherited from the `List` class

The following table lists the properties the `DataGrid` class inherits from the `List` class. When accessing these properties from the `DataGrid` object, use the form

`dataGridInstance.propertyName`.

Property	Description
<code>List.cellRenderer</code>	Assigns the class or symbol to use to display each row of the list.
<code>List.dataProvider</code>	The source of the list items.
<code>List.hPosition</code>	The horizontal position of the list.
<code>List.hScrollPolicy</code>	Indicates whether the horizontal scroll bar is displayed ("on") or not ("off").
<code>List.iconField</code>	A field in each item to be used to specify icons.
<code>List.iconFunction</code>	A function that determines which icon to use.
<code>List.labelField</code>	Specifies a field of each item to be used as label text.
<code>List.labelFunction</code>	A function that determines which fields of each item to use for the label text.
<code>List.length</code>	The number of items in the list. This property is read-only.
<code>List.maxHPosition</code>	The number of pixels the list can scroll to the right, when <code>List.hScrollPolicy</code> is set to "on".
<code>List.multipleSelection</code>	Indicates whether multiple selection is allowed in the list (<code>true</code>) or not (<code>false</code>).
<code>List.rowCount</code>	The number of rows that are at least partially visible in the list.
<code>List.rowHeight</code>	The pixel height of every row in the list.
<code>List.selectable</code>	Indicates whether the list is selectable (<code>true</code>) or not (<code>false</code>).
<code>List.selectedIndex</code>	The index of a selection in a single-selection list.
<code>List.selectedIndices</code>	An array of the selected items in a multiple-selection list.
<code>List.selectedItem</code>	The selected item in a single-selection list. This property is read-only.

Property	Description
<code>List.selectedItems</code>	The selected item objects in a multiple-selection list. This property is read-only.
<code>List.vPosition</code>	Scrolls the list so the topmost visible item is the number assigned.
<code>List.vScrollPolicy</code>	Indicates whether the vertical scroll bar is displayed ("on"), not displayed ("off"), or displayed when needed ("auto").

Event summary for the DataGrid class

The following table lists the events of the DataGrid class.

Event	Description
<code>DataGrid.cellEdit</code>	Broadcast when the cell value has changed.
<code>DataGrid.cellFocusIn</code>	Broadcast when a cell receives focus.
<code>DataGrid.cellFocusOut</code>	Broadcast when a cell loses focus.
<code>DataGrid.cellPress</code>	Broadcast when a cell is pressed (clicked).
<code>DataGrid.change</code>	Broadcast when an item has been selected.
<code>DataGrid.columnStretch</code>	Broadcast when a user resizes a column horizontally.
<code>DataGrid.headerRelease</code>	Broadcast when a user clicks (releases) a header.

Events inherited from the UIObject class

The following table lists the events the DataGrid class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the DataGrid class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Events inherited from the List class

The following table lists the events the DataGrid class inherits from the List class.

Event	Description
<code>List.change</code>	Broadcast whenever user interaction causes the selection to change.
<code>List.itemRollOut</code>	Broadcast when the pointer rolls over and then off of list items.
<code>List.itemRollOver</code>	Broadcast when the pointer rolls over list items.
<code>List.scroll</code>	Broadcast when a list is scrolled.

DataGrid.addColumn()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.addColumn(dataGridColumn)
```

```
myDataGrid.addColumn(name)
```

Parameters

dataGridColumn An instance of the DataGridColumn class.

name A string that indicates the name of a new DataGridColumn object to be inserted.

Returns

A reference to the DataGridColumn object that was added.

Description

Method; adds a new column to the end of the data grid. For more information, see [“DataGridColumn class” on page 164](#).

Example

The following code adds a new `DataGridColumn` object named `Purple`:

```
import mx.controls.gridclasses.DataGridColumn;
myGrid.addColumn(new DataGridColumn("Purple"));
```

DataGrid.addColumnAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.addColumnAt(index, name)
myDataGrid.addColumnAt(index, dataGridColumn)
```

Parameters

index The index position at which the `DataGridColumn` object is added. The first position is 0.

name A string that indicates the name of the `DataGridColumn` object.

dataGridColumn An instance of the `DataGridColumn` class.

Returns

A reference to the `DataGridColumn` object that was added.

Description

Method; adds a new column at the specified position. Columns are shifted to the right and their indexes are incremented. For more information, see [“DataGridColumn class” on page 164](#).

Example

The following example inserts a new `DataGridColumn` object called `"Green"` at the second and fourth columns:

```
import mx.controls.gridclasses.DataGridColumn;
myGrid.addColumnAt(1, "Green");
myGrid.addColumnAt(3, new DataGridColumn("Purple"));
```

DataGrid.addItem()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.addItem(item)
```

Parameters

item An instance of an object to be added to the grid.

Returns

A reference to the instance that was added.

Description

Method; adds an item to the end of the grid (after the last item index).

Note: This differs from the `List.addItem()` method in that an object is passed rather than a string.

Example

The following example adds a new object to the grid `myGrid`:

```
var anObject= {name:"Jim!!", age:30};  
myGrid.addItem(anObject);
```

DataGrid.addItemAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.addItemAt(index, item)
```

Parameters

index The index position (among the child nodes) at which the node should be added. The first position is 0.

item A string that displays the node.

Returns

A reference to the object instance that was added.

Description

Method; adds an item to the grid at the position specified.

Example

The following example inserts an object instance to the grid at index position 4:

```
var anObject= {name:"Jim!!", age:30};  
myGrid.addItemAt(4, anObject);
```

DataGrid.cellEdit

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.cellEdit = function(eventObject){
    // insert your code here
}
myDataGridInstance.addEventListener("cellEdit", listenerObject)
```

Description

Event; broadcast to all registered listeners when cell value changes.

Version 2 components use a dispatcher/listener event model. The DataGrid component dispatches a `cellEdit` event when the value of a cell has changed, and the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `DataGrid.cellEdit` event's event object has four additional properties:

`columnIndex` A number that indicates the index of the target column.

`itemIndex` A number that indicates the index of the target row.

`oldValue` The previous value of the cell.

`type` The string "cellEdit".

For more information, see "EventDispatcher class" in Flash Help.

Example

In the following example, a handler called `myDataGridListener` is defined and passed to `myDataGrid.addEventListener()` as the second parameter. The event object is captured by the `cellEdit` handler in the *eventObject* parameter. When the `cellEdit` event is broadcast, a `trace` statement is sent to the Output panel.

```
myDataGridListener = new Object();
myDataGridListener.cellEdit = function(event){
    var cell = "(" + event.columnIndex + ", " + event.itemIndex + ")";
    trace("The value of the cell at " + cell + " has changed");
}
myDataGrid.addEventListener("cellEdit", myDataGridListener);
```

Note: The grid must be editable for the above code to work.

DataGrid.cellFocusIn

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.cellFocusIn = function(eventObject){
    // insert your code here
}
myDataGridInstance.addEventListener("cellFocusIn", listenerObject)
```

Description

Event; broadcast to all registered listeners when a particular cell receives focus. This event is broadcast after any previously edited cell's `editCell` and `cellFocusOut` events are broadcast.

Version 2 components use a dispatcher/listener event model. When a DataGrid component dispatches a `cellFocusIn` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `DataGrid.cellFocusIn` event's event object has three additional properties:

`columnIndex` A number that indicates the index of the target column.

`itemIndex` A number that indicates the index of the target row.

`type` The string "cellFocusIn".

For more information, see "EventDispatcher class" in Flash Help.

Example

In the following example, a handler called `myListener` is defined and passed to `grid.addEventListener()` as the second parameter. The event object is captured by the `cellFocusIn` handler in the *eventObject* parameter. When the `cellFocusIn` event is broadcast, a trace statement is sent to the Output panel.

```
var myListener = new Object();
myListener.cellFocusIn = function(event) {
    var cell = "(" + event.columnIndex + ", " + event.itemIndex + ")";
    trace("The cell at " + cell + " has gained focus");
};
grid.addEventListener("cellFocusIn", myListener);
```

Note: The grid must be editable for the above code to work.

DataGrid.cellFocusOut

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.cellFocusOut = function(eventObject){
    // insert your code here
}
myDataGridInstance.addEventListener("cellFocusOut", listenerObject)
```

Description

Event; broadcast to all registered listeners whenever a user moves off a cell that has focus. You can use the event object properties to isolate the cell that was left. This event is broadcast after the `cellEdit` event and before any subsequent `cellFocusIn` events are broadcast by the next cell.

Version 2 components use a dispatcher/listener event model. When a `DataGrid` component dispatches a `cellFocusOut` event, the event is handled by a function (also called a *handler*) that is attached to a listener object that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `DataGrid.cellFocusOut` event's event object has three additional properties:

`columnIndex` A number that indicates the index of the target column. The first position is 0.

`itemIndex` A number that indicates the index of the target row. The first position is 0.

`type` The string "cellFocusOut".

For more information, see "EventDispatcher class" in Flash Help.

Example

In the following example, a handler called `myListener` is defined and passed to `grid.addEventListener()` as the second parameter. The event object is captured by the `cellFocusOut` handler in the *eventObject* parameter. When the `cellFocusOut` event is broadcast, a `trace` statement is sent to the Output panel.

```
var myListener = new Object();
myListener.cellFocusOut = function(event) {
    var cell = "(" + event.columnIndex + ", " + event.itemIndex + ")";
    trace("The cell at " + cell + " has lost focus");
};
grid.addEventListener("cellFocusOut", myListener);
```

Note: The grid must be editable for the above code to work.

DataGrid.cellPress

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.cellPress = function(eventObject){
    // insert your code here
}
myDataGridInstance.addEventListener("cellPress", listenerObject)
```

Description

Event; broadcast to all registered listeners when a user presses the mouse button on a cell.

Version 2 components use a dispatcher/listener event model. When a DataGrid component broadcasts a `cellPress` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `DataGrid.cellPress` event's event object has three additional properties:

`columnIndex` A number that indicates the index of the column that was pressed. The first position is 0.

`itemIndex` A number that indicates the index of the row that was pressed. The first position is 0.

`type` The string "cellPress".

For more information, see "EventDispatcher class" in Flash Help.

Example

In the following example, a handler called `myListener` is defined and passed to `grid.addEventListener()` as the second parameter. The event object is captured by the `cellPress` handler in the *eventObject* parameter. When the `cellPress` event is broadcast, a `trace` statement is sent to the Output panel.

```
var myListener = new Object();
myListener.cellPress = function(event) {
    var cell = "(" + event.columnIndex + ", " + event.itemIndex + ")";
    trace("The cell at " + cell + " has been clicked");
};
grid.addEventListener("cellPress", myListener);
```

DataGrid.change

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.change = function(eventObject){
    // insert your code here
}
myDataGridInstance.addEventListener("change", listenerObject)
```

Description

Event; broadcast to all registered listeners when an item has been selected.

Version 2 components use a dispatcher/listener event model. When a DataGrid component dispatches a `change` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `DataGrid.change` event's event object has one additional property, `type`, whose value is "change". For more information, see "EventDispatcher class" in Flash Help.

Example

In the following example, a handler called `myListener` is defined and passed to `grid.addEventListener()` as the second parameter. The event object is captured by `change` handler in the *eventObject* parameter. When the `change` event is broadcast, a trace statement is sent to the Output panel.

```
var myListener = new Object();
myListener.change = function(event) {
    trace("The selection has changed to " + event.target.selectedIndex);
};
grid.addEventListener("change", myListener);
```

DataGrid.columnCount

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.columnCount
```

Description

Property (read-only); the number of columns displayed.

Example

The following example gets the number of displayed columns in the DataGrid instance `grid`:

```
var c = grid.columnCount;
```

DataGrid.columnNames**Availability**

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.columnNames
```

Description

Property; an array of field names within each item that are displayed as columns.

Example

The following example tells the `grid` instance to display only these three fields as columns:

```
grid.columnNames = ["Name", "Description", "Price"];
```

DataGrid.columnStretch**Availability**

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();  
listenerObject.columnStretch = function(eventObject){  
    // insert your code here  
}  
myDataGridInstance.addEventListener("columnStretch", listenerObject)
```

Description

Event; broadcast to all registered listeners when a user resizes a column horizontally.

Version 2 components use a dispatcher/listener event model. When a DataGrid component dispatches a `columnStretch` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `DataGrid.columnStretch` event's event object has two additional properties:

`columnIndex` A number that indicates the index of the target column. The first position is 0.
`type` The string "columnStretch".

For more information, see "EventDispatcher class" in Flash Help.

Example

In the following example, a handler called `myListener` is defined and passed to `grid.addEventListener()` as the second parameter. The event object is captured by the `columnStretch` handler in the *eventObject* parameter. When the `columnStretch` event is broadcast, a `trace` statement is sent to the Output panel.

```
var myListener = new Object();
myListener.columnStretch = function(event) {
    trace("column " + event.columnIndex + " was resized");
};
grid.addEventListener("columnStretch", myListener);
```

DataGrid.dataProvider

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.dataProvider
```

Description

Property; the data model for items viewed in a DataGrid component.

The data grid adds methods to the prototype of the `Array` class so that each `Array` object conforms to the `DataProvider` API (see `DataProvider.as` in the `Classes/mx/controls/listclasses` folder). Any array that is in the same frame or screen as a data grid automatically has all the methods (`addItem()`, `getItemAt()`, and so on) needed for it to be the data model of a data grid, and can be used to broadcast data model changes to multiple components.

In a `DataGrid` component, you specify fields for display in the `DataGrid.columnNames` property. If you don't define the column set (by setting the `DataGrid.columnNames` property or by calling `DataGrid.addColumn()` for the data grid before the `DataGrid.dataProvider` property has been set, the data grid generates columns for each field in the data provider's first item, once that item arrives.

Any object that implements the `DataProvider` API can be used as a data provider for a data grid (including Flash Remoting recordsets, data sets, and arrays).

Use a grid's data provider to communicate with the data in the grid because the data provider remains consistent, regardless of scroll position.

Example

The following example creates an array to be used as a data provider and assigns it directly to the `dataProvider` property:

```
grid.dataProvider = [{name:"Chris", price:"Priceless"}, {name:"Nigel", price:"cheap"}];
```

The following example creates a new `Array` object that is decorated with the `DataProvider` API. It uses a `for` loop to add 20 items to the grid:

```
myDP = new Array();
for (var i=0; i<20; i++)
    myDP.addItem({name:"Nivesh", price:"Priceless"});
list.dataProvider = myDP
```

DataGrid.editable

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.editable
```

Description

Property; determines whether the data grid can be edited by a user (`true`) or not (`false`). This property must be `true` in order for individual columns to be editable and for any cell to receive focus. The default value is `false`.

If you want individual columns to be uneditable, use the `DataGridColumn.editable` property.

Caution: The `DataGrid` is not editable or sortable if it is bound directly to a `WebServiceConnector` component or an `XMLConnector` component. You must bind the `DataGrid` component to the `DataSet` component and bind the `DataSet` component to the `WebServiceConnector` component or `XMLConnector` component if you want the grid to be editable or sortable.

Example

The following example allows users to edit all the columns of the grid except the first column:

```
myDataGrid.editable = true;
myDataGrid.getColumnAt(0).editable = false;
```

See also

[DataGridColumn.editable](#)

DataGrid.editField()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.editField(index, colName, data)
```

Parameters

index The index of the target cell. This number is zero-based.

colName A string indicating the name of the column (field) that contains the target cell.

data The value to be stored in the target cell. This parameter can be of any data type.

Returns

The data that was in the cell.

Description

Method; replaces the cell data at the specified location and refreshes the data grid with the new value. Any cell present for that value will have its `setValue()` method triggered.

Example

The following example places a value in the grid:

```
var prevValue = myGrid.editField(5, "Name", "Neo");
```

DataGrid.focusedCell

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.focusedCell
```

Description

Property; in editable mode only, an object instance that defines the cell that has focus. The object must have the fields `columnIndex` and `itemIndex`, which are both integers that indicate the index of the column and item of the cell. The origin is (0,0). The default value is `undefined`.

Example

The following example sets the focused cell to the third column, fourth row:

```
grid.focusedCell = {columnIndex:2, itemIndex:3};
```

DataGrid.getColumnAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index)
```

Parameters

index The index of the `DataGridColumn` object to be returned. This number is zero-based.

Returns

A `DataGridColumn` object.

Description

Method; gets a reference to the `DataGridColumn` object at the specified index.

Example

The following example gets the `DataGridColumn` object at index 4:

```
var aColumn = myGrid.getColumnAt(4);
```

DataGrid.getColumnIndex()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnIndex(columnName)
```

Parameters

columnName A string that is the name of a column.

Returns

A number that specifies the index of the column.

Description

Method; returns the index of the column specified by the *columnName* parameter.

DataGrid.headerHeight

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.headerHeight
```

Description

Property; the height of the header bar of the data grid, in pixels. The default value is 20.

Example

The following example sets the scroll position to the top of the display:

```
myDataGrid.headerHeight = 30;
```

DataGrid.headerRelease

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();  
listenerObject.headerRelease = function(eventObject){  
    // insert your code here  
}  
myDataGridInstance.addEventListener("headerRelease", listenerObject)
```

Description

Event; broadcast to all registered listeners when a column header has been released. You can use this event with the `DataGridColumn.sortOnHeaderRelease` property to prevent automatic sorting and to let you sort as you like.

Version 2 components use a dispatcher/listener event model. When the DataGrid component dispatches a `headerRelease` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `DataGrid.headerRelease` event's event object has two additional properties:

`columnIndex` A number that indicates the index of the target column.

`type` The string "headerRelease".

For more information, see "EventDispatcher class" in Flash Help.

Example

In the following example, a handler called `myListener` is defined and passed to `grid.addEventListener()` as the second parameter. The event object is captured by the `headerRelease` handler in the *eventObject* parameter. When the `headerRelease` event is broadcast, a trace statement is sent to the Output panel.

```
var myListener = new Object();
myListener.headerRelease = function(event) {
    trace("column " + event.columnIndex + " header was pressed");
};
grid.addEventListener("headerRelease", myListener);
```

DataGrid.hScrollPolicy

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.hScrollPolicy
```

Description

Property; specifies whether the data grid has a horizontal scroll bar. This property can have the value "on", "off", or "auto". The default value is "off".

If `hScrollPolicy` is set to "off", columns scale proportionally to accommodate the finite width.

Note: This differs from the List component, which cannot have `hScrollPolicy` set to "auto".

Example

The following example sets horizontal scroll policy to automatic, which means that the horizontal scroll bar appears if it's necessary to display all the content:

```
myDataGrid.hScrollPolicy = "auto";
```

DataGrid.removeAllColumns()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.removeAllColumns()
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all DataGridColumn objects from the data grid. Calling this method has no effect on the data provider.

Call this method if you are setting a new data provider that has different fields from the previous data provider, and you want to clear the fields that are displayed.

Example

The following example removes all DataGridColumn objects from `myDataGrid`:

```
myDataGrid.removeAllColumns();
```

DataGrid.removeColumnAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.removeColumnAt(index)
```

Parameters

index The index of the column to remove.

Returns

A reference to the DataGridColumn object that was removed.

Description

Method; removes the DataGridColumn object at the specified index.

Example

The following example removes the `DataGridColumn` object at index 2 in `myDataGrid`:

```
myDataGrid.removeColumnAt(2);
```

DataGrid.replaceItemAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.replaceItemAt(index, item)
```

Parameters

index The index of the item to be replaced.

item An object that is the item value to use as a replacement.

Returns

The previous value.

Description

Method; replaces the item at a specified index and refreshes the display of the grid.

Example

The following example replaces the item at index 4 with the item defined in `aNewValue`:

```
var aNewValue = {name:"Jim", value:"tired"};  
var prevValue = myGrid.replaceItemAt(4, aNewValue);
```

DataGrid.resizableColumns

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.resizableColumns
```

Description

Property; a Boolean value that determines whether the columns of the grid can be stretched by the user (`true`) or not (`false`). This property must be `true` for individual columns to be resizable by the user. The default value is `true`.

Example

The following example prevents users from resizing columns:

```
myDataGrid.resizableColumns = false;
```

DataGrid.selectable

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.selectable
```

Description

Property; a Boolean value that determines whether a user can select the data grid (*true*) or not (*false*). The default value is *true*.

Example

The following example prevents the grid from being selected:

```
myDataGrid.selectable = false;
```

DataGrid.showHeaders

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.showHeaders
```

Description

Property; a Boolean value that indicates whether the data grid displays the column headers (*true*) or not (*false*). Column headers are shaded to differentiate them from the other rows in a grid.

Users can click column headers to sort the contents of the column if

[DataGrid.sortableColumns](#) is set to *true*. The default value of *showHeaders* is *true*.

Example

The following example hides the column headers:

```
myDataGrid.showHeaders = false;
```

See also

[DataGrid.sortableColumns](#)

DataGrid.sortableColumns

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

myDataGrid.sortableColumns

Description

Property; a Boolean value that determines whether the columns of the data grid can be sorted (*true*) or not (*false*) when a user clicks the column headers. This property must be *true* for individual columns to be sortable, and for the *headerRelease* event to be broadcast. The default value is *true*.

Caution: The DataGrid is not editable or sortable if it is bound directly to a *WebServiceConnector* component or an *XMLConnector* component. You must bind the DataGrid component to the *DataSet* component and bind the *DataSet* component to the *WebServiceConnector* component or *XMLConnector* component if you want the grid to be editable or sortable.

Example

The following example turns off sorting:

```
myDataGrid.sortableColumns = false;
```

See also

[DataGrid.headerRelease](#)

DataGrid.spaceColumnsEqually()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

myDataGrid.spaceColumnsEqually()

Parameters

None.

Returns

Nothing.

Description

Method; respaces the columns equally.

Example

The following example respaces the columns of `myGrid` when any column header is pressed and released:

```
myGrid.showHeaders = true
myGrid.dataProvider = [{guitar:"Flying V", name:"maggot"}, {guitar:"SG",
    name:"dreschie"}, {guitar:"jagstang", name:"vitapup"}];
gridLO = new Object();
gridLO.headerRelease = function(){
    myGrid.spaceColumnsEqually();
}
myGrid.addEventListener("headerRelease", gridLO);
```

DataGridColumn class

ActionScript Class Name `mx.controls.gridclasses.DataGridColumn`

You can create and configure `DataGridColumn` objects to use as columns of a data grid. Many of the methods of the `DataGrid` class are dedicated to managing `DataGridColumn` objects.

`DataGridColumn` objects are stored in an zero-based array in the data grid; 0 is the leftmost column. After columns have been added or created, you can access them by calling

`DataGrid.getColumnAt(index)`.

There are three ways to add or create columns in a grid. If you want to configure your columns, it is best to use either the second or third way before you add data to a data grid so you don't have to create columns twice.

- Adding a data provider or an item with multiple fields to a grid that has no configured `DataGridColumn` objects automatically generates columns for every field in the reverse order of the `for...in` loop.
- `DataGrid.columnNames` takes in the field names of the desired item fields and generates `DataGridColumn` objects, in order, for each field listed. This approach lets you select and order columns quickly with a minimal amount of configuration. This approach removes any previous column information.
- The most flexible way to add columns is to prebuild them as `DataGridColumn` objects and add them to the data grid by using `DataGrid.addColumn()`. This approach is useful because it lets you add columns with proper sizing and formatting before the columns ever reach the grid (which reduces processor demand). For more information, see [“Constructor for the DataGridColumn class” on page 165](#).

Property summary for the DataGridColumn class

The following table lists the properties of the `DataGridColumn` class.

Property	Description
<code>DataGridColumn.cellRenderer</code>	The linkage identifier of a symbol to be used to display the cells in this column.
<code>DataGridColumn.columnName</code>	Read-only; the name of the field associated with the column.

Property	Description
<code>DataGridColumn.editable</code>	A Boolean value that indicates whether a column is editable (<code>true</code>) or not (<code>false</code>).
<code>DataGridColumn.headerRenderer</code>	The name of a class to be used to display the header of this column.
<code>DataGridColumn.headerText</code>	The text for the header of this column.
<code>DataGridColumn.labelFunction</code>	A function that determines which field of an item to display.
<code>DataGridColumn.resizable</code>	A Boolean value that indicates whether a column is resizable (<code>true</code>) or not (<code>false</code>).
<code>DataGridColumn.sortable</code>	A Boolean value that indicates whether a column is sortable (<code>true</code>) or not (<code>false</code>).
<code>DataGridColumn.sortOnHeaderRelease</code>	A Boolean value that indicates whether a column is sorted (<code>true</code>) or not (<code>false</code>) when a user clicks a column header.
<code>DataGridColumn.width</code>	The width of a column, in pixels.

Constructor for the `DataGridColumn` class

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
new DataGridColumn(name)
```

Parameters

name A string that indicates the name of the `DataGridColumn` object. This parameter is the field of each item to display.

Returns

Nothing.

Description

Constructor; creates a `DataGridColumn` object. Use this constructor to create columns to add to a `DataGrid` component. After you create the `DataGridColumn` objects, you can add them to a data grid by calling `DataGrid.addColumn()`.

Example

The following example creates a `DataGridColumn` object called `Location`:

```
import mx.controls.gridclasses.DataGridColumn;
var column = new DataGridColumn("Location");
```

DataGridColumn.cellRenderer

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).cellRenderer
```

Description

Property; a linkage identifier for a symbol to be used to display cells in this column. Any class used for this property must implement the CellRenderer API (see “CellRenderer API” in Flash Help.) The default value is `undefined`.

Example

The following example uses a linkage identifier to set a new cell renderer:

```
myGrid.getColumnAt(3).cellRenderer = "MyCellRenderer";
```

DataGridColumn.columnName

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).columnName
```

Description

Property (read-only); the name of the field associated with this column. The default value is the name called in the DataGridColumn constructor.

Example

The following example assigns the column name of the column at the third index position to the variable `name`:

```
var name = myGrid.getColumnAt(3).columnName;
```

See also

[Constructor for the DataGridColumn class](#)

DataGridColumn.editable

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).editable
```

Description

Property; determines whether the column can be edited by a user (`true`) or not (`false`). The `DataGrid.editable` property must be `true` in order for individual columns to be editable, even when `DataGridColumn.editable` is set to `true`. The default value is `true`.

Caution: The DataGrid is not editable or sortable if it is bound directly to a `WebServiceConnector` component or an `XMLConnector` component. You must bind the DataGrid component to the `DataSet` component and bind the `DataSet` component to the `WebServiceConnector` component or `XMLConnector` component if you want the grid to be editable or sortable.

Example

The following example prevents the first column in a grid from being edited:

```
myDataGrid.getColumnAt(0).editable = false;
```

See also

[DataGrid.editable](#)

DataGridColumn.headerRenderer

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).headerRenderer
```

Description

Property; a string that indicates a class name to be used to display the header of this column. Any class used for this property must implement the `CellRenderer` API (see “`CellRenderer` API” in Flash Help). The default value is `undefined`.

Example

The following example uses a linkage identifier to set a new header renderer:

```
myGrid.getColumnAt(3).headerRenderer = "MyHeaderRenderer";
```

DataGridColumn.headerText

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).headerText
```

Description

Property; the text in the column header. The default value is the column name.

This property allows you to display something other than the field name as the header.

Example

The following example sets the column header text to “The Price”:

```
var myColumn = new DataGridColumn("price");  
myColumn.headerText = "The Price";
```

DataGridColumn.labelFunction

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).labelFunction
```

Description

Property; specifies a function to determine which field (or field combination) of each item to display. This function receives one parameter, *item*, which is the item being rendered, and must return a string representing the text to display. This property can be used to create virtual columns that have no equivalent field in the item.

Note: The specified function operates in a nondefined scope.

Example

The following example creates a virtual column:

```
var myCol = myGrid.addColumn("Subtotal");  
myCol.labelFunction = function(item) {  
    return "$" + (item.price + (item.price * salesTax));  
};
```

DataGridColumn.resizable

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).resizable
```

Description

Property; a Boolean value that indicates whether a column can be resized by a user (`true`) or not (`false`). The `DataGrid.resizableColumns` property must be set to `true` for this property to take effect. The default value is `true`.

Example

The following example prevents the column at index 1 from being resized:

```
myGrid.getColumnAt(1).resizable = false;
```

DataGridColumn.sortable

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).sortable
```

Description

Property; a Boolean value that indicates whether a column can be sorted by a user (`true`) or not (`false`). The `DataGrid.sortableColumns` property must be set to `true` for this property to take effect. The default value is `true`.

Caution: The DataGrid is not editable or sortable if it is bound directly to a `WebServiceConnector` component or an `XMLConnector` component. You must bind the DataGrid component to the `DataSet` component and bind the `DataSet` component to the `WebServiceConnector` component or `XMLConnector` component if you want the grid to be editable or sortable.

Example

The following example prevents the column at index 1 from being sorted:

```
myGrid.getColumnAt(1).sortable = false;
```

DataGridColumn.sortOnHeaderRelease

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).sortOnHeaderRelease
```

Description

Property; a Boolean value that indicates whether the column is sorted automatically (`true`) or not (`false`) when a user clicks a header. This property can be set to `true` only if `DataGridColumn.sortable` is set to `true`. If `DataGridColumn.sortOnHeaderRelease` is set to `false`, you can catch the `headerRelease` event and perform your own sort.

The default value is `true`.

Caution: The DataGrid is not editable or sortable if it is bound directly to a `WebServiceConnector` component or an `XMLConnector` component. You must bind the DataGrid component to the `DataSet` component and bind the `DataSet` component to the `WebServiceConnector` component or `XMLConnector` component if you want the grid to be editable or sortable.

Example

The following example lets you catch the `headerRelease` event to perform your own sort:

```
myGrid.getColumnAt(7).sortOnHeaderRelease = false;
```

DataGridColumn.width

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDataGrid.getColumnAt(index).width
```

Description

Property; a number that indicates the width of the column, in pixels. The default value is 50.

Example

The following example makes a column half the default width:

```
myGrid.getColumnAt(4).width = 25;
```

DateChooser component

The DateChooser component is a calendar that allows users to select a date. It has buttons that allow users to scroll through months and click a date to select it. You can set parameters that indicate the month and day names, the first day of the week, and disabled dates, as well as highlighting the current date.

A live preview of each DateChooser instance reflects the values indicated by the Property inspector or Component inspector during authoring.

Using the DateChooser component

The DateChooser can be used anywhere you want a user to select a date. For example, you could use a DateChooser component in a hotel reservation system with certain dates selectable and others disabled. You could also use the DateChooser component in an application that displays current events, such as performances or meetings, when a user chooses a date.

DateChooser parameters

You can set the following authoring parameters for each DateChooser component instance in the Property inspector or in the Component inspector:

monthNames sets the month names that are displayed in the heading row of the calendar. The value is an array and the default value is ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"].

dayNames sets the names of the days of the week. The value is an array and the default value is ["S", "M", "T", "W", "T", "F", "S"].

firstDayOfWeek indicates which day of the week (0-6, 0 being the first element of the dayNames array) is displayed in the first column of the date chooser. This property changes the display order of the day columns.

disabledDays indicates the disabled days of the week. This parameter is an array and can have up to seven values. The default value is [] (an empty array).

showToday indicates whether to highlight today's date. The default value is `true`.

You can write ActionScript to control these and additional options for the DateChooser component using its properties, methods, and events. For more information, see [“DateChooser class” on page 175](#).

Creating an application with the DateChooser component

The following procedure explains how to add a DateChooser component to an application while authoring. In this example, the date chooser allows a user to pick a date for an airline reservation system. All dates before October 15th must be disabled. Also, a range in December must be disabled to create a holiday black-out period, and Mondays must be disabled.

To create an application with the DateChooser component:

1. Double-click the DateChooser component in the Components panel to add it to the Stage.
2. In the Property inspector, enter the instance name **flightCalendar**.
3. In the Actions panel, enter the following code on Frame 1 of the Timeline to set the range of selectable dates:

```
flightCalendar.selectableRange = {rangeStart:new Date(2003, 9, 15),  
    rangeEnd:new Date(2003, 11, 31)}
```

This code assigns a value to the `selectableRange` property in an ActionScript object that contains two `Date` objects with the variable names `rangeStart` and `rangeEnd`. This defines an upper and lower end of a range in which the user can select a date.

4. In the Actions panel, enter the following code on Frame 1 of the Timeline to set a range of holiday disabled dates:

```
flightCalendar.disabledRanges = [{rangeStart: new Date(2003, 11, 15),  
    rangeEnd: new Date(2003, 11, 26)}];
```

5. In the Actions panel, enter the following code on Frame 1 of the Timeline to disable Mondays:

```
flightCalendar.disabledDays=[1];
```

6. Select Control > Test Movie.

Customizing the DateChooser component

You can transform a DateChooser component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)).

Using styles with the DateChooser component

You can set style properties to change the appearance of a DateChooser instance. If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in Flash Help.

A DateChooser component supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The glow color for the rollover and selected dates. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>backgroundColor</code>	Both	The background color. The default value is 0xEFEBEF (light gray).
<code>borderColor</code>	Both	The border color. The default value is 0x919999. The DateChooser component uses a solid single-pixel line as its border. This border cannot be modified through styles or skinning.
<code>headerColor</code>	Both	The background color for the component heading. The default color is white.

Style	Theme	Description
<code>rolloverColor</code>	Both	The background color of a rolled-over date. The default value is <code>0xE3FFD6</code> (bright green) with the Halo theme and <code>0xA9AAAA</code> (light gray) with the Sample theme.
<code>selectionColor</code>	Both	The background color of the selected date. The default value is <code>0xCDFFC1</code> (light green) with the Halo theme and <code>0xE8E8E8</code> (very light gray) with the Sample theme.
<code>todayColor</code>	Both	The background color for the today's date. The default value is <code>0x666666</code> (dark gray).
<code>color</code>	Both	The text color. The default value is <code>0x0B333C</code> with the Halo theme and blank with the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is <code>0x848384</code> (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either <code>"normal"</code> or <code>"italic"</code> . The default value is <code>"normal"</code> .
<code>fontWeight</code>	Both	The font weight: either <code>"none"</code> or <code>"bold"</code> . The default value is <code>"none"</code> . All components can also accept the value <code>"normal"</code> in place of <code>"none"</code> during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return <code>"none"</code> .
<code>textDecoration</code>	Both	The text decoration: either <code>"none"</code> or <code>"underline"</code> . The default value is <code>"none"</code> .

The `DateChooser` component uses four categories of text to display the month name, the days of the week, today's date, and regular dates. The text style properties set on the `DateChooser` component itself control the regular date text and provide defaults for the other text. To set text styles for specific categories of text, use the following class-level style declarations.

Declaration name	Description
<code>HeaderDateText</code>	The month name.
<code>WeekDayStyle</code>	The days of the week.
<code>TodayStyle</code>	Today's date.

The following example demonstrates how to set the month name and days of the week to a deep red color.

```
_global.styles.HeaderDateText.setStyle("color", 0x660000);
_global.styles.WeekDayStyle.setStyle("color", 0x660000);
```

Using skins with the DateChooser component

The DateChooser component uses skins to represent the forward and back month buttons and the today indicator. To skin the DateChooser component while authoring, modify skin symbols in the Flash UI Components 2/Themes/MMDefault/DateChooser Assets/States folder in the library of one of the themes FLA files. For more information, see “About skinning components” in Flash Help.

Only the month scrolling buttons can be dynamically skinned in this component. A DateChooser component uses the following skin properties:

Property	Description
<code>backMonthButtonUpSymbolName</code>	The month back button up state. The default value is <code>backMonthUp</code> .
<code>backMonthButtonDownSymbolName</code>	The month back button pressed state. The default value is <code>backMonthDown</code> .
<code>backMonthButtonDisabledSymbolName</code>	The month back button disabled state. The default value is <code>backMonthDisabled</code> .
<code>fwdMonthButtonUpSymbolName</code>	The month forward button up state. The default value is <code>fwdMonthUp</code> .
<code>fwdMonthButtonDownSymbolName</code>	The month forward button pressed state. The default value is <code>fwdMonthDown</code> .
<code>fwdMonthButtonDisabledSymbolName</code>	The month forward button disabled state. The default value is <code>fwdMonthDisabled</code> .

The button symbols are used exactly as is without applying colors or resizing. The size is determined by the symbol during authoring.

To create movie clip symbols for DateChooser skins:

1. Create a new FLA file.
2. Select File > Import > Open External Library, and select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in Flash Help.
3. In the theme’s Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the DateChooser Assets folder to the library for your document.
4. Expand the DateChooser Assets/States folder in the library of your document.
5. Open the symbols you want to customize for editing.
For example, open the `backMonthDown` symbol.
6. Customize the symbol as desired.
For example, change the tint of the arrow to red.
7. Repeat steps 5-6 for all symbols you want to customize.
For example, change the tint of the forward arrow down symbol to match the back arrow.
8. Click the Back button to return to the main Timeline.

9. Drag a DateChooser component to the Stage.

10. Select Control > Test Movie.

Note: The DateChooser Assets/States folder also has a Day Skins folder with a single skin element, `cal_todayIndicator`. This element can be modified during authoring to customize the today indicator. However, it cannot be changed dynamically on a particular DateChooser instance to use a different symbol. In addition, the `cal_todayIndicator` symbol must be a solid single-color graphic, because the DateChooser component will apply the `todayColor` color to the graphic as a whole. The graphic may have cut-outs, but keep in mind that the default text color for today's date is white and the default background for the DateChooser is white, so a cut-out in the middle of the today indicator skin element would make today's date unreadable unless either the background color or today text color is also changed.

DateChooser class

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > DateChooser

ActionScript Class Name mx.controls.DateChooser

The properties of the DateChooser class let you access the selected date and the displayed month and year. You can also set the names of the days and months, indicate disabled dates and selectable dates, set the first day of the week, and indicate whether the current date should be highlighted.

Setting a property of the DateChooser class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.DateChooser.version);
```

Note: The code `trace(myDC.version);` returns `undefined`.

Method summary for the DateChooser class

There are no methods exclusive to the DateChooser class.

Methods inherited from the UIObject class

The following table lists the methods the DateChooser class inherits from the UIObject class.

When calling these methods from the DateChooser object, use the form

dateChooserInstance.methodName.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.

Method	Description
<code>UIObject.invalidate()</code>	Marks the object so it will be redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the `UIComponent` class

The following table lists the methods the `DateChooser` class inherits from the `UIComponent` class. When calling these methods from the `DateChooser` object, use the form *dateChooserInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the `DateChooser` class

The following table lists the properties that are exclusive to the `DateChooser` class.

Property	Description
<code>DateChooser.dayNames</code>	An array indicating the names of the days of the week.
<code>DateChooser.disabledDays</code>	An array indicating the days of the week that are disabled for all applicable dates in the date chooser.
<code>DateChooser.disabledRanges</code>	A range of disabled dates or a single disabled date.
<code>DateChooser.displayedMonth</code>	A number indicating an element in the <code>monthNames</code> array to display in the date chooser.
<code>DateChooser.displayedYear</code>	A number indicating the year to display.
<code>DateChooser.firstDayOfWeek</code>	A number indicating an element in the <code>dayNames</code> array to display in the first column of the date chooser.
<code>DateChooser.monthNames</code>	An array of strings indicating the month names.
<code>DateChooser.selectableRange</code>	A single selectable date or a range of selectable dates.
<code>DateChooser.selectedDate</code>	A <code>Date</code> object indicating the selected date.
<code>DateChooser.showToday</code>	A Boolean value indicating whether the current date is highlighted.

Properties inherited from the `UIObject` class

The following table lists the properties the `DateChooser` class inherits from the `UIObject` class. When accessing these properties from the `DateChooser` object, use the form `dateChooserInstance.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `DateChooser` class inherits from the `UIComponent` class. When accessing these properties from the `DateChooser` object, use the form `dateChooserInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the `DateChooser` class

The following table lists the events that are exclusive to the `DateChooser` class.

Event	Description
<code>DateChooser.change</code>	Broadcast when a date is selected.
<code>DateChooser.scroll</code>	Broadcast when the month buttons are clicked.

Events inherited from the UIObject class

The following table lists the events the DateChooser class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the DateChooser class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

DateChooser.change

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
on(change){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.change = function(eventObject){  
    ...  
}  
chooserInstance.addEventListener("change", listenerObject)
```

Description

Event; broadcast to all registered listeners when a date is selected.

The first usage example uses an `on()` handler and must be attached directly to a `DateChooser` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the date chooser `myDC`, sends “_level0.myDC” to the Output panel:

```
on(change){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*chooserInstance*) dispatches an event (in this case, `change`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

This example, written on a frame of the Timeline, sends a message to the Output panel when a `DateChooser` instance called `myDC` is changed. The first line of code creates a listener object called `form`. The second line defines a function for the `change` event on the listener object. Inside the function is a `trace()` statement that uses the event object that is automatically passed to the function, in this example `eventObj`, to generate a message. The `target` property of an event object is the component that generated the event (in this example, `myDC`). The `NumericStepper.maximum` property is accessed from the event object’s `target` property. The last line calls `EventDispatcher.addEventListener()` from `myDC` and passes it the `change` event and the `form` listener object as parameters.

```
form.change = function(eventObj){
    trace("date selected " + eventObj.target.selectedDate);
}
myDC.addEventListener("change", form);
```

DateChooser.dayNames

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

myDC.dayNames

Description

Property; an array containing the names of the days of the week. Sunday is the first day (at index position 0) and the rest of the day names follow in order. The default value is ["S", "M", "T", "W", "T", "F", "S"].

Example

The following example changes the value of the fifth day of the week (Thursday) from "T" to "R":

```
myDC.dayNames[4] = "R";
```

DateChooser.disabledDays**Availability**

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

myDC.disabledDays

Description

Property; an array indicating the disabled days of the week. All the dates in a month that fall on the specified day are disabled. The elements of this array can have values from 0 (Sunday) to 6 (Saturday). The default value is [] (an empty array).

Example

The following example disables Sundays and Saturdays so that users can select only weekdays:

```
myDC.disabledDays = [0, 6];
```

DateChooser.disabledRanges**Availability**

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

myDC.disabledRanges

Description

Property; disables a single day or a range of days. This property is an array of objects. Each object in the array must be either a `Date` object that specifies a single day to disable, or an object that contains either or both of the properties `rangeStart` and `rangeEnd`, each of whose value must be a `Date` object. The `rangeStart` and `rangeEnd` properties describe the boundaries of the date range. If either property is omitted, the range is unbounded in that direction.

The default value of `disabledRanges` is undefined.

Specify a full date when you define dates for the `disabledRanges` property. For example, specify `new Date(2003,6,24)` rather than `new Date()`. If you don't specify a full date, the time returns as `00:00:00`.

Example

The following example defines an array with `rangeStart` and `rangeEnd` `Date` objects that disable the dates between May 7 and June 7:

```
myDC.disabledRanges = [{rangeStart: new Date(2003, 4, 7), rangeEnd: new Date(2003, 5, 7)}];
```

The following example disables all dates after November 7:

```
myDC.disabledRanges = [ {rangeStart: new Date(2003, 10, 7)} ];
```

The following example disables all dates before October 7:

```
myDC.disabledRanges = [ {rangeEnd: new Date(2002, 9, 7)} ];
```

The following example disables only December 7:

```
myDC.disabledRanges = [ new Date(2003, 11, 7) ];
```

DateChooser.displayedMonth

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDC.displayedMonth
```

Description

Property; a number indicating which month is displayed. The number indicates an element in the `monthNames` array, with 0 being the first month. The default value is the month of the current date.

Example

The following example sets the displayed month to December:

```
myDC.displayedMonth = 11;
```

See also

[DateChooser.displayedYear](#)

DateChooser.displayedYear

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDC.displayedYear
```

Description

Property; a four-digit number indicating which year is displayed. The default value is the current year.

Example

The following example sets the displayed year to 2010:

```
myDC.displayedYear = 2010;
```

See also

[DateChooser.displayedMonth](#)

DateChooser.firstDayOfWeek

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDC.firstDayOfWeek
```

Description

Property; a number indicating which day of the week (0-6, 0 being the first element of the `dayNames` array) is displayed in the first column of the `DateChooser` component. Changing this property changes the order of the day columns but has no effect on the order of the `dayNames` property. The default value is 0 (Sunday).

Example

The following example sets the first day of the week to Monday:

```
myDC.firstDayOfWeek = 1;
```

See also

[DateChooser.dayNames](#)

DateChooser.monthNames

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

myDC.monthNames

Description

Property; an array of strings indicating the month names at the top of the DateChooser component. The default value is ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"].

Example

The following example sets the month names for the instance *myDC*:

```
myDC.monthNames = ["Jan", "Feb", "Mar", "Apr", "May", "June", "July", "Aug",  
"Sept", "Oct", "Nov", "Dec"];
```

DateChooser.scroll

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
on(scroll){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.scroll = function(eventObject){  
    ...  
}  
myDC.addEventListener("scroll", listenerObject)
```

Description

Event; broadcast to all registered listeners when a month button is clicked.

The first usage example uses an `on()` handler and must be attached directly to a `DateChooser` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the date chooser `myDC`, sends “_level0.myDC” to the Output panel:

```
on(scroll){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*myDC*) dispatches an event (in this case, `scroll`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The scroll event’s event object has an additional property, `detail`, that can have one of the following values: `nextMonth`, `previousMonth`, `nextYear`, `previousYear`.

Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

This example, written on a frame of the Timeline, sends a message to the Output panel when a month button is clicked on a `DateChooser` instance called `myDC`. The first line of code creates a listener object called `form`. The second line defines a function for the `scroll` event on the listener object. Inside the function is a `trace()` statement that uses the event object that is automatically passed to the function, in this example `eventObj`, to generate a message. The `target` property of an event object is the component that generated the event—in this example, `myDC`. The last line calls `EventDispatcher.addEventListener()` from `myDC` and passes it the `scroll` event and the `form` listener object as parameters.

```
form = new Object();
form.scroll = function(eventObj){
    trace(eventObj.detail);
}
myDC.addEventListener("scroll", form);
```

DateChooser.selectableRange

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

myDC.selectableRange

Description

Property; sets a single selectable date or a range of selectable dates. The user cannot scroll beyond the selectable range. The value of this property is an object that consists of two `Date` objects named `rangeStart` and `rangeEnd`. The `rangeStart` and `rangeEnd` properties designate the boundaries of the selectable date range. If only `rangeStart` is defined, all the dates after `rangeStart` are enabled. If only `rangeEnd` is defined, all the dates before `rangeEnd` are enabled. The default value is undefined.

If you want to enable only a single day, you can use a single `Date` object as the value of `selectableRange`.

Specify a full date when you define dates—for example, `new Date(2003, 6, 24)` rather than `new Date()`. If you don't specify a full date, the time returns as `00:00:00`.

The value of `DateChooser.selectedDate` is set to undefined if it falls outside the selectable range.

The values of `DateChooser.displayedMonth` and `DateChooser.displayedYear` are set to the nearest last month in the selectable range if the current month falls outside the selectable range. For example, if the current displayed month is August, and the selectable range is from June 2003 to July, 2003, the displayed month will change to July 2003.

Example

The following example defines the selectable range as the dates between and including May 7 and June 7:

```
myDC.selectableRange = {rangeStart: new Date(2001, 4, 7), rangeEnd: new Date(2003, 5, 7)};
```

The following example defines the selectable range as the dates after and including May 7:

```
myDC.selectableRange = {rangeStart: new Date(2003, 4, 7)};
```

The following example defines the selectable range as the dates before and including June 7:

```
myDC.selectableRange = {rangeEnd: new Date(2003, 5, 7)};
```

The following example defines the selectable date as June 7 only:

```
myDC.selectableRange = new Date(2003, 5, 7);
```

DateChooser.selectedDate

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

myDC.selectedDate

Description

Property; a Date object that indicates the selected date if that value falls within the value of the `selectableRange` property. The default value is `undefined`.

You cannot set the `selectedDate` property within a disabled range, outside a selectable range, or on a day that has been disabled. If this property is set to one of these dates, the value is `undefined`.

Example

The following example sets the selected date to June 7:

```
myDC.selectedDate = new Date(2003, 5, 7);
```

DateChooser.showToday

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDC.showToday
```

Description

Property; a Boolean value that determines whether the current date is highlighted. The default value is `true`.

Example

The following example turns off the highlighting on today's date:

```
myDC.showToday = false;
```

DateField component

The DateField component is a nonselectable text field that displays the date with a calendar icon on its right side. If no date has been selected, the text field is blank and the month of today's date is displayed in the date chooser. When a user clicks anywhere inside the bounding box of the date field, a date chooser pops up and displays the dates in the month of the selected date. When the date chooser is open, users can use the month scroll buttons to scroll through months and years and select a date. When a date is selected, the date chooser closes.

The live preview of the DateField does not reflect the values indicated by the Property inspector or Component inspector during authoring, because it is a pop-up component that is not visible during authoring.

Using the DateField component

The DateField component can be used anywhere you want a user to select a date. For example, you could use a DateField component in a hotel reservation system with certain dates selectable and others disabled. You could also use the DateField component in an application that displays current events, such as performances or meetings, when a user chooses a date.

DateField parameters

You can set the following authoring parameters for each DateField component instance in the Property inspector or in the Component inspector:

monthNames sets the month names that are displayed in the heading row of the calendar. The value is an array and the default value is ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"].

dayNames sets the names of the days of the week. The value is an array and the default value is ["S", "M", "T", "W", "T", "F", "S"].

firstDayOfWeek indicates which day of the week (0-6, 0 being the first element of dayNames array) is displayed in the first column of the date chooser. This property changes the display order of the day columns.

The default value is 0, which is "S".

disabledDays indicates the disabled days of the week. This parameter is an array and can have up to seven values. The default value is [] (an empty array).

showToday indicates whether to highlight today's date. The default value is `true`.

You can write ActionScript to control these and additional options for the DateField component using its properties, methods, and events. For more information, see [“DateField class” on page 191](#).

Creating an application with the DateField component

The following procedure explains how to add a DateField component to an application while authoring. In this example, the DateField component allows a user to pick a date for an airline reservation system. All dates before today's date must be disabled. Also, a 15-day range in December must be disabled to create a holiday black-out period. Also, some flights are not available on Mondays, so all Mondays must be disabled for those flights.

To create an application with the DateField component:

1. Double-click the DateField component in the Components panel to add it to the Stage.
2. In the Property inspector, enter the instance name `flightCalendar`.

3. In the Actions panel, enter the following code on Frame 1 of the Timeline to set the range of selectable dates:

```
flightCalendar.selectableRange = {rangeStart:new Date(2001, 9, 1),
    rangeEnd:new Date(2003, 11, 1)};
```

This code assigns a value to the `selectableRange` property in an ActionScript object that contains two `Date` objects with the variable names `rangeStart` and `rangeEnd`. This defines an upper and lower end of a range within which the user can select a date.

4. In the Actions panel, enter the following code on Frame 1 of the Timeline to set the ranges of disabled dates, one during December, and one for all dates before the current date:

```
flightCalendar.disabledRanges = [{rangeStart: new Date(2003, 11, 15),
    rangeEnd: new Date(2003, 11, 31)}, {rangeEnd: new Date(2003, 6, 16)}];
```

5. In the Actions panel, enter the following code on Frame 1 of the Timeline to disable Mondays:

```
flightCalendar.disabledDays=[1];
```

6. Control > Test Movie.

Customizing the DateField component

You can transform a `DateField` component horizontally while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)). Setting the width does not change the dimensions of the date chooser in the `DateField` component. However, you can use the `pullDown` property to access the `DateChooser` component and set its dimensions.

Using styles with the DateField component

You can set style properties to change the appearance of a date field instance. If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in Flash Help.

The `DateField` component supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The glow color for the rollover and selected dates. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen"
<code>backgroundColor</code>	Both	The background color. The default value is 0xEFEBEF (light gray).
<code>borderColor</code>	Both	The border color. The default value is 0x919999.

The `DateField` component's drop-down list uses a solid single-pixel line as its border. This border cannot be modified through styles or skinning.

Style	Theme	Description
<code>headerColor</code>	Both	The background color for the drop-down heading. The default color is white.
<code>rolloverColor</code>	Both	The background color of a rolled-over date. The default value is <code>OxE3FFD6</code> (bright green) with the Halo theme and <code>OxAAAAAA</code> (light gray) with the Sample theme.
<code>selectionColor</code>	Both	The background color of the selected date. The default value is a <code>OxCDFFC1</code> (light green) with the Halo theme and <code>OxEFFFFFFE</code> (very light gray) with the Sample theme.
<code>todayColor</code>	Both	The background color for the today's date. The default value is <code>Ox666666</code> (dark gray).
<code>color</code>	Both	The text color. The default value is <code>Ox0B333C</code> with the Halo theme and blank with the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is <code>Ox848384</code> (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either <code>"normal"</code> or <code>"italic"</code> . The default value is <code>"normal"</code> .
<code>fontWeight</code>	Both	The font weight: either <code>"none"</code> or <code>"bold"</code> . The default value is <code>"none"</code> . All components can also accept the value <code>"normal"</code> in place of <code>"none"</code> during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return <code>"none"</code> .
<code>textDecoration</code>	Both	The text decoration: either <code>"none"</code> or <code>"underline"</code> . The default value is <code>"none"</code> .

The `DateField` component uses four categories of text to display the month name, the days of the week, today's date, and regular dates. The text style properties set on the `DateField` component itself control the regular date text and the text displayed in the collapsed state, and provide defaults for the other text. To set text styles for specific categories of text, use the following class-level style declarations.

Declaration name	Description
<code>HeaderDateText</code>	The month name.
<code>WeekDayStyle</code>	The days of the week.
<code>TodayStyle</code>	Today's date.

The following example demonstrates how to set the month name and days of the week to a deep red color.

```
_global.styles.HeaderDateText.setStyle("color", 0x660000);  
_global.styles.WeekDayStyle.setStyle("color", 0x660000);
```

Using skins with the DateField component

The DateField component uses skins to represent the visual states of the pop-up icon, a RectBorder instance for the border around the text input, and a DateChooser instance for the pop-up. To skin the pop-up icon while authoring, modify skin symbols in the Flash UI Components 2/Themes/MMDefault/DateField Assets/States folder in the library of one of the themes FLA files. For more information, see “About skinning components” in Flash Help. For information about skinning the RectBorder and DateChooser instances, see “RectBorder class” in Flash Help and [“Using skins with the DateChooser component” on page 174](#).

Besides the skins used by the subcomponents mentioned above, a DateField component uses the following skin properties to dynamically skin the pop-up icon:

Property	Description
openDateUp	The up state of the pop-up icon.
openDateDown	The down state of the pop-up icon.
openDateOver	The over state of the pop-up icon.
openDateDisabled	The disabled state of the pop-up icon.

To create movie clip symbols for DateField skins:

1. Create a new FLA file.
2. Select File > Import > Open External Library, and select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in Flash Help.
3. In the theme’s Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the DateField Assets folder to the library for your document.
4. Expand the DateField Assets folder in the library of your document.
5. Ensure that the DateFieldAssets symbol is selected for Export in First Frame.
6. Expand the DateField Assets/States folder in the library of your document.
7. Open the symbols you want to customize for editing.
For example, open the openIconUp symbol.
8. Customize the symbol as desired.
For example, draw a down arrow over the calendar image.
9. Repeat steps 7-8 for all symbols you want to customize.
For example, draw a down arrow over all of the symbols.
10. Click the Back button to return to the main Timeline.

11. Drag a DateField component to the Stage.
12. Select Control > Test Movie.

DateField class

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > ComboBase > DateField

ActionScript Class Name mx.controls.DateField

The properties of the DateField class let you access the selected date and the displayed month and year. You can also set the names of the days and months, indicate disabled dates and selectable dates, set the first day of the week, and indicate whether the current date should be highlighted.

Setting a property of the DateField class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.DateField.version);
```

Note: The code `trace(myDateFieldInstance.version);` returns `undefined`.

Method summary for the DateField class

The following table lists methods of the DateField class.

Method	Description
DateField.close()	Closes the pop-up DateChooser subcomponent.
DateField.open()	Opens the pop-up DateChooser subcomponent.

Methods inherited from the UIObject class

The following table lists the methods the DateField class inherits from the UIObject class. When calling these methods from the DateField object, use the form `dateFieldInstance.methodName`.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it will be redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.

Method	Description
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the `UIComponent` class

The following table lists the methods the `DateField` class inherits from the `UIComponent` class. When calling these methods from the `DateField` object, use the form `dateFieldInstance.methodName`.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the `DateField` class

The following table lists properties of the `DateField` class.

Property	Description
<code>DateField.dateFormatter</code>	A function that formats the date to be displayed in the text field.
<code>DateField.dayNames</code>	An array indicating the names of the days of the week.
<code>DateField.disabledDays</code>	An array indicating the disabled days of the week.
<code>DateField.disabledRanges</code>	A range of disabled dates or a single disabled date.
<code>DateField.displayedMonth</code>	A number indicating which element in the <code>monthNames</code> array to display.
<code>DateField.displayedYear</code>	A number indicating the year to display.
<code>DateField.firstDayOfWeek</code>	A number indicating an element in the <code>dayNames</code> array to display in the first column of the <code>DateField</code> component.
<code>DateField.monthNames</code>	An array of strings indicating the month names.
<code>DateField.pullDown</code>	A reference to the <code>DateChooser</code> subcomponent. This property is read-only.
<code>DateField.selectableRange</code>	A single selectable date or a range of selectable dates.
<code>DateField.selectedDate</code>	A <code>Date</code> object indicating the selected date.
<code>DateField.showToday</code>	A Boolean value indicating whether the current date is highlighted.

Properties inherited from the UIObject class

The following table lists the properties the DateField class inherits from the UIObject class. When accessing these properties from the DateField object, use the form *dateFieldInstance.propertyName*.

Property	Description
UIObject.bottom	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
UIObject.height	The height of the object, in pixels. Read-only.
UIObject.left	The left edge of the object, in pixels. Read-only.
UIObject.right	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
UIObject.scaleX	A number indicating the scaling factor in the x direction of the object, relative to its parent.
UIObject.scaleY	A number indicating the scaling factor in the y direction of the object, relative to its parent.
UIObject.top	The position of the top edge of the object, relative to its parent. Read-only.
UIObject.visible	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
UIObject.width	The width of the object, in pixels. Read-only.
UIObject.x	The left edge of the object, in pixels. Read-only.
UIObject.y	The top edge of the object, in pixels. Read-only.

Properties inherited from the UIComponent class

The following table lists the properties the DateField class inherits from the UIComponent class. When accessing these properties from the DateField object, use the form *dateFieldInstance.propertyName*.

Property	Description
UIComponent.enabled	Indicates whether the component can receive focus and input.
UIComponent.tabIndex	A number indicating the tab order for a component in a document.

Event summary for the DateField class

The following table lists events of the DateField class.

Event	Description
DateField.change	Broadcast when a date is selected.
DateField.close	Broadcast when the DateChooser subcomponent closes.

Event	Description
<code>DateField.open</code>	Broadcast when the DateChooser subcomponent opens.
<code>DateField.scroll</code>	Broadcast when the month buttons are clicked.

Events inherited from the UIObject class

The following table lists the events the DateField class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the DateField class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

DateField.change

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
on(change){
    ...
}
```

Usage 2:

```
listenerObject = new Object();
listenerObject.change = function(eventObject){
    ...
}
myDF.addEventListener("change", listenerObject)
```

Description

Event; broadcast to all registered listeners when a date is selected.

The first usage example uses an `on()` handler and must be attached directly to a `DateField` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the date field `myDF`, sends “_level0.myDF” to the Output panel:

```
on(change){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*myDF*) dispatches an event (in this case, *change*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

This example, written on a frame of the Timeline, sends a message to the Output panel when a date field called `myDF` is changed. The first line of code creates a listener object called `form`. The second line defines a function for the `change` event on the listener object. Inside the function is a `trace()` statement that uses the event object that is automatically passed to the function, in this example `eventObj`, to generate a message. The `target` property of an event object is the component that generated the event—in this example, `myDF`. The `DateField.selectedDate` property is accessed from the event object’s `target` property. The last line calls `EventDispatcher.addEventListener()` from `myDF` and passes it the `change` event and the `form` listener object as parameters.

```
function change(eventObj){
    trace("date selected " + eventObj.target.selectedDate) ;
}
myDF.addEventListener("change", this);
```

DateField.close()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDF.close()
```

Parameters

None.

Returns

Nothing.

Description

Method; closes the pop-up menu.

Example

The following code closes the date chooser pop-up of the `myDF` date field instance:

```
myDF.close();
```

DateField.close

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
on(close){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.close = function(eventObject){  
    ...  
}  
myDF.addEventListener("close", listenerObject)
```

Description

Event; broadcast to all registered listeners when the DateChooser subcomponent closes after a user clicks outside the icon or selects a date.

The first usage example uses an `on()` handler and must be attached directly to a `DateField` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the date field `myDF`, sends “_level0.myDF” to the Output panel:

```
on(close){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*myDF*) dispatches an event (in this case, `close`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

This example, written on a frame of the Timeline, sends a message to the Output panel when the date chooser in `myDF` closes. The first line of code creates a listener object called `form`. The second line defines a function for the `close` event on the listener object. Inside the function is a `trace()` statement that uses the event object that is automatically passed to the function, in this example `eventObj`, to generate a message. The `target` property of an event object is the component that generated the event—in this example, `myDF`. The `target` property is accessed from the event object’s `target` property. The last line calls `EventDispatcher.addEventListener()` from `myDF` and passes it the `close` event and the `form` listener object as parameters.

```
form.close = function(eventObj){
    trace("PullDown Closed" + eventObj.target.selectedDate);
}
myDF.addEventListener("close", form);
```

DateField.dateFormatter

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDF.dateFormatter
```

Description

Property; a function that formats the date to be displayed in the text field. The function must receive a Date object as parameter, and return a string in the format to be displayed.

Example

The following example sets the function to return the format of the date to be displayed:

```
myDF.dateFormatter = function(d:Date){
    return d.getFullYear()+" / "+(d.getMonth()+1)+" / "+d.getDate();
};
```

DateField.dayNames

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDF.dayNames
```

Description

Property; an array containing the names of the days of the week. Sunday is the first day (at index position 0) and the other day names follow in order. The default value is ["S", "M", "T", "W", "T", "F", "S"].

Example

The following example changes the value of the fifth day of the week (Thursday) from "T" to "R":

```
myDF.dayNames[4] = "R";
```

DateField.disabledDays

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDF.disabledDays
```

Description

Property; an array indicating the disabled days of the week. All the dates in a month that fall on the specified day are disabled. The elements of this array can have values between 0 (Sunday) and 6 (Saturday). The default value is [] (an empty array).

Example

The following example disables Sundays and Saturdays so that users can select only weekdays:

```
myDF.disabledDays = [0, 6];
```

DateField.disabledRanges

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDF.disabledRanges
```

Description

Property; disables a single day or a range of days. This property is an array of objects. Each object in the array must be either a `Date` object specifying a single day to disable, or an object containing either or both of the properties `rangeStart` and `rangeEnd`, each of whose value must be a `Date` object. The `rangeStart` and `rangeEnd` properties describe the boundaries of the date range. If either property is omitted, the range is unbounded in that direction.

The default value of `disabledRanges` is `undefined`.

Specify a full date when you define dates for the `disabledRanges` property—for example, `new Date(2003,6,24)` rather than `new Date()`. If you don't specify a full date, the time returns as `00:00:00`.

Example

The following example defines an array with `rangeStart` and `rangeEnd` `Date` objects that disable the dates between May 7 and June 7:

```
myDF.disabledRanges = [ {rangeStart: new Date(2003, 4, 7), rangeEnd: new  
    Date(2003, 5, 7)}];
```

The following example disables all dates after November 7:

```
myDF.disabledRanges = [ {rangeStart: new Date(2003, 10, 7)} ];
```

The following example disables all dates before October 7:

```
myDF.disabledRanges = [ {rangeEnd: new Date(2002, 9, 7)} ];
```

The following example disables only December 7:

```
myDF.disabledRanges = [ new Date(2003, 11, 7) ];
```

DateField.displayedMonth

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDF.displayedMonth
```

Description

Property; a number indicating which month is displayed. The number indicates an element in the `monthNames` array, with 0 being the first month. The default value is the month of the current date.

Example

The following example sets the displayed month to December:

```
myDF.displayedMonth = 11;
```

See also

[DateField.displayedYear](#)

DateField.displayedYear

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDF.displayedYear
```

Description

Property; a number indicating which year is displayed. The default value is the current year.

Example

The following example sets the displayed year to 2010:

```
myDF.displayedYear = 2010;
```

See also

[DateField.displayedMonth](#)

DateField.firstDayOfWeek

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDF.firstDayOfWeek
```

Description

Property; a number indicating which day of the week (0-6, 0 being the first element of the `dayNames` array) is displayed in the first column of the `DateField` component. Changing this property changes the order of the day columns but has no effect on the order of the `dayNames` property. The default value is 0 (Sunday).

Example

The following example sets the first day of the week to Monday:

```
myDF.firstDayOfWeek = 1;
```

See also

[DateField.dayNames](#)

DateField.monthNames

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDF.monthNames
```

Description

Property; an array of strings indicating the month names at the top of the `DateField` component. The default value is ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"].

Example

The following example sets the month names for the instance `myDF`:

```
myDF.monthNames = ["Jan", "Feb", "Mar", "Apr", "May", "June", "July", "Aug",  
"Sept", "Oct", "Nov", "Dec"];
```

DateField.open()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDF.open()
```

Parameters

None.

Returns

Nothing.

Description

Method; opens the pop-up DateChooser subcomponent.

Example

The following code opens the pop-up date chooser of the `df` instance:

```
df.open();
```

DateField.open

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
on(open){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.open = function(eventObject){  
    ...  
}  
myDF.addEventListener("open", listenerObject)
```

Description

Event; broadcast to all registered listeners when a DateChooser subcomponent opens after a user clicks on the icon.

The first usage example uses an `on()` handler and must be attached directly to a `DateField` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the date field `myDF`, sends “_level0.myDF” to the Output panel:

```
on(open){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*myDF*) dispatches an event (in this case, `open`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

This example, written on a frame of the Timeline, sends a message to the Output panel when a date field called `myDF` is opened. The first line of code creates a listener object called `form`. The second line defines a function for the `open` event on the listener object. Inside the function is a `trace()` statement that uses the event object that is automatically passed to the function, in this example `eventObj`, to generate a message. The `target` property of an event object is the component that generated the event—in this example, `myDF`. The `DateField.selectedDate` property is accessed from the event object’s `target` property. The last line calls `EventDispatcher.addEventListener()` from `myDF` and passes it the `open` event and the `form` listener object as parameters.

```
form.open = function(eventObj){
    trace("Pop-up opened and date selected is " +
        eventObj.target.selectedDate) ;
}
myDF.addEventListener("open", form);
```

DateField.pullDown

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDF.pullDown
```

Description

Property (read-only); a reference to the DateChooser component contained by the DateField component. The DateChooser subcomponent is instantiated when a user clicks on the DateField component. However, if the `pullDown` property is referenced before the user clicks on the component, the DateChooser is instantiated and then hidden.

Example

The following example sets the visibility of the DateChooser subcomponent to `false` and then sets the size of the DateChooser subcomponent to 300 pixels high and 300 pixels wide:

```
myDF.pullDown._visible = false;
myDF.pullDown.setSize(300,300);
```

DateField.scroll

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
on(scroll){
    ...
}
```

Usage 2:

```
listenerObject = new Object();
listenerObject.scroll = function(eventObject){
    ...
}
myDF.addEventListener("scroll", listenerObject)
```

Description

Event; broadcast to all registered listeners when a month button is clicked.

The first usage example uses an `on()` handler and must be attached directly to a DateField instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the date field `myDF`, sends “`_level0.myDF`” to the Output panel:

```
on(scroll){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*myDF*) dispatches an event (in this case, *scroll*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The scroll event's event object has an additional property, *detail*, that can have one of the following values: *nextMonth*, *previousMonth*, *nextYear*, *previousYear*.

Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

This example, written on a frame of the Timeline, sends a message to the Output panel when a user clicks a month button on a `DateField` instance called *myDF*. The first line of code creates a listener object called *form*. The second line defines a function for the *scroll* event on the listener object. Inside the function is a `trace()` statement that uses the event object that is automatically passed to the function, in this example *eventObj*, to generate a message. The *target* property of an event object is the component that generated the event—in this example, *myDF*. The last line calls `EventDispatcher.addEventListener()` from *myDF* and passes it the *scroll* event and the *form* listener object as parameters.

```
form = new Object();
form.scroll = function(eventObj){
    trace(eventObj.detail);
}
myDF.addEventListener("scroll", form);
```

DateField.selectableRange

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDF.selectableRange
```

Description

Property; sets a single selectable date or a range of selectable dates. The value of this property is an object that consists of two `Date` objects named *rangeStart* and *rangeEnd*. The *rangeStart* and *rangeEnd* properties designate the boundaries of the selectable date range. If only *rangeStart* is defined, all the dates after *rangeStart* are enabled. If only *rangeEnd* is defined, all the dates before *rangeEnd* are enabled. The default value is *undefined*.

If you want to enable only a single day, you can use a single `Date` object as the value of `selectableRange`.

Specify a full date when you define dates—for example, `new Date(2003, 6, 24)` rather than `new Date()`. If you don't specify a full date, the time returns as `00:00:00`.

The value of `DateField.selectedDate` is set to `undefined` if it falls outside the selectable range.

The values of `DateField.displayedMonth` and `DateField.displayedYear` are set to the nearest last month in the selectable range if the current month falls outside the selectable range. For example, if the current displayed month is August, and the selectable range is from June 2003 to July 2003, the displayed month will change to July 2003.

Example

The following example defines the selectable range as the dates between and including May 7 and June 7:

```
myDF.selectableRange = {rangeStart: new Date(2001, 4, 7), rangeEnd: new Date(2003, 5, 7)};
```

The following example defines the selectable range as the dates after and including May 7:

```
myDF.selectableRange = {rangeStart: new Date(2003, 4, 7)};
```

The following example defines the selectable range as the dates before and including June 7:

```
myDF.selectableRange = {rangeEnd: new Date(2003, 5, 7)};
```

The following example defines the selectable date as June 7 only:

```
myDF.selectableRange = new Date(2003, 5, 7);
```

DateField.selectedDate

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDF.selectedDate
```

Description

Property; a `Date` object that indicates the selected date if that value falls within the value of the `selectableRange` property. The default value is `undefined`.

Example

The following example sets the selected date to June 7:

```
myDF.selectedDate = new Date(2003, 5, 7);
```

DateField.showToday

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myDF.showToday
```

Description

Property; a Boolean value that determines whether the current date is highlighted. The default value is `true`.

Example

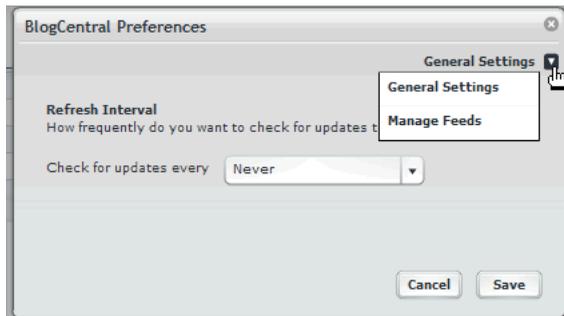
The following example turns off the highlighting on today's date:

```
myDF.showToday = false;
```

DialogBox component

The DialogBox component provides you with a simple dialog box to use in your applications. It also offers a pop-up menu to let you switch among different movie clips within the same dialog box. For example, you could use the DialogBox component to create a Preferences dialog box for your application. The selectable Preferences menu could contain Basic and Advanced preferences.

The following figure shows an example of a Preferences dialog box that can switch to contain the General Settings and Manage Feeds movie clips.



Using the DialogBox component

The DialogBox component uses item objects to build its content. Each item object is an instance of the Object class and contains two properties: label, a string that is the label in the pop-up menu, and data, a string or linkage ID to the content movie clip. These item objects are stored in the data provider for the component. You can access this data provider using:

- `MDialogBox.getDataProvider().someDataProviderAPIMethod();`
- `MDialogBox.setDataProvider(someNewDataProviderInstance);`
- `MDialogBox.someDataProviderAPIMethod();`

The DialogBox component can be used to create a modal or non-modal dialog box. To create a modal dialog box, set the Modal Dialog parameter to `true`, and attach the DialogBox component to the outermost Timeline (effective `_root`).

Use the DialogBox component only when a specific sequence of operations is not required.

DialogBox parameters

You can set the following parameters for each instance of the DialogBox component:

Grow From Sets the origin point from which the dialog box expands. Possible values are `left`, `right`, or `middle`.

Modal Dialog Specifies whether to make the dialog box modal (`true`) or non-modal (`false`). If you set this value to `true`, attach the dialog box to the outermost Timeline (effective `_root`). Modal dialog boxes must be dismissed by the user (for example, by clicking OK, Cancel, or a close button) before focus can shift outside the dialog box. Non-modal dialog boxes let the user shift focus without dismissing the dialog box.

Show Close Button Shows (`true`) or hides (`false`) the close button in the upper right corner of the dialog box.

Gutter Sets the amount of space, in pixels, around the content movie clip.

Single Content Linkage ID Specifies the linkage ID of the content movie clip. Use this parameter only if you have only one movie clip for your dialog box.

Title Sets the title of the DialogBox component, which appears in the title bar.

Labels Specifies a list of item labels for the pop-up menu display.

Data (Linkage IDs) Specifies a list of item linkage IDs for the content movie clips. Use this parameter if you have multiple movie clips for your dialog box.

Minimum Width Sets the minimum width (in pixels) for resize events.

Minimum Height Sets the minimum height (in pixels) for resize events.

About DialogBox states

The DialogBox component has no states.

Method summary for the DialogBox component

The following table summarizes the methods for the MDialogBox component:

Method	Description
<code>MDialogBox.doClose()</code>	Initiates a close event and hides the dialog box.
<code>MDialogBox.doOpen()</code>	Opens a dialog box instance.
<code>MDialogBox.getContent()</code>	Returns a reference to the currently selected content movie clip.
<code>MDialogBox.getDataProvider()</code>	Returns a reference to the current data provider.
<code>MDialogBox.getSelectedContent()</code>	Returns a reference to the currently selected content movie clip.
<code>MDialogBox.getSelectedIndex()</code>	Returns the index of the currently selected item.
<code>MDialogBox.getSelectedItem()</code>	Returns the currently selected item.
<code>MDialogBox.setDataProvider()</code>	Sets the current data provider and updates the screen accordingly.
<code>MDialogBox.setSelectedByData()</code>	Sets the selected item based on a data property
<code>MDialogBox.setSelectedByID()</code>	Sets the selected item based on an ID property
<code>MDialogBox.setSelectedByKey()</code>	Sets the selected item based on a key-value pair match.
<code>MDialogBox.setSelectedByLabel()</code>	Sets the selected item based on a label property.
<code>MDialogBox.setSelectedIndex()</code>	Sets the selected item based on the index number passed in.
<code>MDialogBox.setSelectedItem()</code>	Sets the selected item equal to the item passed in. The item should be retrieved from the data provider.
<code>MDialogBox.setSelectedNextIndex()</code>	Sends the dialog box to the next content item. If the content is at the last item, then the first item appears.
<code>MDialogBox.setSelectedPrevIndex()</code>	Sends the dialog box to the previous content item. If the content is at the first item, then the last item appears.
<code>MDialogBox.setSize()</code>	Sets the width and height of the dialog box and the number of frames it takes to tween to that size.
<code>MDialogBox.setTitle()</code>	Sets the title of the dialog box that appears in the title bar.
<code>MDialogBox.showClose()</code>	Sets whether the close button is shown (<code>true</code>) or hidden (<code>false</code>).

Event summary for the MDialogBox component

The following table summarizes the events for the MDialogBox component.

Event	Description
<code>MDialogBox.onContentChanged()</code>	Executed when the content movie clip in the dialog box changes (in the scope of any listening objects).
<code>MDialogBox.onClosed()</code>	Executed when the dialog box is closed (in the scope of any listening objects).

MDialogBox.doClose()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.doClose()
```

Parameters

None.

Returns

Nothing.

Description

Method; initiates an `onClosed` event and hides the dialog box. This method also cleans up required assets and listeners. The `onClose` event handler for listening objects is triggered before these assets are removed.

This method hides the dialog box by setting `_visible=0`. If you want to completely remove the box from memory, you can call `removeMovieClip()`.

Example

```
myDialog.doClose();
```

MDialogBox.doOpen()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.doOpen()
```

Parameters

None.

Returns

Nothing.

Description

Method; initiates an open event and shows a previously instantiated dialog box instance.

Example

```
myDialog.doOpen();
```

MDialogBox.getContent()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialogBox.getContent()
```

Parameters

None.

Returns

A reference to the movie clip.

Description

Method; gets a reference to the currently selected movie clip.

Tip: You can use either `getContent` or `getSelectedContent` to get a reference to the currently selected content movie clip.

Example

The following example gets a reference to the currently selected movie clip in the `prefsDialog` instance and then calls `doSomeMethod` on it:

```
var gen = prefsDialog.getContent();  
gen.doSomeMethod();
```

MDialogBox.getDataProvider()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.getDataProvider()
```

Parameters

None.

Returns

A reference to the current data provider.

Description

Method; returns a reference to the current data provider for the component instance.

Example

The following example gets the data provider for the `myDialog` instance and adds an item to it:

```
var dp = myDialog.getDataProvider();  
dp.addItem({label:"content1", data:"content1symbol"});  
myDialog.getDataProvider();
```

MDialogBox.getSelectedContent()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.getSelectedContent()
```

Parameters

None.

Returns

A reference to a movie clip.

Description

Method; returns a reference to the currently selected content movie clip.

Tip: You can use either `getContent` or `getSelectedContent` to get a reference to the currently selected content movie clip.

Example

The following example gets a reference to the currently selected movie clip in the `prefsDialog` instance and then calls `doSomeMethod` on it:

```
var gen = prefsDialog.getSelectedContent();
gen.doSomeMethod();
```

MDialogBox.getSelectedIndex()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.getSelectedIndex()
```

Parameters

Nothing.

Returns

The index of the currently selected item.

Description

Method; returns the index of the currently selected item.

Example

The following example creates a variable that is the index of the currently selected item:

```
var dialogIndex = myDialog.getSelectedIndex();
```

MDialogBox.getSelectedItem()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.getSelectedItem()
```

Parameters

None.

Returns

An object.

Description

Method; returns the currently selected item. The item is an object with label and data properties.

Example

The following example creates a variable that is the currently selected item:

```
var item = myDialog.getSelectedItem();
```

MDialogBox.onClosed()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.onClosed()
```

Parameters

None.

Returns

Nothing.

Description

Event handler; executed in the scope of any listening objects. You can override this event handler on a component instance.

Example

The following code overrides the `onClosed` event handler on the component instance `myDialog`:

```
myDialog.onClosed = function()  
{  
    trace("Dialog closed at:" this);  
}
```

MDialogBox.onContentChanged()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.onContentChanged()
```

Parameters

None.

Returns

Nothing.

Description

Event handler; executed in the scope of any listening objects.

Example

The following example traces the content that has changed:

```
var obj = new Object();
obj.controller = this;
obj.onContentChanged = function(ref)
{
    var content = ref.getSelectedContent();
    trace("Dialog changed with: " +content);
}
myDialog.addListener(obj);
```

MDialogBox.setDataProvider()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.setDataProvider(dp)
```

Parameters

dp The data provider.

Returns

Nothing.

Description

Method; sets the data provider for the component. The data provider can be an array or an object of the `DataProvider` class.

The `DialogBox` component uses item objects to build its content. Each item object is an instance of the `Object` class and contains two properties: `label`, a string that is the label in the pop-up menu, and `data`, a string or linkage ID to the content movie clip.

The `DataProvider` class included with the Macromedia Central SDK contains new methods. For more information about the `DataProvider` class and its methods, see “`Central.DataProviderClass` object” in *Developing Central Applications*.

Example

The following code sets the data provider for a Preferences dialog box that has two selectable screens, General and Notices:

```
var dp = new mx.central.data.DataProviderClass();
dp.addItem({label:"Notices", data: "NoticesMovie"});
dp.addItem({label:"General", data: "GeneralMovie"});
myDialog.setDataProvider(dp);
```

MDialogBox.setSelectedByData()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.setSelectedByData(data);
```

Parameters

data A string; the data property of the selected item.

Returns

Nothing.

Description

Method; sets the selected item based on a data property.

Example

The following code sets the selected item to the data property, `prefs`, for the Preferences menu:

```
myDialog.setSelectedByData("prefs");
```

MDialogBox.setSelectedByID()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.setSelectedByID(linkage)
```

Parameters

linkage The linkage ID of the item.

Returns

Nothing.

Description

Method; sets the selected item based on its linkage ID property.

Example

The following code sets the selected item to the item with the linkage ID "prefs":

```
myDialog.setSelectedByID("prefs");
```

MDialogBox.setSelectedByKey()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.setSelectedByKey(label, data)
```

Parameters

label The label of the item.

data The data associated with the item.

Returns

Nothing.

Description

Method; sets the selected item by its key, which is composed of the label and its associated data.

Example

The following code sets the selected item to the Preferences item in the pop-up menu:

```
myDialog.setSelectedByKey("Preferences", "prefs");
```

MDialogBox.setSelectedByLabel()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.setSelectedByLabel(label)
```

Parameters

label The label of the item.

Returns

Nothing.

Description

Method; sets the selected item by its label.

Example

The following code sets the item with label "My Feeds" as selected:

```
myDialog.setSelectedByLabel("My Feeds");
```

MDialogBox.setSelectedIndex()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.setSelectedIndex(index)
```

Parameters

index The index of the item.

Returns

Nothing.

Description

Method; sets the item indicated by *index* as selected.

Example

The following code sets the third item in the dialog box's pop-up menu as selected:

```
myDialog.setSelectedIndex(2);
```

MDialogBox.setSelectedItem()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.setSelectedItem(item)
```

Parameters

item The item to be set to the selected state.

Returns

Nothing.

Description

Method; sets the item specified in the *item* parameter to the selected state. The item should be retrieved from the data provider.

Example

The following code sets George as the selected item:

```
myDialog.setSelectedItem({label: "George"});
```

MDialogBox.setSelectedNextIndex()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.setSelectedNextIndex()
```

Parameters

None.

Returns

Nothing.

Description

Method; changes the dialog box content to the next content item. If the current content is the last item, then this method changes the dialog box to the first content item.

Example

A pop-up menu in the dialog box contains three items: A, B, and C. The current content item is B. The following code changes the content to C:

```
myDialog.setSelectedNextIndex();
```

MDialogBox.setSelectedPrevIndex()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.setSelectedPrevIndex()
```

Parameters

None.

Returns

Nothing

Description

Method; changes the dialog box content to the previous content item. If the current content is the first item, then this method changes the dialog box to the last content item.

Example

A pop-up menu in the dialog box `dialog1` contains three items: A, B, and C. The current content item is B. The following code changes the content to A:

```
dialog1.setSelectedPrevIndex();
```

MDialogBox.setSize()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialogBox.setSize(width, height, tween)
```

Parameters

width The width in pixels.

height The height in pixels.

tween The number of frames it takes to animate (*tween*) to the size specified.

Returns

Nothing.

Description

Method; sets the width and height of the dialog box and the number of frames it takes to tween to that size.

Example

The following example makes a dialog box 300 pixels wide and 200 pixels high. It takes five frames to tween to the dialog box's full size.

```
myDialogBox.setSize(300,200,5);
```

MDialogBox.setTitle()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.setTitle(title)
```

Parameters

title A string.

Returns

Nothing.

Description

Method; sets the title of the dialog box, which appears in the title bar.

Example

The following code sets the title of the dialog box to "My New App Preferences":

```
myAppDialog.setTitle("My New App Preferences");
```

MDialogBox.showClose()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myDialog.showClose(close)
```

Parameters

close A Boolean value.

Returns

Nothing.

Description

Method; sets whether the close button (X) in the upper right corner of the dialog box is visible (`true`) or hidden (`false`). If you hide the close button, be sure that your code provides logic to let the user close the dialog box, using an OK or Cancel button, for example.

Example

The following code shows the close button in the dialog box:

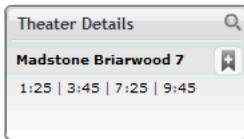
```
dialog2.showClose("true");
```

ExpandingPod component

The ExpandingPod component provides a box that presents a limited view of information and lets users quickly open an expanded view by clicking a magnifying glass icon. When the user clicks the magnifying glass icon, the original box tweens to an expanded view and displays new content. This component lets you present teaser information in a limited space and then lets the user expand the content area to view the full content in a larger space. You need to allow only enough space for the collapsed size in your application.

The component provides a title bar, a magnifying glass icon, and a content area.

The following figures show an ExpandingPod component in its collapsed and expanded views.



Using the ExpandingPod component

The content for the ExpandingPod component is contained in two movie clips: one for the collapsed view and one for the expanded view. For example, the collapsed view might show the face of a movie star, and the expanded view might show a full-body shot of the movie star from a scene in a recent action movie as well as text describing the movie and a link for purchasing tickets to a show. In another example, the collapsed view might show a summary of information about a product or service, and the expanded view might show a request form for more information, which the user can fill out and submit.

The component automatically sizes itself to fit your content movie clips. You can show or hide the title bar of the box that contains information, and you can use the default magnifying glass icon or a custom icon.

ExpandingPod parameters

You can set the following parameters for each instance of the ExpandingPod component:

Minimized Content is the symbol ID of the movie clip for the content in the collapsed view.

Expanded Content is the symbol ID of the movie clip for the content in the expanded view.

Grow From is the direction in which the component expands when the user clicks the magnifying glass icon. Possible values are `left`, `right`, and `middle`.

About ExpandingPod states

The ExpandingPod component has the following states: collapsed and expanded. This component does not use the green borders to reflect change of state, unlike some other components. The user's mouse indicator changes from an arrow to a pointer when the it moves over the magnifier. The magnifying glass icon can be shown or hidden.

The component can be set to expand from the left, right, or middle when the user clicks the magnifying glass icon.

Method summary for the MExpandingPod component

The following table summarizes the methods for the MExpandingPod component:

Method	Description
<code>MExpandingPod.getContentStart()</code>	Gets the x and y coordinates of where the content should start (top left).
<code>MExpandingPod.getExpanded()</code>	Gets whether the component is expanded (<code>true</code>) or collapsed (<code>false</code>).
<code>MExpandingPod.getLargeContent()</code>	Gets a reference to the large content movie clip for the expanded view.
<code>MExpandingPod.getSize()</code>	Returns an object with the width and height (in pixels) properties of the component.
<code>MExpandingPod.getSmallContent()</code>	Gets a reference to the small content movie clip for the collapsed view.

Method	Description
<code>MExpandingPod.setExpanded()</code>	Sets whether the pod is expanded (<code>true</code>) or collapsed (<code>false</code>).
<code>MExpandingPod.setExpandingOrigin()</code>	Sets the direction in which the pod expands.
<code>MExpandingPod.setLargeContent()</code>	Sets the linkage of the large movie clip to appear in the content area in the expanded state.
<code>MExpandingPod.setSmallContent()</code>	Sets the linkage of the small movie clip to appear in the content area in the collapsed state.
<code>MExpandingPod.setTitle()</code>	Sets the title for the content so that it is visible to the user.
<code>MExpandingPod.showMagnifier()</code>	Shows or hides the magnifying glass icon.
<code>MExpandingPod.showTitleBar()</code>	Shows or hides the text in the title bar above the content area.

MExpandingPod.getContentStart()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.getContentStart()
```

Parameters

None.

Returns

An object with *x* and *y* properties.

Description

Method; gets the *x* and *y* coordinates of where the content starts.

Example

The following example creates a variable that contains the *x* and *y* coordinates of the location where the content specified in `setContentStart` begins:

```
var contentLoc = myComponent.getContentStart();
var myMovie = attachMovie("movie","myMovie", 1);
myMovie._x = contentLoc._x;
myMovie._y = contentLoc._y;
```

MExpandingPod.getExpanded()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.getExpanded()
```

Parameters

None.

Returns

A Boolean value: `true` if the component instance is in expanded view; `false` if it is collapsed.

Description

Method; gets a Boolean value that indicates whether the component instance is in expanded view (`true`) or collapsed view (`false`).

Example

The following example gets the current view of the component and traces it:

```
var expandedView = myExpandingPod.getExpanded();  
trace(expandedView);
```

MExpandingPod.getLargeContent()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.getLargeContent()
```

Parameters

None.

Returns

A reference to the large content movie clip.

Description

Method; gets a reference to the large content movie clip, which is shown when the component is expanded.

Example

The following example traces the symbol ID of the large content movie clip shown in expanded view:

```
trace(myExpandingPod.getLargeContent());
```

MExpandingPod.getSize()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.getSize()
```

Parameters

None.

Returns

Returns an object with width and height properties.

Description

Method; gets the width and height (in pixels) of the component.

Example

The following example traces the size of the `myComponent` instance:

```
var size = myComponent.getSize();  
trace(size.height + ", " + size.width);
```

MExpandingPod.getSmallContent()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.getSmallContent()
```

Parameters

None.

Returns

A reference to the small content movie clip.

Description

Method; gets a reference to the small content movie clip, which is shown when the component is in collapsed view.

Example

The following example gets the reference to the small content movie clip:

```
var smallContent = myComponent.getSmallContent();  
trace(smallContent);
```

MExpandingPod.setExpanded()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.setExpanded(expanded)
```

Parameters

expanded A Boolean value: `true` expands the component to contain the large content movie clip; `false` collapses it to contain the small content movie clip.

Returns

Nothing.

Description

Method; sets whether the component is expanded.

Example

The following example sets the component to its expanded view:

```
myComponent.setExpanded(true);
```

MExpandingPod.setExpandingOrigin()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myExpandingPod.setExpandingOrigin(direction)
```

Parameters

direction A string indicating the direction in which the pod expands. Possible values are `left`, `right`, or `middle`.

Returns

Nothing.

Description

Method; sets the direction in which the pod expands when the user clicks the magnifying glass icon in the pod's title bar. The direction you select might depend on where the pod is placed in your application. For example, if the pod is set in the bottom right corner of the application window, you might want the pod to expand to the left.

Example

The following code sets the `moviePod` instance to expand to the left.

```
moviePod.setExpandingOrigin("left");
```

MExpandingPod.setLargeContent()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myPod.setLargeContent(linkageID)
```

Parameters

linkage ID The linkage ID of the large content movie clip.

Returns

Nothing.

Description

Method; sets the linkage ID of the large content movie clip, which appears when the component is expanded. The user clicks the magnifying glass icon to expand the component. The component is automatically resized to accommodate the content.

Example

The following example sets the movie clip for the expanded state to `"formView"`:

```
myPod.setLargeContent("formView");
```

See also

[MExpandingPod.setSmallContent\(\)](#)

MExpandingPod.setSmallContent()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myPod.setSmallContent(linkageID)
```

Parameters

linkage ID The linkage ID of the small content movie clip.

Returns

Nothing.

Description

Method; sets the linkage ID of the small content movie clip, which is shown when the component is collapsed. The user clicks the magnifying glass icon to collapse the component. The component automatically resizes to accommodate the content.

Example

The following example sets the movie clip for the collapsed state to "formView":

```
myPod.setLargeContent("formView");
```

See also

[MExpandingPod.setLargeContent\(\)](#)

MExpandingPod.setTitle()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.setTitle(title)
```

Parameters

title A string; the title that appears in the title bar of the component. The maximum number of characters is determined by the width of the component.

Returns

Nothing.

Description

Method; sets the text for the title that appears in the title bar of the component. This title is visible to the user if you use the `MExpandingPod.showTitleBar()` method to show the title bar.

Example

The following example sets the title of the component instance to “Featured movie of the week”:

```
myComponent.setTitle("Featured movie of the week");
```

MExpandingPod.showMagnifier()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.showMagnifier(show)
```

Parameters

show A Boolean value: `true` shows the magnifying glass icon; `false` hides it.

Returns

Nothing.

Description

Method; lets you show or hide the magnifying glass icon in the upper right corner of the component. Users click the magnifying glass icon to expand or collapse the content area.

If you hide this icon, the user cannot expand or collapse the instance of the `ExpandingPod` component.

Example

The following example shows the magnifying glass icon:

```
myComponent.showMagnifier(true);
```

MExpandingPod.showTitleBar()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myComponent.showTitleBar(show)
```

Parameters

show A Boolean value: `true` shows the text in the title bar above the content area; `false` hides it.

Returns

Nothing.

Description

Method; lets you show or hide the text in the title bar above the content area. You set the text for the title bar by using `MExpandingPod.setTitle()` on page 229.

Example

The following example shows the text in the title bar:

```
myComponent.showTitleBar(true);
```

FocusManager class

You can use the Focus Manager to specify the order in which components receive focus when a user presses the Tab key to navigate in an application. You can also use the Focus Manager to set a button in your document that receives keyboard input when a user presses Enter (Windows) or Return (Macintosh). For example, when users fill out a form, they should be able to tab between fields and press Enter (Windows) or Return (Macintosh) to submit the form.

All components implement Focus Manager support; you don't need to write code to invoke it.

The Focus Manager interacts with the System Manager, which activates and deactivates FocusManager instances as pop-up windows are activated or deactivated. Each modal window has an instance of FocusManager so the components in that window become their own tab set, preventing the user from tabbing into components in other windows.

The Focus Manager recognizes groups of radio buttons (those with a defined `RadioButton.groupName` property) and sets focus to the instance in the group that has a `selected` property that is set to `true`. When the Tab key is pressed, the Focus Manager checks to see if the next object has the same group name as the current object. If it does, it automatically moves focus to the next object with a different group name. Other sets of components that support a `groupName` property can also use this feature.

The Focus Manager handles focus changes caused by mouse clicks. If the user clicks a component, that component is given focus.

Using the Focus Manager

The Focus Manager does not automatically assign focus to a component. You must write a script that calls `FocusManager.setFocus()` on a component if you want a component to have focus when an application loads.

Note: If you call `FocusManager.setFocus()` to set focus to a component when an application loads, the focus ring does not appear around that component. The component has focus, but the indicator is not present.

To create focus navigation in an application, set the `tabIndex` property on any objects (including buttons) that should receive focus. When a user presses the Tab key, the Focus Manager looks for an enabled object with a `tabIndex` property that is higher than the current value of `tabIndex`. Once the Focus Manager reaches the highest `tabIndex` property, it returns to zero. So, in the following example, the `comment` object (probably a `TextArea` component) receives focus first, and then the `okButton` object receives focus:

```
comment.tabIndex = 1;
okButton.tabIndex = 2;
```

You can also use the Accessibility panel to assign a tab index value.

If nothing on the Stage has a tab index value, the Focus Manager uses the *depth* (stacking order, or *z-order*). The depth is 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 depth.

To create a button that receives focus when a user presses Enter (Windows) or Return (Macintosh), set the `FocusManager.defaultPushButton` property to the instance name of the desired button, as shown here:

```
focusManager.defaultPushButton = okButton;
```

Note: The Focus Manager is sensitive to when objects are placed on the Stage (the depth order of objects) and not their relative positions on the Stage. This is different from the way Flash Player handles tabbing.

Using the Focus Manager to allow tabbing

You can use the Focus Manager to create a scheme that allows users to press the Tab key to cycle through objects in a Flash application. (Objects in the tab scheme are called *tab targets*.) The Focus Manager examines the `tabEnabled` and `tabChildren` properties of the objects' parents in order to locate the objects.

A movie clip can be either a container of tab targets, a tab target itself, or neither:

Movie clip type	tabEnabled	tabChildren
Container of tab targets	false	true
Tab target	true	false
Neither	false	false

Note: This is different from the default Flash Player behavior, in which a container's `tabChildren` property can be `undefined`.

Consider the following scenario. On the Stage of the main Timeline are two text fields (`txt1` and `txt2`) and a movie clip (`mc`) that contains a `DataGrid` component (`grid1`) and another text field (`txt3`). You would use the following code to allow users to press Tab and cycle through the objects in the following order: `txt1`, `txt2`, `grid1`, `txt3`.

Note: The `FocusManager` and `TextField` instances are enabled by default.

```
// let Focus Manager know mc has children;
// this overrides mc.focusEnabled=true;
```

```

mc.tabChildren=true;
mc.tabEnabled=false;
// set the tabbing sequence
txt1.tabIndex = 1;
txt2.tabIndex = 2;
mc.grid1.tabIndex = 3;
mc.txt3.tabIndex = 4;

// set initial focus to txt1
txt1.text = "focus";
focusManager.setFocus(txt1);

```

If your movie clip doesn't have an `onPress` or `onRelease` method or a `tabEnabled` property, it won't be seen by the Focus Manager unless you set `focusEnabled` to `true`. Input text fields are always in the tab scheme unless they are disabled.

If a Flash application is playing in a web browser, the application doesn't have focus until a user clicks somewhere in the application. Also, once a user clicks in the Flash application, pressing Tab can cause focus to jump outside the Flash application. To keep tabbing limited to objects inside the Flash application in Flash Player 7 ActiveX control, add the following parameter to the HTML `<object>` tag:

```
<param name="SeamlessTabbing" value="false"/>
```

Creating an application with the Focus Manager

The following procedure creates a focus scheme in a Flash application.

To create a focus scheme:

1. Drag the `TextInput` component from the Components panel to the Stage.
2. In the Property inspector, assign it the instance name **comment**.
3. Drag the `Button` component from the Components panel to the Stage.
4. In the Property inspector, assign it the instance name **okButton** and set the label parameter to **OK**.
5. In Frame 1 of the Actions panel, enter the following:

```

comment.tabIndex = 1;
okButton.tabIndex = 2;
focusManager.setFocus(comment);
function click(evt){
    trace(evt.type);
}
okButton.addEventListener("click", this);

```

This code sets the tab ordering. Although the `comment` field doesn't have a focus ring, it has initial focus, so you can start typing in the `comment` field without clicking on it.

Customizing the Focus Manager

You can change the color of the focus ring in the Halo theme by changing the value of the `themeColor` style, as in this example:

```
_global.style.setStyle("themeColor", "haloBlue");
```

The Focus Manager uses a `FocusRect` skin for drawing focus. This skin can be replaced or modified and subclasses can override `UIComponent.drawFocus` to draw custom focus indicators.

FocusManager class (API)

Inheritance `MovieClip` > [UIObject class](#) > [UIComponent class](#) > `FocusManager`

ActionScript Class Name `mx.managers.FocusManager`

You can use the Focus Manager to specify the order in which components receive focus when a user presses the Tab key to navigate in an application. You can also use the `FocusManager` class to set a button in your document that receives keyboard input when a user presses Enter (Windows) or Return (Macintosh).

Tip: In a class file that inherits from `UIComponent`, it is not good practice to refer to `_root.focusManager`. Every `UIComponent` instance inherits a `getFocusManager()` method, which returns a reference to the `FocusManager` instance responsible for controlling that component's focus scheme.

Method summary for the FocusManager class

The following table lists the methods of the `FocusManager` class.

Method	Description
FocusManager.getFocus()	Returns a reference to the object that has focus.
FocusManager.sendDefaultPushButtonEvent()	Sends a <code>click</code> event to listener objects registered to the default push button.
FocusManager.setFocus()	Sets focus to the specified object.

Methods inherited from the UIObject class

The following table lists the methods the `FocusManager` class inherits from the `UIObject` class.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it will be redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.

Method	Description
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the `UIComponent` class

The following table lists the methods the `FocusManager` class inherits from the `UIComponent` class.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the `FocusManager` class

The following table lists the properties of the `FocusManager` class.

Property	Description
<code>FocusManager.defaultPushButton</code>	The object that receives a <code>click</code> event when a user presses the Return or Enter key.
<code>FocusManager.defaultPushButtonEnabled</code>	Indicates whether keyboard handling for the default push button is turned on (<code>true</code>) or off (<code>false</code>). The default value is <code>true</code> .
<code>FocusManager.enabled</code>	Indicates whether tab handling is turned on (<code>true</code>) or off (<code>false</code>). The default value is <code>true</code> .
<code>FocusManager.nextTabIndex</code>	The next value of the <code>tabIndex</code> property.

Properties inherited from the `UIObject` class

The following table lists the properties the `FocusManager` class inherits from the `UIObject` class.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.

Property	Description
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `FocusManager` class inherits from the `UIComponent` class.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the `FocusManager` class

There are no events exclusive to the `FocusManager` class.

Events inherited from the `UIObject` class

The following table lists the events the `FocusManager` class inherits from the `UIObject` class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the FocusManager class inherits from the UIComponent class.

Event	Description
UIComponent.focusIn	Broadcast when an object receives focus.
UIComponent.focusOut	Broadcast when an object loses focus.
UIComponent.keyDown	Broadcast when a key is pressed.
UIComponent.keyUp	Broadcast when a key is released.

FocusManager.defaultPushButton

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004 and Flash MX Professional 2004.

Usage

```
focusManager.defaultPushButton
```

Description

Property; specifies the default push button for an application. When the user presses Enter (Windows) or Return (Macintosh), the listeners of the default push button receive a `click` event. The default value is `undefined` and the data type of this property is `object`.

The Focus Manager uses the emphasized style declaration of the `SimpleButton` class to visually indicate the current default push button.

The value of the `defaultPushButton` property is always the button that has focus. Setting the `defaultPushButton` property does not give initial focus to the default push button. If there are several buttons in an application, the button that currently has focus receives the `click` event when Enter or Return is pressed. If some other component has focus when Enter or Return is pressed, the `defaultPushButton` property is reset to its original value.

Example

The following code sets the default push button to the `OKButton` instance:

```
focusManager.defaultPushButton = OKButton;
```

See also

[FocusManager.defaultPushButtonEnabled](#),
[FocusManager.sendDefaultPushButtonEvent\(\)](#)

FocusManager.defaultPushButtonEnabled

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
focusManager.defaultPushButtonEnabled
```

Description

Property; a Boolean value that determines if keyboard handling of the default push button is turned on (`true`) or not (`false`). Setting `defaultPushButtonEnabled` to `false` allows a component to receive the Return or Enter key and handle it internally. You must re-enable default push button handling by watching the component's `onKillFocus()` method (see `MovieClip.onKillFocus` in *Flash ActionScript Language Reference*) or `focusOut` event. The default value is `true`.

This property is for use by advanced component developers.

Example

The following code disables default push button handling:

```
focusManager.defaultPushButtonEnabled = false;
```

FocusManager.enabled

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
focusManager.enabled
```

Description

Property; a Boolean value that determines if tab handling is turned on (`true`) or not (`false`) for a particular group of focus objects. (For example, another pop-up window could have its own Focus Manager.) Setting `enabled` to `false` allows a component to receive the tab handling keys and handle them internally. You must re-enable the Focus Manager handling by watching the component's `onKillFocus()` method (see `MovieClip.onKillFocus` in *Flash ActionScript Language Reference*) or `focusOut` event. The default value is `true`.

Example

The following code disables tabbing:

```
focusManager.enabled = false;
```

FocusManager.setFocus()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004 and Flash MX Professional 2004.

Usage

```
focusManager.setFocus()
```

Parameters

None.

Returns

A reference to the object that has focus.

Description

Method; returns a reference to the object that currently has focus.

Example

The following code sets the focus to `myOKButton` if the object that currently has focus is `myInputText`:

```
if (focusManager.setFocus() == myInputText)
{
    focusManager.setFocus(myOKButton);
}
```

See also

[FocusManager.setFocus\(\)](#)

FocusManager.nextTabIndex

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
FocusManager.nextTabIndex
```

Description

Property; the next available tab index number. Use this property to dynamically set an object's `tabIndex` property.

Example

The following code gives the `mycheckbox` instance the next highest `tabIndex` value:

```
mycheckbox.tabIndex = focusManager.nextTabIndex;
```

See also

[UIComponent.tabIndex](#)

FocusManager.sendDefaultPushButtonEvent()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004 and Flash MX Professional 2004.

Usage

```
focusManager.sendDefaultPushButtonEvent()
```

Parameters

None.

Returns

Nothing.

Description

Method; sends a `click` event to listener objects registered to the default push button. Use this method to programmatically send a `click` event.

Example

The following code triggers the default push button `click` event and fills in the user name and password fields when a user selects the `CheckBox` instance `chb` (the check box would be labeled “Automatic Login”):

```
name_txt.tabIndex = 1;
password_txt.tabIndex = 2;
chb.tabIndex = 3;
submit_ib.tabIndex = 4;

focusManager.defaultPushButton = submit_ib;

chbObj = new Object();
chbObj.click = function(o){
    if (chb.selected == true){
        name_txt.text = "Jody";
        password_txt.text = "foobar";
        focusManager.sendDefaultPushButtonEvent();
    } else {
        name_txt.text = "";
        password_txt.text = "";
    }
}
```

```
    }  
  }  
  chb.addEventListener("click", chbObj);  
  
  submitObj = new Object();  
  submitObj.click = function(o){  
    if (password_txt.text != "foobar"){  
      trace("error on submit");  
    } else {  
      trace("Yeah! sendDefaultPushButtonEvent worked!");  
    }  
  }  
  submit_ib.addEventListener("click", submitObj);
```

See also

[FocusManager.defaultPushButton](#)

FocusManager.setFocus()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004 and Flash MX Professional 2004.

Usage

```
focusManager.setFocus(object)
```

Parameters

object A reference to the object to receive focus.

Returns

Nothing.

Description

Method; sets focus to the specified object. If the object to which you want to set focus is not on the main Timeline, use the following code:

```
_root.focusManager.setFocus(object);
```

Example

The following code sets focus to myOKButton:

```
focusManager.setFocus(myOKButton);
```

See also

[FocusManager.getFocus\(\)](#)

IconButton component

The IconButton component lets you create a simple button with a choice of icons or a custom icon.

The following images show various icon buttons that you can create using the IconButton component: (from left to right) Print, Set Notices, Remove from Favorites, and Toss buttons.



Using the IconButton component

Icon buttons look as if you can press them, and each button has an icon on its face. An icon button performs an action when the user clicks it.

The IconButton component offers the following icon buttons, which you can use in your application's interface:

- Close button (`icon_close`)
Use this button as a close button in your application.
- Print button (`icon_print`)
Use this button to let the user print the current content appearing in the application.
- Toss button (`icon_toss-to`)
Use this button to let the user start the application's pod in the Console, that is, to send data from the application to the pod.
- Add to Favorites button (`icon_bookmark_add`)
Use this button to let the user add the current selection to the application tab used for caching data, such as the Favorites tab. This button contains a bookmark icon to reflect the bookmark icon you use to visually identify the Favorites tab.
- Remove from Favorites button (`icon_bookmark_remove`)
Use this button to let the user remove the current selection from the application tab used for caching data, such as the Favorites tab. This button contains a bookmark icon to reflect the bookmark icon you use to visually identify the Favorites tab.
- Set Notices icon button (`icon_alert`)
Use this button to open a dialog box (which you can create with the Window component) where users can select criteria for notices they would like to receive. For example, a weather application could send a notice when the temperature drops below a certain degree. Users would click the Set Notices icon button to access a dialog box that lets them specify that temperature.

Note: These buttons are different from the images found in the Central artwork MXP. These buttons are intended for use in your application, not your pod. For more information on elements you can use in your pods, see the [“CloseButton component” on page 95](#), the [“TossButton component” on page 476](#) and the [“Macromedia Central artwork” on page 10](#).

IconButton parameters

You can set the following parameters for each instance of the IconButton component:

Built-In Icon has the following available values: `icon_print`, `icon_toss-to`, `icon_bookmark_add`, `icon_bookmark_remove`, `icon_alert`, or `Custom`. If you select `Custom`, provide the symbol ID in the following parameter, `Custom Icon`.

Custom Icon is the symbol ID of the image file to use for the icon; for example, `mylogo.jpg`.

Change Handler is the name of a function to call when the state of the button changes.

You can set additional options and functionality for instances of this component by using its methods.

About IconButton states and variations

The IconButton component has the following states: `active`, `rollover`, `press`, `default`, `default rollover`, and `disabled`. The component in all states has a green border to indicate its status to the user, except for the `active` and `disabled` states. The `disabled` button is dimmed and is unavailable to the user.

Method summary for the MIconButton component

The following table summarizes the methods for the MIconButton class:

Method	Description
<code>MIconButton.setEnabled()</code>	Returns <code>true</code> if enabled; <code>false</code> if disabled.
<code>MIconButton.setIcon()</code>	Returns an instance of the icon inside the button.
<code>MIconButton.setChangeHandler()</code>	Assigns a function that is called every time the push button is released (toggle state is <code>false</code>).
<code>MIconButton.setEnabled()</code>	Enables the push button.
<code>MIconButton.setIcon()</code>	Sets the icon that appears in the push button.

MIconButton.setEnabled()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconButton.setEnabled()
```

Parameters

None.

Returns

A Boolean value: `true` indicates that the `IconButton` instance is enabled; `false` indicates that the `IconButton` instance is disabled.

Description

Method; indicates whether the `IconButton` instance is enabled or disabled.

Example

The following example returns the enabled state of the `iconButton1` instance to the Output panel:

```
trace(iconButton1.getEnabled());
```

`MIconButton.getIcon()`

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconButton.getIcon()
```

Parameters

None.

Returns

A reference to the movie clip in the `IconButton` component.

Description

Method; returns an instance of the icon appearing in the icon button.

Example

The following example retrieves a reference to the movie clip inside the `iconButton1` object, stores it in a variable, and sets the rotation property to 45:

```
var icon = iconButton1.getIcon();  
icon._rotation=45;
```

`MIconButton.setChangeHandler()`

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconButton.setChangeHandler(callback)
```

Parameters

callback The string name of the function that is called. The function that it calls should reside on the same Timeline as the IconButton component.

Returns

Nothing.

Description

Method; specifies a change handler function to call when the icon button is released. The function always accepts the instance of the component that has changed as a parameter. Calling this method overrides the `Change Handler` parameter value specified in the Property inspector.

Example

The following example sets a Change Handler for the `iconButton1` object:

```
iconButton1.setChangeHandler("On");
```

MIconButton.setEnabled()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconButton.setEnabled(state)
```

Parameters

state A Boolean value: enables the button (`true`) or disables it (`false`).

Returns

Nothing.

Description

Method; specifies whether the IconButton is enabled (`true`) or disabled (`false`). If an IconButton instance is disabled, it does not accept mouse or keyboard interaction from the user. If you omit this parameter, the default for the method is `true`.

Example

The following example disables the IconButton component:

```
myIconButton.setEnabled(false);
```

MIconButton.setIcon()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconButton.setIcon(linkage)
```

Parameters

linkage A string listing the linkage name or instance reference of the movie clip target.

Returns

Nothing.

Description

Method; sets the icon that appears on the face of the button. Possible values for the *linkage* parameter are as follows: `icon_close`, `icon_print`, `icon_toss-to`, `icon_bookmark_add`, `icon_bookmark_remove`, `icon_alert`. For more information about how to use these icons in your application, see [“Using the IconButton component” on page 242](#).

If the icon movie clip is composed of frames labeled “_up,” “_over,” “_down,” and “disabled,” the content of each frame appears with the corresponding button state. Calling this method overrides the Icon parameter value set in authoring.

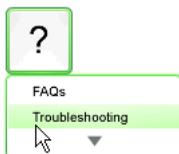
Example

The following example applies the icon movie clip in the library with the specified linkage:

```
iconButton1.setIcon("foo");
```

IconMenu component

The IconMenu component combines an icon button with a multiselect ListBox component. When users click the menu button, they can select items from a pop-up menu.



The IconMenu component

Using the IconMenu component

Use the IconMenu component to present a list of commands in a pop-up menu when space is limited to the size of the button. You can use any icon for the button, and you can change the width and height of the button that the users click to display the menu.

If the menu contains more items than are viewable, a scroll arrow appears at the bottom of the list. You can move the mouse pointer toward the arrow to scroll the list and display the other items. As you scroll through the list, a scroll arrow appears at the top of the list. When you reach the bottom of the list, the bottom scroll arrow disappears. You can make menu items scroll faster by moving the mouse pointer farther below or above the scroll arrows.

If you turn on check marks by using the `showCheckmarks` method, a check mark appears next to an item that the user selects.

The `MIconMenu.setDataProvider()` method lets you set a data provider for the menu. A data provider can be an array or an instance of the `DataProvider` class. For more information about arrays, see the Help system in the Flash authoring tool. For more information about the `DataProvider` class, see *Developing Central Applications*.

IconMenu parameters

You can set the following parameters for each instance of the `IconMenu` component:

Icon is the path to the image file to use on the button; for example, `logo.jpg`.

Label is the text label that appears on the button.

Label Placement is the position for the label, to the left or right of the button, or at the top or bottom of the button.

Menu Labels is an array of text strings specifying the items in the pop-up menu. Enter the text strings for the array using the Values dialog box or using the `MIconMenu.addItem` or `MIconMenu.addItemAt` methods to add items at runtime.

Data is an array of text strings specifying the values associated with the items (labels) in the pop-up menu. Enter the text strings for the array using the Values dialog box or using the `MIconMenu.addItem` or `IconMenu.addItemAt` methods to add items at runtime.

Change Handler is the name of the function that you call when the user selects an item in the pop-up menu. You must define this function in the same Timeline as the instance of the `IconMenu`. This parameter is optional and must be specified only if you want an action to occur when the user selects an item in the pop-up menu. For more information, see [“Writing event listeners for components” on page 11](#).

Check Marks lets you show check marks next to items.

Row Count indicates the number of rows, or how many items appear, in the menu.

You can set additional options and functionality for `IconMenu` instances by using the methods of the `IconMenu` component.

About IconMenu states and variations

The `IconMenu` component has the following states: disabled, enabled, rollover, pressed, focused, clicked, and scrolling. In the clicked and scrolling states, the menu appears.

`IconMenu` also offers two optional variations for the menu. Items in the menu can appear with check marks to the left, indicating a default selection. Items can also appear as disabled, or dimmed, so that the user cannot select them.

Method summary for the `MIconMenu` component

The following table summarizes the methods for the `MIconMenu` class:

Method	Description
<code>MIconMenu.addItem()</code>	Adds a new item to the menu.
<code>MIconMenu.addItemAt()</code>	Adds a new item to the menu at the position indicated by the index value.
<code>MIconMenu.clearChecked()</code>	Turns off check marks for any selected items in the menu.
<code>MIconMenu.clearDisabled()</code>	Re-enables all items in the menu that were previously disabled using <code>setEnabledIndices()</code> .
<code>MIconMenu.getCheckedIndices()</code>	Returns an array of indexes for items that have a check mark next to them.
<code>MIconMenu.getCheckedItems()</code>	Returns an array of objects with label and data properties for items in the menu that have check marks next to them.
<code>MIconMenu.getCheckmarks()</code>	Returns <code>true</code> if check marks are turned on for the menu. Returns <code>false</code> if check marks are turned off.
<code>MIconMenu.getDisabledIndices()</code>	Returns an array of indexes corresponding to items in the menu that have been disabled using <code>setEnabledIndices()</code> .
<code>MIconMenu.getEnabled()</code>	Returns <code>true</code> if the <code>MIconMenu</code> is enabled; <code>false</code> if the <code>MIconMenu</code> is disabled.
<code>MIconMenu.getIcon()</code>	Returns the name of the symbol in the library that is the icon for the <code>MIconMenu</code> component.
<code>MIconMenu.getItemAt()</code>	Returns an object containing label and data properties for the item in the menu indicated by the index.
<code>MIconMenu.getItemID()</code>	Returns the value of the <code>item._ID_</code> property found at <code>index</code> .
<code>MIconMenu.getLabel()</code>	Retrieves the label of the button used to toggle the pop-up menu.
<code>MIconMenu.getLabelPlacement()</code>	Retrieves the position of the label of the icon menu toggle button.
<code>MIconMenu.getLength()</code>	Returns the number of items in the menu.
<code>MIconMenu.getRowCount()</code>	Returns the number of rows visible in the menu.
<code>MIconMenu.getSelectedIndex()</code>	Returns the index of the most recently selected item in the menu.
<code>MIconMenu.getSelectedItem()</code>	Returns an object with label and data properties of the most recently selected item in the menu.
<code>MIconMenu.getValue()</code>	Returns the label of the most recently selected item in the menu.
<code>MIconMenu.isDisabled()</code>	Returns <code>true</code> if the item indicated in the index is disabled. Returns <code>false</code> if the item indicated by the index is enabled.
<code>MIconMenu.removeAll()</code>	Removes all items from the menu.
<code>MIconMenu.removeIcon()</code>	Removes the icon from the icon button.

Method	Description
<code>MIconMenu.removeItemAt()</code>	Removes the item from the menu indicated by the index.
<code>MIconMenu.replaceAllItems()</code>	Deletes and replaces all the items in the <code>DataProvider</code> instance.
<code>MIconMenu.replaceItemAt()</code>	Overwrites an item at the specified index with a new item object.
<code>MIconMenu.setCheckedIndices()</code>	Turns on the check marks for items indicated in the <code>itemArray</code> parameter.
<code>MIconMenu.setDataProvider()</code>	Sets the data provider for the menu to the one indicated in <code>dataProvider</code> . The data provider should be an array of labels or a <code>DataProvider</code> object.
<code>MIconMenu.setEnabled()</code>	Enables or disables the <code>IconMenu</code> component.
<code>MIconMenu.setEnabledIndices()</code>	Enables or disables individual items in the menu.
<code>MIconMenu.setIcon()</code>	Sets the icon of the <code>IconButton</code> component to the symbol in the library indicated by <code>symbolName</code> .
<code>MIconMenu.setLabel()</code>	Sets the label of the <code>IconButton</code> component.
<code>MIconMenu.setLabelPlacement()</code>	Sets the placement of the label in the <code>IconButton</code> component relative to the icon. The values can be <code>left</code> , <code>right</code> , <code>top</code> , and <code>button</code> .
<code>MIconMenu.setMenuWidth()</code>	Sets the width of the menu.
<code>MIconMenu.setPopUpLocation()</code>	Sets the location of the pop-up menu to that of the movie clip instance.
<code>MIconMenu.setRowCount()</code>	Sets the number of rows visible in the menu. The index parameter indicates the number of visible rows.
<code>MIconMenu.setSize()</code>	Sets the width and height of the <code>IconButton</code> component.
<code>MIconMenu.showCheckmarks()</code>	Shows or hides check marks that can appear to the left of items in the menu.

MIconMenu.addItem()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIcon.addItem(label [,data])
```

Parameters

label A text string for the menu item.

data The optional value to associate with the menu item.

Returns

Nothing.

Description

Method; adds a new item with the specified label and optional data to the end of the menu and updates the menu. The data can be any Flash object, string, Boolean value, integer, object, or movie clip.

For best performance and load-time results, do not add more than 400 items in a single frame.

Example

The following example adds the item `Kenny` with an associated value of `Keen` to the end of the menu in the menu `teacherList`:

```
teacherList.addItem("Kenny", Keen);
```

MenuItem.addItemAt()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myMenuItem.addItemAt(index, label[, data])
```

Parameters

index An integer specifying the position at which to insert the item.

label A text string for the menu item.

data The optional value to associate with the list item.

Returns

Nothing.

Description

Method; adds a new item with the specified label and optional associated data to the menu at the specified index position. The *data* parameter can be any Flash object, string, Boolean value, integer, object, or movie clip. As you add each item, the list is updated and the scroll bar is resized.

The `MenuItem` component uses a zero-based index, where the item at index 0 appears at the top of the list.

For best performance and load-time results, do not add more than 400 items in a single frame. This guideline applies whether you are adding the items to a single menu or to several menus.

Example

The following example adds the item `Justin` with the associated value `Ace` as the fifth item in the list of the menu `Favorites`:

```
Favorites.addItemAt(4, "Justin", Ace);
```

MIconMenu.clearChecked()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.clearChecked()
```

Parameters

None.

Returns

Nothing.

Description

Method; turns off check marks for any selected items in the menu.

Example

The following example turns off check marks:

```
myIconMenu.clearChecked();
```

See also

[MIconMenu.setCheckedIndices\(\)](#)

MIconMenu.clearDisabled()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.clearDisabled()
```

Parameters

None.

Returns

Nothing.

Description

Method; reenables all items in the menu that were previously disabled using [MIconMenu.setEnabledIndices\(\)](#) on page 266.

Example

The following code reenables disabled menu items:

```
myIconMenu.clearDisabled();
```

See also

[MIconMenu.setEnabledIndices\(\)](#)

MIconMenu.getCheckedIndices()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.getCheckedIndices()
```

Parameters

None.

Returns

Nothing.

Description

Method; returns an array of indexes for items in the menu that have check marks next to them.

Example

The following code traces the indexes of items in the menu with check marks:

```
var checked = myIconMenu.getCheckedIndices();  
trace(checked);
```

See also

[MIconMenu.setCheckedIndices\(\)](#)

MenuItem.getCheckedItems()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myMenuItem.getCheckedItems()
```

Parameters

Nothing.

Returns

An array of objects with label and data properties for menu items that have check marks next to them.

Description

Method; returns an array of objects with label and data properties for menu items that have check marks next to them.

Example

The following code traces the menu items with check marks:

```
var checked = myMenuItem.getCheckedItems();  
trace(checked);
```

MenuItem.getCheckmarks()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myMenuItem.getCheckmarks()
```

Parameters

Nothing.

Returns

A Boolean value: if `true`, check marks are turned on for the menu; if `false`, check marks are turned off.

Description

Method; returns a Boolean value. If `true`, check marks are turned on for the menu; if `false`, check marks are turned off.

Example

The following code traces the return value for `getCheckmarks`:

```
trace(myIconMenu.getCheckmarks());
```

MIconMenu.getDisabledIndices()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.getDisabledIndices()
```

Parameters

None.

Returns

An array of indexes corresponding to items in the menu that were disabled using the [MIconMenu.setEnabledIndices\(\)](#) method.

Description

Method; returns an array of indexes for items disabled using [MIconMenu.setEnabledIndices\(\)](#) [on page 266](#).

Example

The following code traces the array of indexes for disabled items in the icon menu instance:

```
var disabledInd = myIconMenu.getDisabledIndices();  
trace(disabledInd);
```

MIconMenu.setEnabled()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.setEnabled()
```

Parameters

None.

Returns

A Boolean value: indicates whether the icon menu is enabled (`true`) or disabled (`false`).

Description

Method; indicates whether the icon menu is enabled or disabled.

Example

The following example traces the value indicating whether the icon menu is enabled:

```
trace(myIconMenu.getEnabled());
```

MIconMenu.getIcon()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.getIcon()
```

Parameters

None.

Returns

A movie clip instance.

Description

Method; retrieves the name of the symbol in the library (the movie clip instance) that serves as the icon for the button that toggles the pop-up menu.

Example

The following example traces the name of the icon button's movie clip instance:

```
trace(myIconMenu.getIcon());
```

MIconMenu.getItemAt()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.getItemAt(index)
```

Parameters

index The index of the desired item.

Returns

An object containing label and data properties for the item in the menu indicated by *index*.

Description

Method; retrieves an item in the menu, specified by its index.

Example

The following example gets the third item in the icon menu:

```
myIconMenu.getItemAt(2);
```

MIconMenu.getItemID()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.getItemID(index)
```

Parameters

index The index of the item whose ID is to be retrieved.

Returns

A string.

Description

Method; returns the value of the `item._ID_` property found at *index*. The returned value is a string datatype but can be evaluated to a number.

Example

The following example gets the ID of the third item in the list:

```
getItemID(2);
```

MIconMenu.getLabel()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.getLabel()
```

Parameters

None.

Returns

A string.

Description

Method; retrieves the label of the button used to toggle the pop-up menu, if a label exists. The label appears on the button.

Example

The following code traces the text label of the button:

```
var label = myIconMenu.getLabel();  
trace(label);
```

MIconMenu.getLabelPlacement()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.getLabelPlacement()
```

Parameters

None.

Returns

The position of the label for the button

Description

Method; retrieves the position of the label of the button. Possible values are `left`, `right`, `top`, and `bottom`.

Example

The following example traces the label position of the button:

```
var labelPosition = myIconMenu.getLabelPlacement();
trace(labelPosition);
```

MIconMenu.getLength()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.getLength()
```

Parameters

Nothing.

Returns

The number of items in the icon menu.

Description

Method; returns the number of items in the icon menu.

Example

The following example traces the number of items in the icon menu:

```
trace(myIconMenu.getLength());
```

MIconMenu.getRowCount()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.getRowCount()
```

Parameters

None.

Returns

The number of rows visible in the icon menu.

Description

Method; retrieves the number of rows visible in the icon menu.

Example

The following example traces the number of rows in the icon menu:

```
var rows = myIconMenu.getRowCount();
trace(rows);
```

MIconMenu.getSelectedIndex()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.getSelectedIndex()
```

Parameters

None.

Returns

The index of the menu item that the user selected most recently.

Description

Method; retrieves the index of the menu item that the user selected most recently.

Example

The following example traces the index of the most recently selected menu item:

```
var lastSelected = myIconMenu.getSelectedIndex();
trace(lastSelected);
```

MIconMenu.getSelectedItem()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.getSelectedItem()
```

Parameters

None.

Returns

An object with label and data properties of the most recently selected menu item.

Description

Method; returns an object with label and data properties of the menu item most recently selected by the user.

Example

The following example traces the most recently selected item:

```
var lastSelected = myIconMenu.getSelectedItem();  
trace(lastSelected);
```

MIconMenu.getValue()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.getValue()
```

Parameters

None.

Returns

The label of the most recently selected menu item.

Description

Method; returns the label of the most recently selected menu item.

Example

The following example gets the label of the most recently selected menu item:

```
var recentSelection = myIconMenu.getValue();  
trace(recentSelection);
```

MIconMenu.isDisabled()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.isDisabled(index)
```

Parameters

index The index of the item whose status is being checked.

Returns

A Boolean value; if `true`, the item indicated is disabled; if `false`, it is enabled.

Description

Method; returns `true` if the item indicated in *index* is disabled; `false` if the item is enabled.

Example

The following example checks whether the second item in the icon menu is disabled:

```
trace(myIconMenu.isDisabled(1));
```

MIconMenu.removeAll()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.removeAll();
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all items from the menu.

Example

The following example removes all items from the icon menu:

```
myIconMenu.removeAll();
```

MIconMenu.removeIcon()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.removeIcon();
```

Parameters

None.

Returns

Nothing.

Description

Method; removes the icon from the button.

Example

The following code removes the icon from the button:

```
myIconMenu.removeIcon();
```

MIconMenu.removeItemAt()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.removeItemAt(index)
```

Parameters

index The index number for the item that you want to remove from the menu.

Returns

Nothing.

Description

Method; removes the specified item from the menu.

Example

The following code removes the fifth item from the icon menu:

```
myIconMenu.removeItemAt(4);
```

MIconMenu.replaceAllItems()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.replaceAllItems(items)
```

Parameters

items An array of objects (ordered) or an object of objects (unordered).

Returns

Nothing.

Description

Method; deletes all items in the list and replaces them with the items specified in the *items* parameter.

Example

The following code replaces all items in the list with a new array of objects:

```
var podMenu = new Array();
podMenu[0] = "Jody";
podMenu[1] = "Mary";
podMenu[2] = "Marcelle";
podMenu[3] = "Dale";
podMenu[4] = "Stephanie";
podMenu[5] = "Barbara";
myIconMenu.replaceAllItems(podMenu);
```

MIconMenu.replaceItemAt()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.replaceItemAt(index, item)
```

Parameters

index The index of the item to be replaced.

item An object.

Returns

Nothing.

Description

Method; overwrites an item at the specified index with a new item object. If *item* has an `_ID_` property, that property is overwritten with a new one.

Example

The following code replaces the fourth item in the list with the new item:

```
replaceItemAt(3, {category: "circus" , name: "lion"});
```

MIconMenu.setCheckedIndices()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.setCheckedIndices(itemArray)
```

Parameters

itemArray An array of indexes for the menu items that appear with check marks to the left.

Returns

Nothing.

Description

Method; turns on the check marks for items indicated in *itemArray*.

Example

The following code turns on check marks for the second, third, and fourth items in the menu (in a zero-based index):

```
myIconMenu.setCheckedIndices([1,2,3]);
```

MIconMenu.setDataProvider()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.setDataProvider(dataprovider)
```

Parameters

dataprovider An array of labels or a DataProvider object.

Returns

Nothing.

Description

Method; sets the data provider for the menu to the one indicated in *dataprovider*.

If *dataProvider* is an instance of the DataProvider class, it must implement the methods of the DataProvider class.

The `DataProvider` class included with the Macromedia Central SDK contains new methods. For more information about the `DataProvider` class and its methods, see “`Central.DataProviderClass` object” in *Developing Central Applications*.

The following examples show how you can use an `Array` object or a `DataProvider` object as a data provider.

Example 1

The following code specifies the `Array` object `podMenu` as the data provider for `iconMenu1` and creates the array `podMenu` to display the labels of the items listed in `iconMenu1`:

```
iconMenu1.setDataProvider(podMenu);
podMenu = new Array();
podMenu[0] = "Jody";
podMenu[1] = "Mary";
podMenu[2] = "Marcelle";
podMenu[3] = "Dale";
podMenu[4] = "Stephanie";
podMenu[5] = "Barbara";
```

Example 2

The following code creates the array `itemList1`, which specifies both the label and the data for list items. This `Array` object could be used as an alternative data provider for `iconMenu1`.

```
itemList1 = new Array();
for (i=0; i<10; i++) {

    // create a real item
    var myItem = new Object();
    myItem.label = "Item" + i;
    myItem.data = 75;

    // put it in the array
    itemList1[i] = myItem;
}
```

Example 3

The following code specifies `prefsList`, an instance of the `DataProvider` class, as the data provider for `iconMenu1`:

```
iconMenu1.setDataProvider(prefsList);
```

The following code creates a new instance of the `DataProvider` class and then adds the item labels using the `DataProvider` `addItem` method:

```
prefsList = new mx.central.data.DataProviderClass();
prefsList.addItem("Names");
prefsList.addItem("Addresses");
prefsList.addItem("Email Addresses");
prefsList.addItem("Phone Numbers");
```

Note: The `addItem` method is one method of the `DataProvider` class. Programmers interested in using the `DataProvider` class should refer to the `DataProvider` documentation in *Developing Central Applications* before using the methods.

MIconMenu.setEnabled()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.setEnabled(state)
```

Parameters

state A Boolean value; enables (*true*) or disables (*false*) the component.

Returns

Nothing.

Description

Method; sets the state of the icon menu. If *true*, the icon menu is enabled; if *false*, the icon menu is disabled.

Example

The following code disables the icon menu, so that it appears dimmed and is unavailable to the user:

```
myIconMenu.setEnabled(false);
```

MIconMenu.setEnabledIndices()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.setEnabledIndices(itemArray, state)
```

Parameters

itemArray An array of indexes for the enabled or disabled items.

state A Boolean value; enables the items in *itemArray* (*true*) or disables the items (*false*).

Returns

Nothing.

Description

Method; enables or disables individual items in the menu. The *itemArray* parameter is an array of indexes for items in the list to be enabled or disabled. When *state* is `true`, the items indicated in *itemArray* are enabled; when *state* is `false`, the items are disabled.

Example

The following code disables the first and second items in the icon menu:

```
myIconMenu.setEnabledIndices([0,1],false);
```

MIconMenu.setIcon()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.setIcon(symbol)
```

Parameters

symbol A string; the name of the icon symbol as specified in the library.

Returns

Nothing.

Description

Method; sets the icon of the icon menu component to the symbol specified in the library.

Example

The following code sets the icon to `triangle.jpg`:

```
myIconMenu.setIcon("triangle.jpg");
```

MIconMenu.setLabel()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.setLabel(label)
```

Parameters

label A string; the label that appears on the icon menu button.

Returns

Nothing.

Description

Method; sets the label that appears on the icon menu button.

Example

The following code sets the label to Options:

```
myIconMenu.setLabel("Options");
```

MIconMenu.setLabelPlacement()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.setLabelPlacement(position)
```

Parameters

position A string; the position of the label, relative to the icon. Possible values are left, right, top, and bottom.

Returns

Nothing.

Description

Method; sets the position of the label in the IconButton component, relative to the icon. Possible values are left, right, top, and bottom.

Example

The following code sets the position of the label to the left of the button:

```
myIconMenu.setLabelPlacement("left");
```

MIconMenu.setMenuWidth()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.setMenuWidth(width)
```

Parameters

width An integer to set the width of the menu, in pixels.

Returns

Nothing.

Description

Method; sets the width of the menu. The length is determined by the number of items.

Example

The following code sets the width of the menu to 50 pixels:

```
myIconMenu.setMenuWidth(50);
```

MIconMenu.setPopUpLocation()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.setPopUpLocation(movieclip)
```

Parameters

movieclip A movie clip instance.

Returns

Nothing.

Description

Method; sets the location of the pop-up menu to that of the movie clip instance.

MIconMenu.setRowCount()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.setRowCount(index)
```

Parameters

index The number of visible rows in the menu.

Returns

Nothing.

Description

Method; sets the number of rows visible in the menu. The *index* parameter indicates the number of visible rows.

Example

The following code example sets the number of visible rows to five, in a zero-based index:

```
myIconMenu.setRowCount(4);
```

MIconMenu.setSize()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.setSize(width, height)
```

Parameters

width The width of the button, in pixels.

height The height of the button, in pixels.

Returns

Nothing.

Description

Method; sets the width and height of the button, in pixels.

Example

The following code sets the width to 20 pixels and the height to 15 pixels:

```
myIconMenu.setSize(20,15);
```

MIconMenu.showCheckmarks()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myIconMenu.showCheckmarks(state)
```

Parameters

state A Boolean value; shows the check marks (*true*) or hides them (*false*).

Returns

Nothing.

Description

Method; sets whether check marks are visible in the menu. When *state* is *true*, check marks appear to the left of items that the user selected in the menu. When *state* is *false*, no check marks appear.

Example

The following example hides check marks:

```
myIconMenu.showCheckmarks(false);
```

Label component

A label component is a single line of text. You can specify that a label be formatted with HTML. 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. The `autoSize` parameter is not supported in live preview.

When you add the Label component to an application, you can use the Accessibility panel to make it accessible to screen readers. First, you must add the following line of code to enable accessibility:

```
mx.accessibility.LabelAccImpl.enableAccessibility();
```

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

Using 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. If you're building an application using components based on version 2 of the Macromedia Component Architecture, 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. See ["Using styles with the Label component" on page 273](#).

Label parameters

You can set the following authoring parameters for each Label component instance in the Property inspector or in the Component inspector:

text indicates the text of the label; the default value is `Label`.

html indicates whether the label is formatted with HTML (`true`) or not (`false`). If this parameter is set to `true`, a label cannot be formatted with styles. The default value is `false`.

autoSize indicates how the label sizes and aligns to fit the text. The default value is `none`. The parameter can have any of the following four values:

- `none` specifies that the label doesn't resize or align to fit the text.
- `left` specifies that the right and bottom sides of the label resize to fit the text. The left and top sides don't resize.
- `center` specifies that the bottom side of the label resizes to fit the text. The horizontal center of the label stays anchored at its original horizontal center position.
- `right` specifies that the left and bottom sides of the label resize to fit the text. The top and right side don't resize.

Note: The Label component's `autoSize` property is different from the built-in ActionScript TextField object's `autoSize` property.

You can write ActionScript to set additional options for Label instances using its methods, properties, and events. For more information, see [“Label class” on page 273](#).

Creating 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 is beside a combo box with dates in a shopping cart application.

To create an application with the Label component:

1. Drag a Label component from the Components panel to the Stage.
2. In the Component inspector, enter **Expiration Date** for the label parameter.

Customizing the Label component

You can transform a Label component horizontally and vertically while authoring and at runtime. 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 does resize. For more information, see [“Label parameters” on page 272](#). At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)) or `Label.autoSize`.

Using 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. For example, you can't set the `color` style to "blue" for one word in a label and to "red" for the second word in the same label.

If the name of a style property ends in "Color", it is a color style property and behaves differently than noncolor style properties. For more information about styles, see "Using styles to customize component color and text" in Flash Help.

A Label component supports the following styles:

Style	Theme	Description
<code>color</code>	Both	The text color. The default value is <code>0x0B333C</code> for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is <code>0x848384</code> (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is <code>10</code> .
<code>fontStyle</code>	Both	The font style: either <code>"normal"</code> or <code>"italic"</code> . The default value is <code>"normal"</code> .
<code>fontWeight</code>	Both	The font weight: either <code>"none"</code> or <code>"bold"</code> . The default value is <code>"none"</code> . All components can also accept the value <code>"normal"</code> in place of <code>"none"</code> during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return <code>"none"</code> .
<code>textAlign</code>	Both	The text alignment: either <code>"left"</code> , <code>"right"</code> , or <code>"center"</code> . The default value is <code>"left"</code> .
<code>textDecoration</code>	Both	The text decoration: either <code>"none"</code> or <code>"underline"</code> . The default value is <code>"none"</code> .

Using skins with the Label component

The Label component does not have any visual elements to skin.

Label class

Inheritance `MovieClip` > [UIObject class](#) > `Label`

ActionScript Class Name `mx.controls.Label`

The properties of the Label class allow you at runtime to specify text for the label, indicate whether the text can be formatted with HTML, and indicate whether the label auto-sizes to fit the text.

Setting a property of the Label class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.Label.version);
```

Note: The code `trace(myLabelInstance.version);` returns `undefined`.

Method summary for the Label class

There are no methods exclusive to the Label class.

Methods inherited from the UIObject class

The following table lists the methods the Label class inherits from the UIObject class. When calling these methods from the Label object, use the form `labelInstance.methodName`.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it will be redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Property summary for the Label class

The following table lists properties of the Label class.

Property	Description
<code>Label.autoSize</code>	A string that indicates how a label sizes and aligns to fit the value of its <code>text</code> property. There are four possible values: "none", "left", "center", and "right". The default value is "none".
<code>Label.html</code>	A Boolean value that indicates whether a label can be formatted with HTML (<code>true</code>) or not (<code>false</code>).
<code>Label.text</code>	The text on the label.

Properties inherited from the UIObject class

The following table lists the properties the Label class inherits from the UIObject class. When accessing these properties, use the form `labelInstance.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Event summary for the Label class

There are no events exclusive to the Label class.

Events inherited from the UIObject class

The following table lists the events the Label class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Label.autoSize

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

labelInstance.autoSize

Description

Property; a string that indicates how a label sizes and aligns to fit the value of its `text` property. There are four possible values: "none", "left", "center", and "right". The default value is "none".

- `none` The label doesn't resize or align to fit the text.
- `left` The right and bottom sides of the label resize to fit the text. The left and top sides don't resize.
- `center` The bottom side of the label resizes to fit the text. The horizontal center of the label stays anchored at its original horizontal center position.
- `right` The left and bottom sides of the label resize to fit the text. The top and right sides don't resize.

Note: The Label component's `autoSize` property is different from the built-in ActionScript TextField object's `autoSize` property.

Label.html

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

labelInstance.html

Description

Property; a Boolean value that indicates whether the label can be formatted with HTML (`true`) or not (`false`). The default value is `false`. Label components with the `html` property set to `true` cannot be formatted with styles.

To retrieve plain text from HTML-formatted text, set the `HTML` property to `false` and then access the `text` property. This will remove the HTML formatting, so you may want to copy the label text to an offscreen Label or TextArea component before you retrieve the plain text.

Example

The following example sets the `html` property to `true` so the label can be formatted with HTML. The `text` property is then set to a string that includes HTML formatting.

```
lbl.html = true;
lbl.text = "The <b>Royal</b> Nonesuch";
```

The word “Royal” displays in bold.

Label.text

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

labelInstance.text

Description

Property; the text of a label. The default value is "Label".

Example

The following code sets the `text` property of the Label instance `labelControl` and sends the value to the Output panel:

```
labelControl.text = "The Royal Nonesuch";
trace(labelControl.text);
```

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.

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 <code>Key.getAscii()</code> 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.

Key	Description
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: 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 will show items 0-9, 9-18, 18-27, and so on, with one item overlapping per page.

For more information about controlling focus, see “Creating custom focus navigation” in Flash Help or [“FocusManager class” on page 231](#).

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 use the Accessibility panel to make it accessible to screen readers. First, you must add the following line of code to enable accessibility:

```
mx.accessibility.ListAccImpl.enableAccessibility();
```

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

Using 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, and the user scrolls through a list and selects one 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.

Understanding the design of the List component

When you design an application with the List component, or any component that extends the List class, it is helpful to understand how the list was designed. The following are some fundamental assumptions and requirements that Macromedia used when developing the List class:

- Keep it small, fast, and simple.
Don't make something more complicated than absolutely necessary. This was the prime design directive. Most of the requirements listed below are based on this directive.
- Lists have uniform row heights.
Every row must be the same height; the height can be set during authoring or at runtime.
- Lists must scale to thousands of records.

- Lists don't measure text.

This restriction has the most potential ramifications. Because a list must scale to thousands of records, any one of which could contain an unusually long string, it shouldn't grow to fit the largest string of text within it, or add a horizontal scroll bar in "auto" mode. Also, measuring thousands of strings would be too intensive. The compromise is the `maxHPosition` property, which, when `vScrollPolicy` is set to "on", gives the list extra buffer space for scrolling.

If you know you're likely to deal with long strings, turn `hScrollPolicy` to "on", and add a 200-pixel `maxHPosition` value to your List or Tree component. A user is more or less guaranteed to be able to scroll to see everything. The DataGrid component, however, does support "auto" as an `hScrollPolicy` value, because it measures columns (which are the same width per item), not text.

The fact that lists don't measure text also explains why lists have uniform row heights. Sizing individual rows to fit text would require intensive measuring. For example, if you wanted to accurately show the scroll bars on a list with nonuniform row height, you'd need to premeasure every row.

- Lists perform worse as a function of their visible rows.

Although lists can display 5000 records, they can't render 5000 records at once. The more visible rows (specified by the `rowCount` property) you have on the Stage, the more work the list must do to render. Limiting the number of visible rows, if at all possible, is the best solution.

- Lists aren't tables.

For example, DataGrid components, which extend the List class, are intended to provide an interface for many records. They're not designed to display complete information; they're designed to display enough information so that users can drill down to see more. The message view in Microsoft Outlook is a prime example. You don't read the entire e-mail in the grid; the mail would be difficult to read and the client would perform terribly. Outlook displays enough information so that a user can drill into the post to see the details.

List parameters

You can set the following authoring parameters for each List component instance in the Property inspector or in the Component inspector:

data is an array of values that populate the data of the list. The default value is `[]` (an empty array). There is no equivalent runtime property.

labels is an array of text values that populate the label values of the list. The default value is `[]` (an empty array). There is no equivalent runtime property.

multipleSelection is a Boolean value that indicates whether you can select multiple values (`true`) or not (`false`). The default value is `false`.

rowHeight indicates the height, in pixels, of each row. The default value is 20. Setting a font does not change the height of a row.

You can write ActionScript to set additional options for List instances using its methods, properties, and events. For more information, see ["List class" on page 284](#).

Creating an application with the List component

The following procedure explains how to add a List component to an application while authoring. In this example, the list is a sample with three items.

To add a simple List component to an application:

1. Drag a List component from the Components panel to the Stage.
2. Select the list and select Modify > Transform to resize it to fit your application.
3. In the Property inspector, do the following:
 - Enter the instance name **myList**.
 - Enter **Item1**, **Item2**, and **Item3** for the labels parameter.
 - Enter **item1.html**, **item2.html**, **item3.html** for the data parameter.

4. Select Control > Test Movie to see the list with its items.

You could use the data property values in your application to open HTML files.

To populate a List instance with a data provider:

1. Drag a List component from the Components panel to the Stage.
2. Select the list and select Modify > Transform to resize it to fit your application.
3. In the Actions panel, enter the instance name **myList**.
4. Select Frame 1 of the Timeline and, in the Actions panel, enter the following:

```
myList.dataProvider = myDP;
```

If you have defined a data provider named `myDP`, the list will fill with data. (For more information about data providers, see [List.dataProvider](#).)

5. Select Control > Test Movie to see the list with its items.

Customizing the List component

You can transform a List component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `List.setSize()` method (see [UIObject.setSize\(\)](#)).

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 position themselves automatically. For more information about scroll bars, see “[UIScrollBar component](#)” on page 557.

Using styles with the List component

You can set style properties to change the appearance of a List component.

A List component uses the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>alternatingRowColors</code>	Both	Specifies colors for rows in an alternating pattern. The value can be an array of two or more colors, for example, <code>0xFF00FF</code> , <code>0xCC6699</code> , and <code>0x996699</code> . Unlike single-value color styles, <code>alternatingRowColors</code> does not accept color names; the values must be numeric color codes. By default, this style is not set and <code>backgroundColor</code> is used in its place for all rows.
<code>backgroundColor</code>	Both	The background color of the list. The default color is white and is defined on the class style declaration. This style is ignored if <code>alternatingRowColors</code> is specified.
<code>backgroundDisabledColor</code>	Both	The background color when the component's <code>enabled</code> property is set to <code>false</code> . The default value is <code>0xDDDDDD</code> (medium gray).
<i>border styles</i>	Both	The List component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See "RectBorder class" in Flash Help. The default border style is <code>inset</code> .
<code>color</code>	Both	The text color.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is <code>0x848384</code> (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either <code>"normal"</code> or <code>"italic"</code> . The default value is <code>"normal"</code> .
<code>fontWeight</code>	Both	The font weight: either <code>"none"</code> or <code>"bold"</code> . The default value is <code>"none"</code> . All components can also accept the value <code>"normal"</code> in place of <code>"none"</code> during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return <code>"none"</code> .
<code>textAlign</code>	Both	The text alignment: either <code>"left"</code> , <code>"right"</code> , or <code>"center"</code> . The default value is <code>"left"</code> .

Style	Theme	Description
<code>textDecoration</code>	Both	The text decoration: either "none" or "underline". The default value is "none".
<code>textIndent</code>	Both	A number indicating the text indent. The default value is 0.
<code>defaultIcon</code>	Both	The name of the default icon to display on each row. The default value is <code>undefined</code> , which means no icon is displayed.
<code>repeatDelay</code>	Both	The number of milliseconds of delay between when a user first presses a button in the scrollbar and when the action begins to repeat. The default value is 500, half a second.
<code>repeatInterval</code>	Both	The number of milliseconds between automatic clicks when a user holds the mouse button down on a button in the scrollbar. The default value is 35.
<code>rolloverColor</code>	Both	The background color of a rolled-over row. The default value is <code>OxE3FFD6</code> (bright green) with the Halo theme and <code>OxAAAAAA</code> (light gray) with the Sample theme. When <code>themeColor</code> is changed through a <code>setStyle()</code> call, the framework sets <code>rolloverColor</code> to a value related to the <code>themeColor</code> chosen.
<code>selectionColor</code>	Both	The background color of a selected row. The default value is a <code>OxCDFFC1</code> (light green) with the Halo theme and <code>OxEEEEEE</code> (very light gray) with the Sample theme. When <code>themeColor</code> is changed through a <code>setStyle()</code> call, the framework sets <code>selectionColor</code> to a value related to the <code>themeColor</code> chosen.
<code>selectionDuration</code>	Both	The length of the transition from a normal to selected state or back from selected to normal, in milliseconds. The default value is 200.
<code>selectionDisabledColor</code>	Both	The background color of a selected row. The default value is a <code>OxDDDDDD</code> (medium gray). Because the default value for this property is the same as the default for <code>backgroundDisabledColor</code> , the selection is not visible when the component is disabled unless one of these style properties is changed.
<code>selectionEasing</code>	Both	A reference to the easing equation used to control the transition between selection states. This applies only for the transition from a normal to a selected state. The default equation uses a sine in/out formula. For more information, see "Customizing component animations" in Flash Help.
<code>textRolloverColor</code>	Both	The color of text when the mouse pointer rolls over it. The default value is <code>Ox2B333C</code> (dark gray). This style is important when you set <code>rolloverColor</code> , because the two settings must complement each other so that text is easily viewable during a rollover.

Style	Theme	Description
<code>textSelectedColor</code>	Both	The color of text in the selected row. The default value is <code>0x005F33</code> (dark gray). This style is important when you set <code>selectionColor</code> , because the two settings must complement each other so that text is easily viewable while selected.
<code>useRollOver</code>	Both	Determines whether rolling over a row activates highlighting. The default value is <code>true</code> .

Setting styles for all List components in a document

The `List` class inherits from the `ScrollSelectList` class. The default class-level style properties are defined on the `ScrollSelectList` class, which the `Menu` component and all `List`-based components extend. You can set new default style values on this class directly, and the new settings will be reflected in all affected components.

```
_global.styles.ScrollSelectList.setStyle("backgroundColor", 0xFF00AA);
```

To set a style property on the `List` and `List`-based components only, you can create a new `CSSStyleDeclaration` instance and store it in `_global.styles.List`.

```
import mx.styles.CSSStyleDeclaration;
if (_global.styles.List == undefined) {
    _global.styles.List = new CSSStyleDeclaration();
}
_global.styles.List.setStyle("backgroundColor", 0xFF00AA);
```

When creating a new class-level style declaration, you will lose all default values provided by the `ScrollSelectList` declaration. This includes `backgroundColor`, which is required for supporting mouse events. To create a class-level style declaration and preserve defaults, use a `for..in` loop to copy the old settings to the new declaration.

```
var source = _global.styles.ScrollSelectList;
var target = _global.styles.List;
for (var style in source) {
    target.setStyle(style, source.getStyle(style));
}
```

To provide styles for the `List` component but not for components that extend `List` (`DataGrid` and `Tree`), you must provide class-level style declarations for these subclasses.

```
import mx.styles.CSSStyleDeclaration;
if (_global.styles.DataGrid == undefined) {
    _global.styles.DataGrid = new CSSStyleDeclaration();
}
_global.styles.DataGrid.setStyle("backgroundColor", 0xFFFFFFFF);
if (_global.styles.Tree == undefined) {
    _global.styles.Tree = new CSSStyleDeclaration();
}
_global.styles.Tree.setStyle("backgroundColor", 0xFFFFFFFF);
```

For more information about class-level styles, see “Setting styles for a component class” in [Flash Help](#).

Using skins with the List component

The List component uses an instance of `RectBorder` for its border and scroll bars for scrolling images. For more information about skinning these visual elements, see “[RectBorder class](#)” in Flash Help and “[Using skins with the ScrollPane component](#)” on page 424.

List class

Inheritance `MovieClip` > [UIObject class](#) > [UIComponent class](#) > `View` > `ScrollView` > `ScrollSelectList` > `List`

ActionScript Class Name `mx.controls.List`

The List component is composed of three parts: items, rows, and a data provider.

An *item* is an ActionScript object used for storing the units of information in the list. A list can be thought of as an array; each indexed space of the array is an item. 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 *row* is a component that is used to display an item. Rows are either supplied by default by the list (the `SelectableRow` class is used), or you can supply them, usually as a subclass of the `SelectableRow` class. The `SelectableRow` class implements the `CellRenderer` API, which is the set of properties and methods that allow the list to manipulate each row and send data and state information (for example, `size`, `selected`, and so on) to the row for display.

A data provider is a data model of the list of items in a list. Any array in the same frame as a list is automatically given methods that let you manipulate data and broadcast changes to multiple views. You can build an `Array` instance or get one from a server and use it as a data model for multiple lists, combo boxes, data grids, and so on. The List component has methods that proxy to its data provider (for example, `addItem()` and `removeItem()`). If no external data provider is provided to the list, these methods create a data provider instance automatically, which is exposed through `List.dataProvider`.

To add a List component to the tab order of an application, set its `tabIndex` property (see [UIComponent.tabIndex](#)). The List component uses the Focus Manager to override the default Flash Player focus rectangle and draw a custom focus rectangle with rounded corners. For more information, see “[Creating custom focus navigation](#)” in Flash Help.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.List.version);
```

Note: The code `trace(myListInstance.version);` returns `undefined`.

Method summary for the List class

The following table lists methods of the List class.

Method	Description
List.addItem()	Adds an item to the end of the list.
List.addItemAt()	Adds an item to the list at the specified index.
List.getItemAt()	Returns the item at the specified index.
List.removeAll()	Removes all items from the list.
List.removeItemAt()	Removes the item at the specified index.
List.replaceItemAt()	Replaces the item at the specified index with another item.
List.setPropertiesAt()	Applies the specified properties to the specified item.
List.sortItems()	Sorts the items in the list according to the specified compare function.
List.sortItemsBy()	Sorts the items in the list according to a specified property.

Methods inherited from the UIObject class

The following table lists the methods the List class inherits from the UIObject class. When calling these methods, use the form *listInstance.methodName*.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it will be redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Methods inherited from the `UIComponent` class

The following table lists the methods the `List` class inherits from the `UIComponent` class. When calling these methods, use the form `listInstance.methodName`.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the `List` class

The following table lists properties of the `List` class.

Property	Description
<code>List.cellRenderer</code>	Assigns the class or symbol to use to display each row of the list.
<code>List.dataProvider</code>	The source of the list items.
<code>List.hPosition</code>	The horizontal position of the list.
<code>List.hScrollPolicy</code>	Indicates whether the horizontal scroll bar is displayed ("on") or not ("off").
<code>List.iconField</code>	A field in each item to be used to specify icons.
<code>List.iconFunction</code>	A function that determines which icon to use.
<code>List.labelField</code>	Specifies a field of each item to be used as label text.
<code>List.labelFunction</code>	A function that determines which fields of each item to use for the label text.
<code>List.length</code>	The number of items in the list. This property is read-only.
<code>List.maxHPosition</code>	The number of pixels the list can scroll to the right, when <code>List.hScrollPolicy</code> is set to "on".
<code>List.multipleSelection</code>	Indicates whether multiple selection is allowed in the list (<code>true</code>) or not (<code>false</code>).
<code>List.rowCount</code>	The number of rows that are at least partially visible in the list.
<code>List.rowHeight</code>	The pixel height of every row in the list.
<code>List.selectable</code>	Indicates whether the list is selectable (<code>true</code>) or not (<code>false</code>).
<code>List.selectedIndex</code>	The index of a selection in a single-selection list.
<code>List.selectedIndices</code>	An array of the selected items in a multiple-selection list.
<code>List.selectedItem</code>	The selected item in a single-selection list. This property is read-only.
<code>List.selectedItems</code>	The selected item objects in a multiple-selection list. This property is read-only.
<code>List.vPosition</code>	The topmost visible item of the list.
<code>List.vScrollPolicy</code>	Indicates whether the vertical scroll bar is displayed ("on"), not displayed ("off"), or displayed when needed ("auto").

Properties inherited from the UIObject class

The following table lists the properties the List class inherits from the UIObject class. When accessing these properties, use the form *listInstance.propertyName*.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the UIComponent class

The following table lists the properties the List class inherits from the UIComponent class. When accessing these properties, use the form *listInstance.propertyName*.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the List class

The following table lists events that of the List class.

Event	Description
<code>List.change</code>	Broadcast whenever user interaction causes the selection to change.
<code>List.itemRollOut</code>	Broadcast when the pointer rolls over and then off of list items.
<code>List.itemRollOver</code>	Broadcast when the pointer rolls over list items.
<code>List.scroll</code>	Broadcast when a list is scrolled.

Events inherited from the UIObject class

The following table lists the events the List class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the List class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

List.addItem()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.addItem(label[, data])
```

```
listInstance.addItem(itemObject)
```

Parameters

label A string that indicates the label for the new item.

data The data for the item. This parameter is optional and can be of any data type.

itemObject An item object that usually has `label` and `data` properties.

Returns

The index at which the item was added.

Description

Method; adds a new item to the end of the list.

In the first usage example, an item object is always created with the specified `label` property, and, if specified, the `data` property.

The second usage example adds the specified item object.

Calling this method modifies the data provider of the List component. If the data provider is shared with other components, those components will update as well.

Example

Both of the following lines of code add an item to the `myList` instance. To try this code, drag a List component to the Stage and give it the instance name `myList`. Add the following code to Frame 1 in the Timeline:

```
myList.addItem("this is an Item");  
myList.addItem({label:"Gordon",age:"very old",data:123});
```

List.addItemAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.addItemAt(index, label[, data])  
listInstance.addItemAt(index, itemObject)
```

Parameters

index A number greater than or equal to 0 that indicates the position of the item.

label A string that indicates the label for the new item.

data The data for the item. This parameter is optional and can be of any data type.

itemObject An item object that usually has `label` and `data` properties.

Returns

The index at which the item was added.

Description

Method; adds a new item to the position specified by the `index` parameter.

In the first usage example, an item object is always created with the specified `label` property, and, if specified, the `data` property.

The second usage example adds the specified item object.

Calling this method modifies the data provider of the List component. If the data provider is shared with other components, those components will update as well.

Example

The following line of code adds an item to the third index position, which is the fourth item in the list:

```
myList.addItemAt(3, {label: 'Red', data: 0xFF0000});
```

List.cellRenderer

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.cellRenderer
```

Description

Property; assigns the cell renderer to use for each row of the list. This property must be a class object reference or a symbol linkage identifier. Any class used for this property must implement the CellRenderer API.

Example

The following example uses a linkage identifier to set a new cell renderer:

```
myList.cellRenderer = "ComboBoxCell";
```

List.change

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(change){  
    // your code here  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.change = function(eventObject){  
    // your code here  
}  
listInstance.addEventListener("change", listenerObject)
```

Description

Event; broadcast to all registered listeners when the selected index of the list changes as a result of user interaction.

The first usage example uses an `on()` handler and must be attached directly to a `List` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `List` instance `myBox`, sends “_level0.myBox” to the Output panel:

```
on(click){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*listInstance*) dispatches an event (in this case, `change`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see “EventDispatcher class (API)” in Flash Help.

Finally, you call the `addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

Example

The following example sends the instance name of the component that generated the `change` event to the Output panel:

```
form.change = function(eventObj){
    trace("Value changed to " + eventObj.target.value);
}
myList.addEventListener("change", form);
```

See also

[EventDispatcher.addEventListener\(\) in Flash Help](#)

List.dataProvider

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.dataProvider
```

Description

Property; the data model for items viewed in a list. The value of this property can be an array or any object that implements the DataProvider API. The default value is []. For more information, see “DataProvider API” in Flash Help.

The List component, like other data-aware components, adds methods to the Array object’s prototype so that they conform to the DataProvider API. Therefore, any array that exists at the same time as a list automatically has all the methods (addItem(), getItemAt(), and so on) it needs to be the data model for the list, and can be used to broadcast model changes to multiple components.

If the array contains objects, the `List.labelField` or `List.labelFunction` properties are accessed to determine what parts of the item to display. The default value is "label", so if a label field exists, it is chosen for display; if it doesn’t exist, a comma-separated list of all fields is displayed.

Note: If the array contains strings at each index, and not objects, the list is not able to sort the items and maintain the selection state. Any sorting will cause the selection to be lost.

Any instance that implements the DataProvider API can be a data provider for a List component. This includes Flash Remoting recordsets, Firefly data sets, and so on.

Example

This example uses an array of strings to populate the list:

```
list.dataProvider = ["Ground Shipping","2nd Day Air","Next Day Air"];
```

This example creates a data provider array and assigns it to the dataProvider property, as in the following:

```
myDP = new Array();
list.dataProvider = myDP;

for (var i=0; i<accounts.length; i++) {
    // these changes to the data provider will be broadcast to the list
    myDP.addItem({label: accounts[i].name,
                  data: accounts[i].accountID});
}
```

List.getItemAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.getItemAt(index)
```

Parameters

index A number greater than or equal to 0, and less than `List.length`. It specifies the index of the item to retrieve.

Returns

The indexed item object; undefined if the index is out of range.

Description

Method; retrieves the item at the specified index.

Example

The following code displays the label of the item at index position 4:

```
trace(myList.getItemAt(4).label);
```

List.hPosition

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.hPosition
```

Description

Property; scrolls the list horizontally to the number of pixels specified. You can't set `hPosition` unless the value of `hScrollPolicy` is "on" and the list has a `maxHPosition` that is greater than 0.

Example

The following example gets the horizontal scroll position of `myList`:

```
var scrollPos = myList.hPosition;
```

The following example sets the horizontal scroll position all the way to the left:

```
myList.hPosition = 0;
```

List.hScrollPolicy

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.hScrollPolicy
```

Description

Property; a string that determines whether the horizontal scroll bar is displayed; the value can be "on" or "off". The default value is "off". The horizontal scroll bar does not measure text; you must set a maximum horizontal scroll position (see [List.maxHPosition](#)).

Note: `List.hScrollPolicy` does not support the value "auto".

Example

The following code enables the list to scroll horizontally up to 200 pixels:

```
myList.hScrollPolicy = "on";  
myList.Box.maxHPosition = 200;
```

See also

[List.hPosition](#), [List.maxHPosition](#)

List.iconField

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.iconField
```

Description

Property; specifies the name of a field to be used as an icon identifier. If the field has a value of undefined, the default icon specified by the `defaultIcon` style is used. If the `defaultIcon` style is undefined, no icon is used.

Example

The following example sets the `iconField` property to the `icon` property of each item:

```
list.iconField = "icon"
```

See also

[List.iconFunction](#)

List.iconFunction

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

listInstance.iconFunction

Description

Property; specifies a function that determines which icon each row will use to display its item. This function receives a parameter, *item*, which is the item being rendered, and must return a string representing the icon's symbol identifier.

Example

The following example adds icons that indicate whether a file is an image or a text document. If the `data.fileExtension` field contains a value of "jpg" or "gif", the icon used will be "pictureIcon", and so on.

```
list.iconFunction = function(item){
    var type = item.data.fileExtension;
    if (type=="jpg" || type=="gif") {
        return "pictureIcon";
    } else if (type=="doc" || type=="txt") {
        return "docIcon";
    }
}
```

List.itemRollOut

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(itemRollOut){
    // your code here
}
```

Usage 2:

```
listenerObject = new Object();
listenerObject.itemRollOut = function(eventObject){
    // your code here
}
listInstance.addEventListener("itemRollOut", listenerObject)
```

Event object

In addition to the standard properties of the event object, the `itemRollOut` event has an `index` property, which specifies the number of the item that was rolled out.

Description

Event; broadcast to all registered listeners when the pointer rolls over and then off of list items.

The first usage example uses an `on()` handler and must be attached directly to a `List` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `List` instance `myList`, sends “_level0.myList” to the Output panel:

```
on(itemRollOut){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*listInstance*) dispatches an event (in this case, `itemRollOut`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “`EventDispatcher` class (API)” in Flash Help.

Example

The following example sends a message to the Output panel that indicates which item index number has been rolled over:

```
form.itemRollOut = function (eventObj) {
    trace("Item #" + eventObj.index + " has been rolled out.");
}
myList.addEventListener("itemRollOut", form);
```

See also

[List.itemRollOver](#)

List.itemRollOver

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(itemRollOver){
    // your code here
}
```

Usage 2:

```
listenerObject = new Object();
listenerObject.itemRollOver = function(eventObject){
    // your code here
}
listInstance.addEventListener("itemRollOver", listenerObject)
```

Event object

In addition to the standard properties of the event object, the `itemRollOver` event has an `index` property that specifies the number of the item that was rolled over.

Description

Event; broadcast to all registered listeners when the list items are rolled over.

The first usage example uses an `on()` handler and must be attached directly to a `List` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `List` instance `myList`, sends “_level0.myList” to the Output panel:

```
on(itemRollOver){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (`listInstance`) dispatches an event (in this case, `itemRollOver`) and the event is handled by a function, also called a *handler*, on a listener object (`listenerObject`) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (`eventObject`) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class (API)” in Flash Help.

Example

The following example sends a message to the Output panel that indicates which item index number has been rolled over:

```
form.itemRollOver = function (eventObj) {
    trace("Item #" + eventObj.index + " has been rolled over.");
}
myList.addEventListener("itemRollOver", form);
```

See also

[List.itemRollOut](#)

List.labelField

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

listInstance.labelField

Description

Property; specifies a field in each item to be used as display text. This property takes the value of the field and uses it as the label. The default value is "label".

Example

The following example sets the `labelField` property to be the "name" field of each item. "Nina" would display as the label for the item added in the second line of code:

```
list.labelField = "name";  
list.addItem({name: "Nina", age: 25});
```

See also

[List.labelFunction](#)

List.labelFunction

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

listInstance.labelFunction

Description

Property; specifies a function that determines which field (or field combination) of each item to display. This function receives one parameter, *item*, which is the item being rendered, and must return a string representing the text to display.

Example

The following example makes the label display some formatted details of the items:

```
list.labelFunction = function(item){  
    return "The price of product " + item.productID + ", " + item.productName +  
        " is $"  
    + item.price;  
}
```

See also

[List.labelField](#)

List.length**Availability**

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.length
```

Description

Property (read-only); the number of items in the list.

Example

The following example places the value of `length` in a variable:

```
var len = myList.length;
```

List.maxHPosition**Availability**

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.maxHPosition
```

Description

Property; specifies the number of pixels the list can scroll when [List.hScrollPolicy](#) is set to "on". The list doesn't precisely measure the width of text that it contains. You must set `maxHPosition` to indicate the amount of scrolling that the list requires. The list does not scroll horizontally if this property is not set.

Example

The following example creates a list with 400 pixels of horizontal scrolling:

```
myList.hScrollPolicy = "on";  
myList.maxHPosition = 400;
```

See also

[List.hScrollPolicy](#)

List.**multipleSelection**

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.multipleSelection
```

Description

Property; indicates whether multiple selections are allowed (`true`) or only single selections are allowed (`false`). The default value is `false`.

Example

The following example tests to determine whether multiple items can be selected:

```
if (myList.multipleSelection){  
    // your code here  
}
```

The following example allows the list to take multiple selections:

```
myList.multipleSelection = true;
```

List.**removeAll()**

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.removeAll()
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all items in the list.

Calling this method modifies the data provider of the List component. If the data provider is shared with other components, those components will update as well.

Example

The following code clears the list:

```
myList.removeAll();
```

List.removeItemAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.removeItemAt(index)
```

Parameters

index A string that indicates the label for the new item. The value must be greater than 0 and less than `List.length`.

Returns

An object; the removed item (undefined if no item exists).

Description

Method; removes the item at the specified index position. The list indices after the specified index collapse by one.

Calling this method modifies the data provider of the List component. If the data provider is shared with other components, those components will update as well.

Example

The following code removes the item at index position 3:

```
myList.removeItemAt(3);
```

List.replaceItemAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.replaceItemAt(index, label[, data])  
listInstance.replaceItemAt(index, itemObject)
```

Parameters

index A number greater than 0 and less than `List.length` that indicates the position at which to insert the item (the index of the new item).

label A string that indicates the label for the new item.

data The data for the item. This parameter is optional and can be of any type.

itemObject An object to use as the item, usually containing `label` and `data` properties.

Returns

Nothing.

Description

Method; replaces the content of the item at the specified index.

Calling this method modifies the data provider of the List component. If the data provider is shared with other components, those components will update as well.

Example

The following example changes the fourth index position:

```
myList.replaceItemAt(3, "new label");
```

List.rowCount

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.rowCount
```

Description

Property; the number of rows that are at least partially visible in the list. This is useful if you've scaled a list by pixel and need to count its rows. Conversely, setting the number of rows guarantees that an exact number of rows will be displayed, without a partial row at the bottom.

The code `myList.rowCount = num` is equivalent to the code `myList.setSize(myList.width, h)` (where `h` is the height required to display `num` items).

The default value is based on the height of the list as set during authoring, or set by the `list.setSize()` method (see [UIObject.setSize\(\)](#)).

Example

The following example discovers the number of visible items in a list:

```
var rowCount = myList.rowCount;
```

The following example makes the list display four items:

```
myList.rowCount = 4;
```

This example removes a partial row at the bottom of a list, if there is one:

```
myList.rowCount = myList.rowCount;
```

This example sets a list to the smallest number of rows it can fully display:

```
myList.rowCount = 1;  
trace("myList has "+myList.rowCount+" rows");
```

List.rowHeight

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.rowHeight
```

Description

Property; the height, in pixels, of every row in the list. The font settings do not make the rows grow to fit, so setting the `rowHeight` property is the best way to make sure items are fully displayed. The default value is 20.

Example

The following example sets each row to 30 pixels:

```
myList.rowHeight = 30;
```

List.scroll

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(scroll){  
    // your code here  
}
```

Usage 2:

```
listenerObject = new Object();
listenerObject.scroll = function(eventObject){
    // your code here
}
listInstance.addEventListener("scroll", listenerObject)
```

Event object

Along with the standard event object properties, the `scroll` event has one additional property, `direction`. It is a string with two possible values, "horizontal" or "vertical". For a `ComboBox` scroll event, the value is always "vertical".

Description

Event; broadcast to all registered listeners when a list scrolls.

The first usage example uses an `on()` handler and must be attached directly to a `List` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `List` instance `myList`, sends “_level0.myList” to the Output panel:

```
on(scroll){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (`listInstance`) dispatches an event (in this case, `scroll`) and the event is handled by a function, also called a *handler*, on a listener object (`listenerObject`) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (`eventObject`) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class (API)” in Flash Help.

Example

The following example sends the instance name of the component that generated the change event to the Output panel:

```
form.scroll = function(eventObj){
    trace("list scrolled");
}
myList.addEventListener("scroll", form);
```

List.selectable

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

listInstance.selectable

Description

Property; a Boolean value that indicates whether the list is selectable (*true*) or not (*false*). The default value is *true*.

List.selectedIndex

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

listInstance.selectedIndex

Description

Property; the selected index of a single-selection list. The value is undefined if nothing is selected; the value is equal to the last item selected if there are multiple selections. If you assign a value to *selectedIndex*, any current selection is cleared and the indicated item is selected.

Using the *selectedIndex* property to change selection doesn't dispatch a change event. To dispatch the change event, use the following code:

```
myList.dispatchEvent({type:"change", target:myList});
```

Example

This example selects the item after the currently selected item. If nothing is selected, item 0 is selected.

```
var selIndex = myList.selectedIndex;  
myList.selectedIndex = (selIndex==undefined ? 0 : selIndex+1);
```

See also

[List.selectedIndexes](#), [List.selectedItem](#), [List.selectedItems](#)

List.selectedIndices

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

listInstance.selectedIndices

Description

Property; an array of indices of the selected items. Assigning this property replaces the current selection. Setting `selectedIndices` to a zero-length array (or undefined) clears the current selection. The value is undefined if nothing is selected.

The `selectedIndices` property reflects the order in which the items were selected. If you click the second item, then the third item, and then the first item, `selectedIndices` returns `[1,2,0]`.

Example

The following example retrieves the selected indices:

```
var selIndices = myList.selectedIndices;
```

The following example selects four items:

```
var myArray = new Array (1,4,5,7);  
myList.selectedIndices = myArray;
```

See also

[List.selectedIndex](#), [List.selectedItem](#), [List.selectedItems](#)

List.selectedItem

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

listInstance.selectedItem

Description

Property (read-only); an item object in a single-selection list. (In a multiple-selection list with multiple items selected, `selectedItem` returns the item that was most recently selected.) If there is no selection, the value is undefined.

Example

This example displays the selected label:

```
trace(myList.selectedItem.label);
```

See also

[List.selectedIndex](#), [List.selectedIndices](#), [List.selectedItems](#)

List.selectedItems

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.selectedItems
```

Description

Property (read-only); an array of the selected item objects. In a multiple-selection list, `selectedItems` lets you access the set of items selected as item objects.

Example

The following example retrieves an array of selected item objects:

```
var myObjArray = myList.selectedItems;
```

See also

[List.selectedIndex](#), [List.selectedItem](#), [List.selectedIndices](#)

List.setPropertiesAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.setPropertiesAt(index, styleObj)
```

Parameters

index A number greater than 0 or less than `List.length` indicating the index of the item to change.

styleObj An object that enumerates the properties and values to set.

Returns

Nothing.

Description

Method; applies the specified properties to the specified item. The supported properties are `icon` and `backgroundColor`.

Example

The following example changes the fourth item to black and gives it an icon:

```
myList.setPropertiesAt(3, {backgroundColor:0x000000, icon: "file"});
```

List.sortItems()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.sortItems(compareFunc)
```

Parameters

compareFunc A reference to a function. This function is used to compare two items to determine their sort order.

For more information, see `Array.sort()` in *Flash ActionScript Language Reference*.

Returns

The index at which the item was added.

Description

Method; sorts the items in the list by using the function specified in the *compareFunc* parameter.

Example

The following example sorts the items according to uppercase labels. Note that the `a` and `b` parameters that are passed to the function are items that have `label` and `data` properties.

```
myList.sortItems(upperCaseFunc);
function upperCaseFunc(a,b){
    return a.label.toUpperCase() > b.label.toUpperCase();
}
```

See also

[List.sortItemsBy\(\)](#)

List.sortItemsBy()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
myList.sortItemsBy(fieldName, optionsFlag)
```

```
myList.sortItemsBy(fieldName, order)
```

Parameters

fieldName A string that specifies the name of the field to use for sorting. This value is usually "label" or "data".

order A string that specifies whether to sort the items in ascending order ("ASC") or descending order ("DESC").

optionsFlag Lets you perform multiple sorts of different types on a single array without having to replicate the entire array or resort it repeatedly.

The following are possible values for *optionsFlag*:

- `Array.DECENDING`, which sorts from highest to lowest.
- `Array.CASEINSENSITIVE`, which sorts without regard to case.
- `Array.NUMERIC`, which sorts numerically if the two elements being compared are numbers. If they aren't numbers, use a string comparison (which can be case-insensitive if that flag is specified).
- `Array.UNIQUESORT`, which returns an error code (0) instead of a sorted array if two objects in the array are identical or have identical sort fields.
- `Array.RETURNINDEXEDARRAY`, which returns an integer index array that is the result of the sort. For example, the following array would return the second line of code and the array would remain unchanged:

```
["a", "d", "c", "b"]  
[0, 3, 2, 1]
```

You can combine these options into one value. For example, the following code combines options 3 and 1:

```
array.sort (Array.NUMERIC | Array.DECENDING)
```

Returns

Nothing.

Description

Method; sorts the items in the list in the specified order, using the specified field name. If the *fieldName* items are a combination of text strings and integers, the integer items are listed first. The *fieldName* parameter is usually "label" or "data", but you can specify any primitive data value.

This is the fastest way to sort data in a component. It also maintains the component's selection state. The `sortItemsBy()` method is fast because it doesn't run any ActionScript while sorting. The `sortItems()` method needs to run an ActionScript compare function, and is therefore slower.

Example

The following code sorts the items in the list `surnameMenu` in ascending order using the labels of the list items:

```
surnameMenu.sortItemsBy("label", "ASC");
```

See also

[List.sortItems\(\)](#)

List.vPosition

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listInstance.vPosition
```

Description

Property; sets the topmost visible item of the list. If you set this property to an index number that doesn't exist, the list scrolls to the nearest index. The default value is 0.

Example

The following example sets the position of the list to the first index item:

```
myList.vPosition = 0;
```

List.vScrollPolicy

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

`listInstance.vScrollPolicy`

Description

Property; a string that determines whether the list supports vertical scrolling. The value of this property can be "on", "off" or "auto". The value "auto" causes a scroll bar to appear when needed.

Example

The following example disables the scroll bar:

```
myList.vScrollPolicy = "off";
```

You can still create scrolling by using [List.vPosition](#).

See also

[List.vPosition](#)

Loader component

The Loader component is a container that can display a SWF or JPEG file. You can scale the contents of the loader or resize the loader itself to accommodate the size of the contents. By default, the contents are scaled to fit the loader. You can also load content at runtime and monitor loading progress.

A Loader component can't receive focus. However, content loaded into the Loader component can accept focus and have its own focus interactions. For more information about controlling focus, see "Creating custom focus navigation" in Flash Help or "[FocusManager class](#)" on [page 231](#).

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

You can use the Accessibility panel to make Loader component content accessible to screen readers.

Using the Loader component

You can use a loader whenever you need to retrieve content from a remote location and pull it into a Flash application. For example, you could use a loader to add a company logo (JPEG file) to a form. You could also use a loader to leverage Flash work that has already been completed. For example, if you had already built a Flash application and wanted to expand it, you could use the loader to pull the old application into a new application, perhaps as a section of a tab interface. In another example, you could use the loader component in an application that displays photos. Use [Loader.load\(\)](#), [Loader.percentLoaded](#), and [Loader.complete](#) to control the timing of the image loads and display progress bars to the user during loading.

If you load certain version 2 components into a SWF file or into the Loader component, the components may not work correctly. These components include the following: Alert, ComboBox, DateField, Menu, MenuBar, and Window.

Use the `_lockroot` property when calling `loadMovie()` or loading into the Loader component. If you're using the Loader component, add the following code:

```
myLoaderComponent.content._lockroot = true;
```

If you're using a movie clip with a call to `loadMovie()`, add the following code:

```
myMovieClip._lockroot = true;
```

If you don't set `_lockroot` to `true` in the loader movie clip, the loader only has access to its own library, but not the library in the loaded movie clip.

The `_lockroot` property is supported by Flash Player 7.

Loader parameters

You can set the following authoring parameters for each Loader component instance in the Property inspector or in the Component inspector:

autoload indicates whether the content should load automatically (`true`), or wait to load until the `Loader.load()` method is called (`false`). The default value is `true`.

contentPath an absolute or relative URL indicating the file to load into the loader. A relative path must be relative to the SWF file loading the content. The URL must be in the same subdomain as the URL where the Flash content currently resides. For use in Flash Player or in test-movie mode, all SWF files must be stored in the same folder, and the filenames cannot include folder or disk drive specifications. The default value is `undefined` until the load starts.

scaleContent indicates whether the content scales to fit the loader (`true`), or the loader scales to fit the content (`false`). The default value is `true`.

You can write ActionScript to set additional options for Loader instances using its methods, properties, and events. For more information, see [“Loader class” on page 313](#).

Creating an application with the Loader component

The following procedure explains how to add a Loader component to an application while authoring. In this example, the loader loads a logo JPEG from an imaginary URL.

To create an application with the Loader component:

1. Drag a Loader component from the Components panel to the Stage.
2. Select the loader on the Stage and use the Free Transform tool to size it to the dimensions of the corporate logo.
3. In the Property inspector, enter the instance name **logo**.
4. Select the loader on the Stage and in the Component inspector, and enter **`http://corp.com/websites/logo/corplgo.jpg`** for the `contentPath` parameter.

Customizing the Loader component

You can transform a Loader component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)).

The sizing behavior of the Loader 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 [UIObject.setSize\(\)](#) is called). When `scaleContent` is `false`, the size of the component is fixed to the size of the content and [UIObject.setSize\(\)](#) has no effect.

Using styles with the Loader component

The Loader component uses the following styles.

Style	Theme	Description
<i>border styles</i>	Both	The Loader component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See “RectBorder class” in Flash Help.

The default border style is "none".

Using skins with the Loader component

The Loader component uses an instance of `RectBorder` for its border (see “RectBorder class” in Flash Help).

Loader class

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > View > Loader

ActionScript Class Name mx.controls.Loader

The properties of the Loader class let you set content to load and monitor its loading progress at runtime.

Setting a property of the Loader class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.Loader.version);
```

Note: The code `trace(myLoaderInstance.version);` returns `undefined`.

Method summary for the Loader class

The following table lists the method of the Loader class.

Method	Description
<code>Loader.load()</code>	Loads the content specified by the <code>contentPath</code> property.

Methods inherited from the UIObject class

The following table lists the methods the Loader class inherits from the UIObject class. When calling these methods from the Loader object, use the form *LoaderInstance.methodName*.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it will be redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the Loader class inherits from the UIComponent class. When calling these methods from the Loader object, use the form *LoaderInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the Loader class

The following table lists properties of the Loader class.

Property	Description
<code>Loader.autoLoad</code>	A Boolean value that indicates whether the content loads automatically (<code>true</code>) or you must call <code>Loader.load()</code> (<code>false</code>).
<code>Loader.bytesLoaded</code>	A read-only property that indicates the number of bytes that have been loaded.
<code>Loader.bytesTotal</code>	A read-only property that indicates the total number of bytes in the content.
<code>Loader.content</code>	A reference to the content of the loader. This property is read-only.
<code>Loader.contentPath</code>	A string that indicates the URL of the content to be loaded.
<code>Loader.percentLoaded</code>	A number that indicates the percentage of loaded content. This property is read-only.
<code>Loader.scaleContent</code>	A Boolean value that indicates whether the content scales to fit the loader (<code>true</code>), or the loader scales to fit the content (<code>false</code>).

Properties inherited from the UIObject class

The following table lists the properties the Loader class inherits from the UIObject class. When accessing these properties from the Loader object, use the form `LoaderInstance.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `Loader` class inherits from the `UIComponent` class. When accessing these properties from the `Loader` object, use the form `LoaderInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the `Loader` class

The following table lists events of the `Loader` class.

Event	Description
<code>Loader.complete</code>	Triggered when the content finished loading.
<code>Loader.progress</code>	Triggered while content is loading.

Events inherited from the `UIObject` class

The following table lists the events the `Loader` class inherits from the `UIObject` class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the `UIComponent` class

The following table lists the events the `Loader` class inherits from the `UIComponent` class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Loader.autoLoad

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

loaderInstance.autoLoad

Description

Property; a Boolean value that indicates whether to automatically load the content (*true*), or wait until `Loader.load()` is called (*false*). The default value is *true*.

Example

The following code sets up the loader component to wait for a `Loader.load()` call:

```
loader.autoLoad = false;
```

Loader.bytesLoaded

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

loaderInstance.bytesLoaded

Description

Property (read-only); the number of bytes of content that have been loaded. The default value is 0 until content begins loading.

Example

The following code creates a progress bar and a Loader component. It then creates a listener object with a progress event handler that shows the progress of the load. The listener is registered with the loader instance.

```
createClassObject(mx.controls.ProgressBar, "pBar", 0);
createClassObject(mx.controls.Loader, "loader", 1);
loadListener = new Object();
loadListener.progress = function(eventObj){
    // eventObj.target is the component that generated the progress event,
    // that is, the loader
    pBar.setProgress(loader.bytesLoaded, loader.bytesTotal); // show progress
}
loader.addEventListener("progress", loadListener);
loader.content = "logo.swf";
```

When you create an instance with `createClassObject()`, you have to position it on the Stage with `move()` and `setSize()`. See [UIObject.move\(\)](#) and [UIObject.setSize\(\)](#).

See also

[Loader.bytesTotal](#), [UIObject.createClassObject\(\)](#)

Loader.bytesTotal

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

loaderInstance.bytesTotal

Description

Property (read-only); the size of the content, in bytes. The default value is 0 until content begins loading.

Example

The following code creates a progress bar and a Loader component. It then creates a load listener object with a progress event handler that shows the progress of the load. The listener is registered with the loader instance, as follows:

```
createClassObject(mx.controls.ProgressBar, "pBar", 0);
createClassObject(mx.controls.Loader, "loader", 1);
loadListener = new Object();
loadListener.progress = function(eventObj){
    // eventObj.target is the component that generated the progress event,
    // that is, the loader
    pBar.setProgress(loader.bytesLoaded, loader.bytesTotal); // show progress
}
loader.addEventListener("progress", loadListener);
loader.content = "logo.swf";
```

See also

[Loader.bytesLoaded](#)

Loader.complete

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(complete){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.complete = function(eventObject){  
    ...  
}  
loaderInstance.addEventListener("complete", listenerObject)
```

Description

Event; broadcast to all registered listeners when the content has finished loading.

The first usage example uses an `on()` handler and must be attached directly to a `Loader` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `Loader` instance `myLoaderComponent`, sends “`_level0.myLoaderComponent`” to the Output panel:

```
on(complete){  
    trace(this);  
}
```

The second usage example uses a dispatcher/listener event model. A component instance (`loaderInstance`) dispatches an event (in this case, `complete`) and the event is handled by a function, also called a *handler*, on a listener object (`listenerObject`) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (`eventObject`) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “`EventDispatcher` class” in Flash Help.

Example

The following example creates a `Loader` component and then defines a listener object with a `complete` event handler that sets the loader’s `visible` property to `true`:

```
createClassObject(mx.controls.Loader, "loader", 0);  
loadListener = new Object();  
loadListener.complete = function(eventObj){  
    loader.visible = true;  
}  
loader.addEventListener("complete", loadListener);  
loader.contentPath = "logo.swf";
```

Loader.content

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

loaderInstance.content

Description

Property (read-only); a reference to a movie clip instance that contains the contents of the loaded file. The value is undefined until the load begins.

See also

[Loader.contentPath](#)

Loader.contentPath

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

loaderInstance.contentPath

Description

Property; a string that indicates an absolute or relative URL of the file to load into the loader. A relative path must be relative to the SWF file that loads the content. The URL must be in the same subdomain as the loading SWF file.

If you are using Flash Player or test-movie mode in Flash, all SWF files must be stored in the same folder, and the filenames cannot include folder or disk drive information.

Example

The following example tells the loader instance to display the contents of the logo.swf file:

```
loader.contentPath = "logo.swf";
```

Loader.load()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
loaderInstance.load([path])
```

Parameters

path An optional parameter that specifies the value for the `contentPath` property before the load begins. If a value is not specified, the current value of `contentPath` is used as is.

Returns

Nothing.

Description

Method; tells the loader to begin loading its content.

Example

The following code creates a `Loader` instance and sets the `autoLoad` property to `false` so that the loader must wait for a call to `load()` to begin loading content. Next, the `contentPath` property is set, which indicates where to load content from. Then other tasks can be performed before the content is loaded with `loader.load()`.

```
createClassObject(mx.controls.Loader, "loader", 0);
loader.autoLoad = false;
loader.contentPath = "logo.swf";

// Perform other tasks here and *then* start loading the file.

loader.load();
```

Loader.percentLoaded

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
loaderInstance.percentLoaded
```

Description

Property (read-only); a number indicating what percent of the content has loaded. Typically, this property is used to present the progress to the user in an easily readable form. Use the following code to round the figure to the nearest integer:

```
Math.round(bytesLoaded/bytesTotal*100)
```

Example

The following example creates a `Loader` instance and then creates a listener object with a progress handler that traces the percent loaded and sends it to the Output panel:

```
createClassObject(Loader, "loader", 0);
loadListener = new Object();
loadListener.progress = function(eventObj){
    // eventObj.target is the component that generated the progress event,
    // that is, the loader
    trace("logo.swf is " + loader.percentLoaded + "% loaded."); // track loading
    progress
}
loader.addEventListener("complete", loadListener);
loader.content = "logo.swf";
```

Loader.progress

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(progress){
    ...
}
```

Usage 2:

```
listenerObject = new Object();
listenerObject.progress = function(eventObject){
    ...
}
loaderInstance.addEventListener("progress", listenerObject)
```

Description

Event; broadcast to all registered listeners while content is loading. This event occurs when the load is triggered by the `autoLoad` parameter or by a call to `Loader.load()`. The progress event is not always broadcast, and the `complete` event may be broadcast without any progress events being dispatched. This can happen if the loaded content is a local file.

The first usage example uses an `on()` handler and must be attached directly to a `Loader` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `Loader` instance `myLoaderComponent`, sends “`_level0.myLoaderComponent`” to the Output panel:

```
on(progress){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*loaderInstance*) dispatches an event (in this case, *progress*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following code creates a `Loader` instance and then creates a listener object with an event handler for the `progress` event that sends a message to the Output panel telling what percent of the content has loaded:

```
createClassObject(mx.controls.Loader, "loader", 0);
loadListener = new Object();
loadListener.progress = function(eventObj){
    // eventObj.target is the component that generated the progress event,
    // that is, the loader
    trace("logo.swf is " + loader.percentLoaded + "% loaded."); // track loading
    progress
}
loader.addEventListener("progress", loadListener);
loader.contentPath = "logo.swf";
```

Loader.scaleContent

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

loaderInstance.scaleContent

Description

Property; indicates whether the content scales to fit the loader (`true`), or the loader scales to fit the content (`false`). The default value is `true`.

Example

The following code tells the loader to resize itself to match the size of its content:

```
loader.scaleContent = false;
```

Menu component

The Menu component lets a user select an item from a pop-up menu, much like the File or Edit menu of most software applications.

A Menu component usually opens in an application when a user rolls over or clicks a button-like menu activator. You can also script a Menu component to open when a user presses a certain key.

Menu components are always created dynamically at runtime. You must add the component to the document from the Components panel, and delete it to add it to the library. Then, use the following code to create a menu with ActionScript:

```
var myMenu = mx.controls.Menu.createMenu(parent, menuDataProvider);
```

Use the following code to open a menu in an application:

```
myMenu.show(x, y);
```

A `menuShow` event is broadcast to all of the Menu instance's listeners immediately before the menu is rendered, so you can update the state of the menu items. Similarly, immediately after a Menu instance is hidden, a `menuHide` event is broadcast.

The items in a menu are described by XML. For more information, see [“Understanding the Menu component: view and data” on page 326](#).

You cannot make the Menu component accessible to screen readers.

Interacting with the Menu component

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

After a Menu component is opened, it remains visible until it is closed by a script or until the user clicks the mouse outside the menu or inside an enabled item.

Clicking selects a menu item, except with the following types of menu items:

Disabled items or separators Rollovers and clicks have no effect (the menu remains visible).

Anchors for a submenu Rollovers activate the submenu; clicks have no effect; rolling onto any item other than those of the submenu closes the submenu.

When an item is selected, a `Menu.change` event is sent to all of the menu's listeners, the menu is hidden, and the following actions occur, depending on item type:

check The item's `selected` attribute is toggled.

radio The item becomes the current selection of its radio group.

Moving the mouse triggers `Menu.rollOut` and `Menu.rollOver` events.

Pressing the mouse outside the menu closes the menu and triggers a `Menu.menuHide` event.

Releasing the mouse in an enabled item affects item types in the following ways:

check The item's `selected` attribute is toggled.

radio The item's `selected` attribute is set to `true`, and the previously selected item's `selected` attribute in the radio group is set to `false`. The `selection` property of the corresponding radio object is set to refer to the selected menu item.

undefined and the parent of a hierarchical menu The visibility of the hierarchical menu is toggled.

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

Key	Description
Down Arrow Up Arrow	Moves the selection down and up the rows of the menu. The selection cycles at the top or bottom row.
Right Arrow	Opens a submenu, or moves selection to the next menu in a menu bar (if a menu bar exists).
Left Arrow	Closes a submenu and returns focus to the parent menu (if a parent menu exists), or moves selection to the previous menu in a menu bar (if the menu bar exists).
Enter	Opens a submenu. If a submenu does not exist, this key has the same effect as clicking and releasing on a row.

Note: If a menu is opened, you can press the Tab key to move out of the menu. You must either make a selection or dismiss the menu by pressing Escape.

Using the Menu component

You can use the Menu component to create a menu of selectable choices; this menu is like the File or Edit menu of most software applications. You can also use the Menu component to create context-sensitive menus that appear when a user clicks a hot spot or a presses a modifier key. Use the Menu component with the MenuBar component to create a horizontal menu bar with menus that extend under each menu bar item.

Like standard desktop menus, the Menu component supports menu items whose functions fall into the following general categories:

Command activators These items trigger events; you write code to handle those events.

Submenu anchors These items are anchors that open submenus.

Radio buttons These items operate in groups; you can select only one item at a time.

Check box items These items represent a Boolean (`true` or `false`) value.

Separators These items provide a simple horizontal line that divides the items in a menu into different visual groups.

Understanding the Menu component: view and data

Conceptually, the Menu component consists of a data model and a view that displays the data. The Menu class provides the view and contains the visual configuration methods. The [MenuDataProvider class](#) adds methods to the global XML prototype object (much like the DataProvider API does to the Array object); these methods let you externally construct data providers and add them to multiple menu instances. The data provider broadcasts any changes to all of its client views. (See “[MenuDataProvider class](#)” on page 354.)

A Menu instance is a hierarchical collection of XML elements that correspond to individual menu items. The attributes define the behavior and appearance of the corresponding menu item on the screen. The collection is easily translated to and from XML, which is used to describe menus (the `menu` tag) and items (the `menuitem` tag). The built-in ActionScript XML class is the basis for the model underlying the Menu component.

A simple menu with two items can be described in XML with two menu item subelements:

```
<menu>
  <menuitem label="Up" />
  <menuitem label="Down" />
</menu>
```

Note: The tag names of the XML nodes (`menu` and `menuitem`) are not important; the attributes and their nesting relationships are used in the menu.

About hierarchical menus

To create hierarchical menus, embed XML elements within a parent XML element, as follows:

```
<menu>
  <menuitem label="MenuItem A" >
    <menuitem label="SubMenuItem 1-A" />
    <menuitem label="SubMenuItem 2-A" />
  </menuitem>
  <menuitem label="MenuItem B" >
    <menuitem label="SubMenuItem 1-B" />
    <menuitem label="SubMenuItem 2-B" />
  </menuitem>
</menu>
```

This converts the parent menu item into a pop-up menu anchor, so it does not generate events when selected.

About menu item XML attributes

The attributes of a menu item XML element determine what is displayed, how the menu item behaves, and how it is exposed to ActionScript. The following table describes the attributes of an XML menu item:

Attribute name	Type	Default	Description
label	String	undefined	The text that is displayed to represent a menu item. This attribute is required for all item types, except separator.
type	separator, check, radio, normal, or undefined	undefined	The type of menu item: separator, check box, radio button, or normal (a command or submenu activator). If this attribute does not exist, the default value is normal.
icon	String	undefined	The linkage identifier of an image asset. This attribute is not required and is not available for the check, radio, or separator type.
instanceName	String	undefined	An identifier that you can use to reference the menu item instance from the root menu instance. For example, a menu item named <i>yellow</i> can be referenced as <code>myMenu.yellow</code> . This attribute is not required.
groupName	String	undefined	An identifier that you can use to associate several radio button items in a radio group, and to expose the state of a radio group from the root menu instance. For example, a radio group named <i>colors</i> can be referenced as <code>myMenu.colors</code> . This attribute is required only for the type <code>radio</code> .
selected	A Boolean value (<code>false</code> or <code>true</code>) or string (" <code>false</code> " or " <code>true</code> ")	<code>false</code>	A Boolean or string value indicating whether a check or radio item is on (<code>true</code>) or off (<code>false</code>). This attribute is not required.
enabled	A Boolean value (<code>false</code> or <code>true</code>) or string (" <code>false</code> " or " <code>true</code> ")	<code>true</code>	A Boolean or string value indicating whether this menu item can be selected (<code>true</code>) or not (<code>false</code>). This attribute is not required.

About menu item types

There are four kinds of menu items, specified by the `type` attribute:

```
<menu>
  <menuitem label="Normal Item" />
  <menuitem type="separator" />
  <menuitem label="Checkbox Item" type="check" instanceName="check_1"/>
  <menuitem label="RadioButton Item" type="radio" groupName="radioGroup_1" /
  >
</menu>
```

Normal menu items

The `Normal Item` menu item doesn't have a `type` attribute, which means that the `type` attribute defaults to `normal`. Normal items can be command activators or submenu activators, depending on whether they have nested subitems.

Separator menu items

A menu item whose `type` attribute is set to `separator` acts as a visual divider in a menu. The following XML creates three menu items, `Top`, `Middle`, and `Bottom`, with separators between them:

```
<menu>
  <menuitem label="Top" />
  <menuitem type="separator" />
  <menuitem label="Middle" />
  <menuitem type="separator" />
  <menuitem label="Bottom" />
</menu>
```

All separator items are disabled. Clicking on or rolling over a separator has no effect.

Check box menu items

A menu item whose `type` attribute is set to `check` acts as check box item in the menu; when the `selected` attribute is set to `true`, a check mark appears beside the menu item's label. When a check box item is selected, its state automatically toggles, and a `change` event is broadcast to all listeners on the root menu. The following example defines three check box menu items:

```
<menu>
  <menuitem label="Apples" type="check" instanceName="buyApples"
  selected="true" />
  <menuitem label="Oranges" type="check" instanceName="buyOranges"
  selected="false" />
  <menuitem label="Bananas" type="check" instanceName="buyBananas"
  selected="false" />
</menu>
```

You can use the instance names in `ActionScript` to access the menu items directly from the menu itself, as in the following example:

```
myMenu.setMenuItemSelected(myMenu.buyapples, true);
myMenu.setMenuItemSelected(myMenu.buyoranges, false);
```

Note: The `selected` attribute should be modified only with the `setMenuItemSelected()` method. You can directly examine the `selected` attribute, but it returns a string value of `true` or `false`.

Radio button menu items

Menu items whose `type` attribute is set to `radio` can be grouped together so that only one of the items can be selected at a time. You create a radio group by giving the menu items the same value for their `groupName` attribute, as in the following example:

```
<menu>
  <menuItem label="Center" type="radio" groupName="alignment_group"
    instanceName="center_item"/>
  <menuItem type="separator" />
  <menuItem label="Top" type="radio" groupName="alignment_group" />
  <menuItem label="Bottom" type="radio" groupName="alignment_group" />
  <menuItem label="Right" type="radio" groupName="alignment_group" />
  <menuItem label="Left" type="radio" groupName="alignment_group" />
</menu>
```

When the user selects one of the items, the current selection automatically changes, and a change event is broadcast to all listeners on the root menu. The currently selected item in a radio group is available in ActionScript through the `selection` property, as follows:

```
var selectedItem = myMenu.alignment_group.selection;
myMenu.alignment_group = myMenu.center_item;
```

Each `groupName` value must be unique within the scope of the root menu instance.

Note: The `selected` attribute should be modified only with the `setMenuItemSelected()` method. You can directly examine the `selected` attribute, but it returns a string value of `true` or `false`.

Exposing menu items to ActionScript

You can assign each menu item a unique identifier in the `instanceName` attribute, which makes the menu item accessible directly from the root menu. For example, the following XML code provides `instanceName` attributes for each menu item:

```
<menu>
  <menuItem label="Item 1" instanceName="item_1" />
  <menuItem label="Item 2" instanceName="item_2" >
    <menuItem label="SubItem A" instanceName="sub_item_A" />
    <menuItem label="SubItem B" instanceName="sub_item_B" />
  </menuItem>
</menu>
```

You can use ActionScript to access the corresponding instances and their attributes directly from the menu component, as follows:

```
var aMenuItem = myMenu.item_1;
myMenu.setMenuItemEnabled(item_2, true);
var aLabel = myMenu.sub_item_A.label;
```

Note: Each `instanceName` attribute must be unique within the scope of the root menu component instance (including all of the submenus of root).

About initialization object properties

The *initObject* (initialization object) parameter is a fundamental concept in creating the layout for the Menu component. This parameter is an object with properties. Each property represents one of the possible the XML attributes of a menu item. (For a description of the properties allowed in the *initObject* parameter, see [“About menu item XML attributes” on page 327.](#))

The *initObject* parameter is used in the following methods:

- `Menu.addItem()`
- `Menu.addItemAt()`
- `MenuDataProvider.addItem()`
- `MenuDataProvider.addItemAt()`

The following example creates an *initObject* parameter with two properties, `label` and `instanceName`:

```
var i = myMenu.addItem({label:"myMenuItem", instanceName:"myFirstItem"});
```

Several of the properties work together to create a particular type of menu item. You assign specific properties to create certain types of menu items (normal, separator, check box, or radio button).

For example, you can initialize a normal menu item with the following *initObject* parameter:

```
myMenu.addItem({label:"myMenuItem", enabled:true, icon:"myIcon",  
instanceName:"myFirstItem"});
```

You can initialize a separator menu item with the following *initObject* parameter:

```
myMenu.addItem({type:"separator"});
```

You can initialize a check box menu item with the following *initObject* parameter:

```
myMenu.addItem({type:"check", label:"myMenuCheck", enabled:false,  
selected:true, instanceName:"myFirstCheckItem"})
```

You can initialize a radio button menu item with the following *initObject* parameter:

```
myMenu.addItem({type:"radio", label:"myMenuRadio1", enabled:true,  
selected:false, groupName:"myRadioGroup" instanceName:"myFirstRadioItem"})
```

You should treat the `instanceName`, `groupName`, and `type` attributes of a menu item as read-only.

You should set them only while creating an item (for example, in a call to `addItem()`).

Modifying these attributes after creation may produce unpredictable results.

Menu parameters

There are no authoring parameters for the Menu component.

You can write ActionScript to control the Menu component using its properties, methods, and events. For more information, see [“Menu class” on page 337.](#)

Creating an application with the Menu component

In the following example, a developer is building an application and uses the Menu component to expose some of the commands that users can issue, such as Open, Close, and Save.

To create an application with the Menu component:

1. Select File > New and create a Flash document.
2. Drag the Menu component from the Components panel to the Stage and delete it.
This adds the Menu component to the library without adding it to the application. Menus are created dynamically through ActionScript.
3. Drag a Button component from the Components panel to the Stage.
The button will be used to activate the menu.
4. In the Property inspector, give the button the instance name **commandBtn**, and change its text property to **Commands**.
5. In the Actions panel on the first frame, enter the following code to add an event listener to listen for click events on the **commandBtn** instance:

```
// Create a menu
var myMenu = mx.controls.Menu.createMenu();

// Add some menu items
myMenu.addItem("Open");
myMenu.addItem("Close");
myMenu.addItem("Save");
myMenu.addItem("Revert");

// Add a change-listener to Menu to detect which menu item is selected
var changeListener = new Object();
changeListener.change = function(event) {
    var item = event.menuItem;
    trace("Item selected: " + item.attributes.label);
}
myMenu.addEventListener("change", changeListener);

// Add a button that displays the menu when the button is clicked
var listener = new Object();
listener.click = function(evtObj) {
    var button = evtObj.target; // get reference to the button

    // Display the menu at the bottom of the button
    _root.myMenu.show(button.x, button.y + button.height);
}
commandBtn.addEventListener("click", listener);
```

6. Select Control > Test Movie.

Click the Commands button to see the menu appear. When you select a menu item, a `trace()` statement reports the selection in the Output panel.

To use XML data from a server to create and populate a menu:

1. Select File > New and create a Flash document.
2. Drag the Menu component from the Components panel to the Stage and delete it.
This adds the Menu component to the library without adding it to the application. Menus are created dynamically through ActionScript.
3. In the Actions panel, add the following code to the first frame to create a menu and add some items:

```
var myMenu = mx.controls.Menu.createMenu();
// Import an XML file
var loader = new XML();
loader.menu = myMenu;
loader.ignoreWhite = true;
loader.onLoad = function(success) {
    // When the data arrives, pass it to the menu
    if(success) {
        this.menu.dataProvider = this.firstChild;
    }
};
loader.load(url);
```

Note: The menu items are described by the children of the XML document's first child.

4. Select Control > Test Movie.

To use a well-formed XML string to create and populate a menu:

1. Select File > New and create a Flash document.
2. Drag the Menu component from the Components panel to the Stage and delete it.
This adds the Menu component to the library without adding it to the application. Menus are created dynamically through ActionScript.
3. In the Actions panel, add the following code to the first frame to create a menu and add some items:

```
// Create an XML string containing a menu definition
var s = "";
s += "<menu>";
s += "<menuitem label='Undo' />";
s += "<menuitem type='separator' />";
s += "<menuitem label='Cut' />";
s += "<menuitem label='Copy' />";
s += "<menuitem label='Paste' />";
s += "<menuitem label='Clear' />";
s += "<menuitem type='separator' />";
s += "<menuitem label='Select All' />";
s += "</menu>";
// Create an XML object from the string
var xml = new XML(s);
xml.ignoreWhite = true;
// Create a menu from the XML object's first child
var myMenu = mx.controls.Menu.createMenu(_root, xml.firstChild);
```

4. Select Control > Test Movie.

To use the `MenuDataProvider` class to create and populate a menu:

1. Select File > New and create a Flash document.
2. Drag the Menu component from the Components panel to the Stage and delete it.
This adds the Menu component to the library without adding it to the application. Menus are created dynamically through `ActionScript`.
3. In the Actions panel, add the following code to the first frame to create a menu and add some items:

```
// Create an XML object to act as a factory.
var xml = new XML();

// The item created next will not appear in the menu.
// The createMenu() method call (below) expects to
// receive a root element whose children will become
// the items. This is just a simple way to create that
// root element and give it a convenient name along
// the way.
var theMenuElement = xml.addMenuItem("Edit");

// Add the menu items.
theMenuElement.addMenuItem({label:"Undo"});
theMenuElement.addMenuItem({type:"separator"});
theMenuElement.addMenuItem({label:"Cut"});
theMenuElement.addMenuItem({label:"Copy"});
theMenuElement.addMenuItem({label:"Paste"});
theMenuElement.addMenuItem({label:"Clear", enabled:"false"});
theMenuElement.addMenuItem({type:"separator"});
theMenuElement.addMenuItem({label:"Select All"});
// Create the Menu object.
var theMenuControl = mx.controls.Menu.createMenu(_root, theMenuElement);
```

4. Select Control > Test Movie.

Customizing the Menu component

The menu sizes itself horizontally to fit its widest text. You can also call the `setSize()` method to size the component. Icons should be sized to a maximum of 16 by 16 pixels.

Using styles with the Menu component

You can call the `setStyle()` method to change the style of the menu, its items, and its submenus. The Menu component supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>alternatingRowColors</code>	Both	Specifies colors for rows in an alternating pattern. The value can be an array of two or more colors, for example, <code>0xFF00FF</code> , <code>0xCC6699</code> , and <code>0x996699</code> . Unlike single-value color styles, <code>alternatingRowColors</code> does not accept color names; the values must be numeric color codes. By default, this style is not set, and <code>backgroundColor</code> is used in its place for all rows.
<code>backgroundColor</code>	Both	The background color of the menu. The default color is white and is defined on the class style declaration. This style is ignored if <code>alternatingRowColors</code> is specified.
<code>backgroundDisabledColor</code>	Both	The background color when the component's <code>enabled</code> property is set to "false". The default value is <code>0xDDDDDD</code> (medium gray).
<i>border styles</i>	Both	The Menu component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See "RectBorder class" in Flash Help. The default border style is "menuBorder".
<code>color</code>	Both	The text color.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is <code>0x848384</code> (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either "normal" or "italic". The default value is "normal".
<code>fontWeight</code>	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return "none".
<code>textAlign</code>	Both	The text alignment: either "left", "right", or "center". The default value is "left".

Style	Theme	Description
<code>textDecoration</code>	Both	The text decoration: either "none" or "underline". The default value is "none".
<code>textIndent</code>	Both	A number indicating the text indent. The default value is 0.
<code>defaultIcon</code>	Both	The name of the default icon to display on each row. The default value is <code>undefined</code> , which means no icon is displayed.
<code>popupDuration</code>	Both	The duration of the transition as a menu opens. The value is specified in milliseconds; 0 indicates no transition. The default value is 150.
<code>rolloverColor</code>	Both	The background color of a rolled-over row. The default value is <code>0xE3FFD6</code> (bright green) with the Halo theme and <code>0xA6AAAA</code> (light gray) with the Sample theme. When <code>themeColor</code> is changed through a <code>setStyle()</code> call, the framework sets <code>rolloverColor</code> to a value related to the <code>themeColor</code> chosen.
<code>selectionColor</code>	Both	The background color of a selected row. The default value is a <code>0xCDFFC1</code> (light green) with the Halo theme and <code>0xEEEEEE</code> (very light gray) with the Sample theme. When <code>themeColor</code> is changed through a <code>setStyle()</code> call, the framework sets <code>selectionColor</code> to a value related to the <code>themeColor</code> chosen.
<code>selectionDuration</code>	Both	The length of the transition from a normal to selected state, in milliseconds. The default value is 200.
<code>selectionEasing</code>	Both	A reference to the easing equation used to control the transition between selection states. The default equation uses a sine in/out formula. For more information, see "Customizing component animations" in Flash Help.
<code>textRolloverColor</code>	Both	The color of text when the mouse pointer rolls over. The default value is <code>0x2B333C</code> (dark gray). This style is important when you set <code>rolloverColor</code> , because the two settings must complement each other so that text is easily viewable during rollover.
<code>textSelectedColor</code>	Both	The color of text in the selected row. The default value is <code>0x005F33</code> (dark gray). This style is important when you set <code>selectionColor</code> , because the two must complement each other so that text is easily viewable while selected.
<code>useRollover</code>	Both	Determines whether rolling over a row activates highlighting. The default value is <code>true</code> .

Setting styles for all Menu components in a document

The Menu class inherits from the ScrollSelectList class. The default class-level style properties are defined on the ScrollSelectList class, which is shared by all List-based components. You can set new default style values on this class directly, and the new settings will be reflected in all affected components.

```
_global.styles.ScrollSelectList.setStyle("backgroundColor", 0xFF00AA);
```

To set a style property on the Menu components only, you can create a new CSSStyleDeclaration and store it in `_global.styles.Menu`.

```
import mx.styles.CSSStyleDeclaration;
if (_global.styles.Menu == undefined) {
    _global.styles.Menu = new CSSStyleDeclaration();
}
_global.styles.Menu.setStyle("backgroundColor", 0xFF00AA);
```

When creating a new class-level style declaration, you will lose all default values provided by the ScrollSelectList declaration. This includes `backgroundColor`, which is required for supporting mouse events. To create a class-level style declaration and preserve defaults, use a `for..in` loop to copy the old settings to the new declaration.

```
var source = _global.styles.ScrollSelectList;
var target = _global.styles.Menu;
for (var style in source) {
    target.setStyle(style, source.getStyle(style));
}
```

For more information about class-level styles see “Setting styles for a component class” in Flash Help.

Using skins with the Menu component

The Menu component uses an instance of RectBorder for its border (see “RectBorder class” in Flash Help).

The Menu component has visual assets for the branch, check mark, radio dot, and separator graphics. These assets are not dynamically skinnable, but the assets can be copied from the Flash UI Components 2/Themes/MMDefault/Menu Assets/States folder in both themes and modified as desired. The linkage identifiers cannot be changed, and all Menu instances must use the same symbols.

To create movie clip symbols for Menu assets:

1. Create a new FLA file.
2. Select File > Import > Open External Library, and select the HaloTheme.fla file.

This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in Flash Help.

3. In the theme’s Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the Menu Assets folder to the library for your document.
4. Expand the Menu Assets/States folder in the library of your document.

5. Open the symbols you want to customize for editing.

For example, open the `MenuCheckEnabled` symbol.

6. Customize the symbol as desired.

For example, change the image to be an X instead of a check mark.

7. Repeat steps 6-7 for all symbols you want to customize.

8. Click the Back button to return to the main Timeline.

9. Drag a Menu component to the Stage and delete it.

This adds the Menu component to the library and makes it available at runtime.

10. Add `ActionScript` to the main timeline to create a Menu instance at runtime:

```
var myMenu = mx.controls.Menu.createMenu();
myMenu.addItem({label: "One", type: "check", selected: true});
myMenu.addItem({label: "Two", type: "check"});
myMenu.addItem({label: "Three", type: "check"});
myMenu.show(0, 0);
```

11. Select `Control > Test Movie`.

Menu class

Inheritance `MovieClip` > `UIObject class` > `UIComponent class` > `View` > `ScrollView` > `ScrollSelectList` > `Menu`

ActionScript Class Name `mx.controls.Menu`

The methods and properties of the Menu class let you create and edit menus at runtime.

Setting a property of the menu class with `ActionScript` overrides the parameter of the same name set in the Property inspector or Component inspector.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.Menu.version);
```

Note: The code `trace(myMenuInstance.version);` returns `undefined`.

Method summary for the Menu class

The following table lists methods of the Menu class.

Method	Description
<code>Menu.addItem()</code>	Adds a menu item to the menu.
<code>Menu.addItemAt()</code>	Adds a menu item to the menu at a specific location.
<code>Menu.createMenu()</code>	Creates an instance of the Menu class. This is a static method.
<code>Menu.getItemAt()</code>	Gets a reference to a menu item at a specified location.
<code>Menu.hide()</code>	Closes a menu.

Method	Description
<code>Menu.indexOf()</code>	Returns the index of a given menu item.
<code>Menu.removeAll()</code>	Removes all items from a menu.
<code>Menu.removeItem()</code>	Removes the specified menu item.
<code>Menu.removeItemAt()</code>	Removes a menu item from a menu at a specified location.
<code>Menu.setMenuItemEnabled()</code>	Indicates whether a menu item is enabled (<code>true</code>) or not (<code>false</code>).
<code>Menu.setMenuItemSelected()</code>	Indicates whether a menu item is selected (<code>true</code>) or not (<code>false</code>).
<code>Menu.show()</code>	Opens a menu at a specific location or at its previous location.

Methods inherited from the UIObject class

The following table lists the methods the Menu class inherits from the UIObject class. When calling these methods from the Menu object, use the form *MenuInstance.methodName*.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it will be redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the Menu class inherits from the UIComponent class. When calling these methods from the Menu object, use the form *MenuInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the Menu class

The following table lists the property of the Menu class.

Property	Description
<code>Menu.dataProvider</code>	The data source for a menu.

Properties inherited from the UIObject class

The following table lists the properties the Menu class inherits from the UIObject class. When accessing these properties from the Menu object, use the form *MenuInstance.propertyName*.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the UIComponent class

The following table lists the properties the Menu class inherits from the UIComponent class. When accessing these properties from the Menu object, use the form *MenuInstance.propertyName*.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the Menu class

The following table lists events of the Menu class.

Event	Description
<code>Menu.change</code>	Broadcast when a user causes a change in a menu.
<code>Menu.menuHide</code>	Broadcast when a menu closes.
<code>Menu.menuShow</code>	Broadcast when a menu opens.
<code>Menu.rollOut</code>	Broadcast when the pointer rolls off an item.
<code>Menu.rollOver</code>	Broadcast when the pointer rolls over an item.

Events inherited from the UIObject class

The following table lists the events the Menu class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the Menu class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Menu.addItem()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
myMenu.addItem(initObject)
```

Usage 2:

```
myMenu.addItem(childMenuItem)
```

Parameters

initObject An object containing properties that initialize a menu item's attributes. See [“About menu item XML attributes” on page 327](#).

childMenuItem An XML node object.

Returns

A reference to the added XML node.

Description

Method; Usage 1 adds a menu item at the end of the menu. The menu item is constructed from the values supplied in the *initObject* parameter. Usage 2 adds a menu item that is a prebuilt XML node (in the form of an XML object) at the end of the menu. Adding a preexisting node removes the node from its previous location.

Example

Usage 1: The following example appends a menu item to a menu:

```
myMenu.addItem({label:"Item 1", type:"radio", selected:false,  
enabled:true, instanceName:"radioItem1", groupName:"myRadioGroup"});
```

Usage 2: The following example moves a node from one menu to the root of another menu:

```
myMenu.addItem(mySecondMenu.getItemAt(3));
```

Menu.addItemAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
myMenu.addItemAt(index, initObject)
```

Usage 2:

```
myMenu.addItemAt(index, childMenuItem)
```

Parameters

index An integer indicating the index position (among the child nodes) at which the item is added.

initObject An object containing properties that initialize a menu item's attributes. See [“About menu item XML attributes” on page 327](#).

childMenuItem An XML node object.

Returns

A reference to the added XML node.

Description

Method; Usage 1 adds a menu item (child node) at the specified location in the menu. The menu item is constructed from the values supplied in the *initObject* parameter. Usage 2 adds a menu item that is a prebuilt XML node (in the form of an XML object) at a specified location in the menu. Adding a preexisting node removes the node from its previous location.

Example

Usage 1: The following example adds a new node as the second child of the root of the menu:

```
myMenu.addItemAt(1, {label:"Item 1", instanceName:"radioItem1",
    type:"radio", selected:false, enabled:true, groupName:"myRadioGroup"});
```

Usage 2: The following example moves a node from one menu to the fourth child of the root of another menu:

```
myMenu.addItemAt(3, mySecondMenu.getItemAt(3));
```

Menu.change

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.change = function(eventObject){
    // insert your code here
}
myMenu.addEventListener("change", listenerObject)
```

Description

Event; broadcast to all registered listeners whenever a user causes a change in the menu.

Version 2 components use a dispatcher-listener event model. When a Menu component broadcasts a change event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Menu.change` event's event object has the following additional properties:

- `menuBar` A reference to the `MenuBar` instance that is the parent of the target menu. When the target menu does not belong to a `MenuBar` instance, this value is `undefined`.
- `menu` A reference to the `Menu` instance where the target item is located.
- `menuItem` An XML node that is the menu item that was selected.
- `groupName` A string indicating the name of the radio button group to which the item belongs. If the item is not in a radio button group, this value is `undefined`.

For more information, see “`EventDispatcher` class” in Flash Help.

Example

In the following example, a handler called `listener` is defined and passed to `myMenu.addEventListener()` as the second parameter. The event object is captured by the `change` handler in the `evt` parameter. When the `change` event is broadcast, a `trace` statement is sent to the Output panel.

```
listener = new Object();
listener.change = function(evt){
    trace("Menu item chosen: "+evt.menuItem.attributes.label);
}
myMenu.addEventListener("change", listener);
```

Menu.createMenu()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
Menu.createMenu([parent [, mdp]])
```

Parameters

parent A `MovieClip` instance. The movie clip is the parent component that contains the new `Menu` instance. This parameter is optional.

mdp The `MenuDataProvider` instance that describes this `Menu` instance. This parameter is optional.

Returns

A reference to the new menu instance.

Description

Method (static); instantiates a Menu instance, and optionally attaches it to the specified parent, with the specified MenuDataProvider as the data source for the menu items.

If the *parent* parameter is omitted or null, the Menu is attached to the `_root` Timeline.

If the *mdp* parameter is omitted or null, the menu does not have any items; you must call `addMenu()` or `setDataProvider()` to populate the menu.

Example

In the following example, line 1 creates a MenuDataProvider instance, which is an XML object that has the methods of the MenuDataProvider class.

In the following example, the first line creates an XML object instance, which is given the methods of the MenuDataProvider class (because the [MenuDataProvider class](#) is a decorator class of the XMLNode class). The next line adds a menu item (New) with a submenu (File, Project, and Resource). The next block of code adds more items to the main menu. The third block of code creates an empty menu attached to `myParentClip`, fills it with the data source `myMDP`, and opens it at the coordinates (100,20):

```
var myMDP:XML = new XML();
var newItem:Object = myMDP.addMenuItem({label:"New"});
    newItem.addMenuItem({label:"File..."});
    newItem.addMenuItem({label:"Project..."});
    newItem.addMenuItem({label:"Resource..."});

myMDP.addMenuItem({label:"Open", instanceName:"miOpen"});
myMDP.addMenuItem({label:"Save", instanceName:"miSave"});
myMDP.addMenuItem({type:"separator"});
myMDP.addMenuItem({label:"Quit", instanceName:"miQuit"});

var myMenu:mx.controls.Menu = mx.controls.Menu.createMenu(myParentClip,
    myMDP);
myMenu.show(100, 20);
```

To test this code, place it in the Actions panel on Frame 1 of the main Timeline. Drag a Menu component from the Components panel to the Stage and delete it. (This adds it to the library without placing it in the document.)

Menu.dataProvider

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenu.dataProvider
```

Description

Property; the data source for items in a Menu component.

`Menu.dataProvider` is an XML node object. Setting this property replaces the existing data source of the menu.

The default value is undefined.

Note: All XML or XMLNode instances are automatically given the methods and properties of the MenuDataProvider class when they are used with the Menu component.

Example

The following example imports an XML file and assigns it to the `Menu.dataProvider` property:

```
var myMenuDP = new XML();
myMenuDP.load("http://myServer.myDomain.com/source.xml");
myMenuDP.onLoad = function(){
    myMenuControl.dataProvider = myMenuDP;
}
```

Menu.getMenuItemAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenu.getMenuItemAt(index)
```

Parameters

index An integer indicating the index of the node in the menu.

Returns

A reference to the specified node.

Description

Method; returns a reference to the specified child node of the menu.

Example

The following example gets a reference to the second child node in `myMenu` and copies the value into the variable `myItem`:

```
var myItem = myMenu.getMenuItemAt(1);
```

Menu.hide()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenu.hide()
```

Parameters

None.

Returns

Nothing.

Description

Method; closes a menu.

Example

The following example retracts an extended menu:

```
myMenu.hide();
```

See also

[Menu.show\(\)](#)

Menu.indexOf()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenu.indexOf(item)
```

Parameters

item A reference to an XML node that describes a menu item.

Returns

The index of the specified menu item, or `undefined` if the item does not belong to this menu.

Description

Method; returns the index of the specified menu item within this menu instance.

Example

The following example adds a menu item to a parent item and then gets the item's index within its parent:

```
var myItem = myMenu.addItem({label:"That item"});
var myIndex = myMenu.indexOf(myItem);
```

Menu.menuHide

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.menuHide = function(eventObject){
    // insert your code here
}
myMenu.addEventListener("menuHide", listenerObject)
```

Description

Event; broadcast to all registered listeners whenever a menu closes.

Version 2 components use a dispatcher-listener event model. When a Menu component dispatches a `menuHide` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler and the name of the listener object as parameters.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Menu.menuHide` event's event object has two additional properties:

- `menuBar` A reference to the `MenuBar` instance that is the parent of the target menu. When the target menu does not belong to a `MenuBar` instance, this value is `undefined`.
- `menu` A reference to the `Menu` instance that is hidden.

For more information, see “EventDispatcher class” in Flash Help.

Example

In the following example, a handler called `form` is defined and passed to `myMenu.addEventListener()` as the second parameter. The event object is captured by the `menuHide` handler in the `evt` parameter. When the `menuHide` event is broadcast, a trace statement is sent to the Output panel.

```
form = new Object();
form.menuHide = function(evt){
    trace("Menu closed: "+evt.menu);
}
myMenu.addEventListener("menuHide", form);
```

See also

[Menu.menuShow](#)

Menu.menuShow

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.menuShow = function(eventObject){
    // insert your code here
}
myMenu.addEventListener("menuShow", listenerObject)
```

Description

Event; broadcast to all registered listeners whenever a menu opens. All parent nodes open menus to show their children.

Version 2 components use a dispatcher-listener event model. When a Menu component dispatches a `menuShow` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler and listener object as parameters.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Menu.menuShow` event's event object has two additional properties:

- `menuBar` A reference to the `MenuBar` instance that is the parent of the target menu. When the target menu does not belong to a `MenuBar` instance, this value is `undefined`.
- `menu` A reference to the `Menu` instance that is shown.

For more information, see “EventDispatcher class” in Flash Help.

Example

In the following example, a handler called `form` is defined and passed to `myMenu.addEventListener()` as the second parameter. The event object is captured by the `menuShow` handler in the `evt` parameter. When the `menuShow` event is broadcast, a `trace` statement is sent to the Output panel.

```
form = new Object();
form.menuShow = function(evt){
    trace("Menu opened: "+evt.menu);
}
myMenu.addEventListener("menuShow", form);
```

See also

[Menu.menuHide](#)

Menu.removeAll()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenu.removeAll();
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all items and refreshes the menu.

Example

The following example removes all nodes from the menu:

```
myMenu.removeAll();
```

Menu.removeItem()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
menuItem.removeMenuItem();
```

Returns

A reference to the returned menu item (XML node). This value is undefined if there is no item in that position.

Description

Method; removes the specified menu item and all its children, and refreshes the menu.

Example

The following example removes the menu item referenced by the variable `theItem`:

```
theItem.removeMenuItem();
```

Menu.removeMenuItemAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenu.removeMenuItemAt(index)
```

Parameters

index The index of the menu item to remove.

Returns

A reference to the returned menu item (XML node). This value is undefined if there is no item in that position.

Description

Method; removes the menu item and all its children at the specified index. If there is no menu item at that index, calling this method has no effect.

Example

The following example removes a menu item at index 3:

```
var item = myMenu.removeMenuItemAt(3);
```

Menu.rollOut

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();  
listenerObject.rollOut = function(eventObject){  
    // insert your code here  
}  
myMenu.addEventListener("rollOut", listenerObject)
```

Description

Event; broadcast to all registered listeners when the pointer rolls off a menu item.

Version 2 components use a dispatcher-listener event model. When a Menu component broadcasts a `rollOut` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Menu.rollOut` event's event object has one additional property: `menuItem`, which is a reference to the menu item (XML node) that the pointer rolled off.

For more information, see "EventDispatcher class" in Flash Help.

Example

In the following example, a handler called `form` is defined and passed to `myMenu.addEventListener()` as the second parameter. The event object is captured by the `rollOut` handler in the `evt` parameter. When the `rollOut` event is broadcast, a trace statement is sent to the Output panel.

```
form = new Object();
form.rollOut = function(evt){
    trace("Menu rollOut: "+evt.menuItem.attributes.label);
}
myMenu.addEventListener("rollOut", form);
```

Menu.rollOver

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.rollOver = function(eventObject){
    // insert your code here
}
myMenu.addEventListener("rollOver", listenerObject)
```

Description

Event; broadcast to all registered listeners when the pointer rolls over a menu item.

Version 2 components use a dispatcher-listener event model. When a Menu component broadcasts a `rollOver` event, the event is handled by a function (also called a *handler*) that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Menu.rollOver` event's event object has one additional property: `menuItem`, which is a reference to the menu item (XML node) that the pointer rolled over.

For more information, see “EventDispatcher class” in Flash Help.

Example

In the following example, a handler called `form` is defined and passed to `myMenu.addEventListener()` as the second parameter. The event object is captured by the `rollOver` handler in the `evt` parameter. When the `rollOver` event is broadcast, a trace statement is sent to the Output panel.

```
form = new Object();
form.rollOver = function(evt){
    trace("Menu rollOver: "+evt.menuItem.attributes.label);
}
myMenu.addEventListener("rollOver", form);
```

Menu.setMenuItemEnabled()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenu.setMenuItemEnabled(item, enable)
```

Parameters

item An XML node; the target menu item's node in the data provider.

enable A Boolean value indicating whether the item is enabled (`true`) or not (`false`).

Returns

Nothing.

Description

Method; changes the target item's `enabled` attribute to the state specified in the `enable` parameter. If this call results in a change of state, the item is redrawn with the new state.

Example

The following example disables the second child of `myMenu`:

```
var myItem = myMenu.getMenuItemAt(1);
myMenu.setMenuItemEnabled(myItem, false);
```

See also

[Menu.setMenuItemSelected\(\)](#)

Menu.setMenuItemSelected()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenu.setMenuItemSelected(item, select)
```

Parameters

item An XML node. The target menu item's node in the data provider.

select A Boolean value indicating whether the item is selected (`true`) or not (`false`). If the item is a check box, its check mark is visible or not visible. If a selected item is a radio button, it becomes the current selection in the radio group.

Returns

Nothing.

Description

Method; changes the `selected` attribute of the item to the state specified by the `select` parameter. If this call results in a change of state, the item is redrawn with the new state. This is only meaningful for items whose `type` attribute is set to "radio" or "check", because it causes their dot or check to appear or disappear. If you call this method on an item whose type is "normal" or "separator", it has no effect.

Example

The following example deselects the second child of `myMenu`:

```
var myItem = myMenu.getMenuItemAt(1);  
myMenu.setMenuItemSelected(myItem, false);
```

Menu.show()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenu.show(x, y)
```

Parameters

- x The x coordinate.
- y The y coordinate.

Returns

Nothing.

Description

Method; opens a menu at a specific location. The menu is automatically resized so that all of its top-level items are visible, and the upper left corner is placed at the specified location in the coordinate system provided by the component's parent.

If the x and y parameters are omitted, the menu is shown at its previous location.

Example

The following example displays a menu 10 pixels down and to the right of the (0,0) origin point of the component's parent:

```
myMenu.show(10, 10);
```

See also

[Menu.hide\(\)](#)

MenuDataProvider class

ActionScript Class Name mx.controls.menuelements.MenuDataProvider

The MenuDataProvider class is a decorator (mix-in) class that adds functionality to the XMLNode global class. This functionality lets XML instances assigned to a Menu.dataProvider property use the MenuDataProvider methods and properties to manipulate their own data as well as the associated menu views.

Keep in mind these concepts about the MenuDataProvider class:

- MenuDataProvider is a decorator (mix-in) class. You do not need to instantiate it to use it.
- Menus natively accept XML as a dataProvider property value.
- If a Menu class is instantiated, all XML instances in the SWF file are decorated by the MenuDataProvider class.
- Only MenuDataProvider methods broadcast events to the Menu components. You can still use native XML methods, but they do not broadcast events that refresh the Menu views. To control the data model, use MenuDataProvider methods. For read-only operations like moving through the Menu hierarchy, use XML methods.
- All items in the Menu component are XML objects decorated with the MenuDataProvider class.
- Changes to item attributes are not reflected in the onscreen menu until redrawing occurs.

Method summary for the MenuDataProvider class

The following table lists the methods of the MenuDataProvider class.

Method	Description
<code>MenuDataProvider.addItem()</code>	Adds a child item.
<code>MenuDataProvider.addItemAt()</code>	Adds a child item at a specified location.
<code>MenuDataProvider.getItemAt()</code>	Gets a reference to a menu item at a specified location.
<code>MenuDataProvider.indexOf()</code>	Returns the index of a specified menu item.
<code>MenuDataProvider.removeItem()</code>	Removes a menu item.
<code>MenuDataProvider.removeItemAt()</code>	Removes a menu item at a specified location.

MenuDataProvider.addItem()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
myMenuDataProvider.addItem(initObject)
```

Usage 2:

```
myMenuDataProvider.addItem(childMenuItem)
```

Parameters

initObject An object containing the attributes that initialize a Menu item's attributes. For more information, see [“About menu item XML attributes” on page 327](#).

childMenuItem An XML node.

Returns

A reference to an XMLNode object.

Description

Method; Usage 1 adds a child item to the end of a parent menu item (which could be the menu itself). The menu item is constructed from the values passed in the *initObject* parameter.

Usage 2 adds a child item that is defined in the specified XML *childMenuItem* parameter to the end of a parent menu item.

Any node or menu item in a MenuDataProvider instance can call the methods of the MenuDataProvider class.

Example

The following example adds a new node to a specified node in the menu:

```
var item0 = myMenuDP.getMenuItemAt(0);
item0.addMenuItem("Inbox", { label:"Item 1", icon:"radioItemIcon",
    type:"radio", selected:false, enabled:true, instanceName:"radioItem1",
    groupName:"myRadioGroup" } );
```

MenuDataProvider.addItemAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
myMenuDataProvider.addItemAt(index, initObject)
```

Usage 2:

```
myMenuDataProvider.addItemAt(index, childMenuItem)
```

Parameters

index An integer.

initObject An object containing the specific attributes that initialize a Menu item's attributes. For more information, see [“About menu item XML attributes” on page 327](#).

childMenuItem An XML node.

Returns

A reference to the added XML node.

Description

Method; Usage 1 adds a child item at the specified index position in the parent menu item (which could be the menu itself). The menu item is constructed from the values passed in the *initObject* parameter. Usage 2 adds a child item that is defined in the specified XML *childMenuItem* parameter to the specified index of a parent menu item.

Any node or menu item in a MenuDataProvider instance can call the methods of the MenuDataProvider class.

Example

Usage 1: The following example adds a new node as the second child of the root of the menu:

```
myMenuDataProvider.addItemAt(1, {label:"Item 1", type:"radio",
    selected:false, enabled:true, instanceName:"radioItem1",
    groupName:"myRadioGroup"});
```

MenuDataProvider.getMenuItemAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenuDataProvider.getMenuItemAt(index)
```

Parameters

index An integer indicating the position of the menu.

Returns

A reference to the specified XML node.

Description

Method; returns a reference to the specified child menu item of the current menu item.

Any node or menu item in a MenuDataProvider instance can call the methods of the MenuDataProvider class.

Example

The following example finds the node you want to get, and then gets the second child of myMenuItem:

```
var myMenuItem = myMenuDP.firstChild.firstChild;  
myMenuItem.getMenuItemAt(1);
```

MenuDataProvider.indexOf()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenuDataProvider.indexOf(item)
```

Parameters

item A reference to the XML node that describes the menu item.

Returns

The index of the specified menu item; returns `undefined` if the item does not belong to this menu.

Description

Method; returns the index of the specified menu item in this parent menu item.

Any node or menu item in a `MenuDataProvider` instance can call the methods of the `MenuDataProvider` class.

Example

The following example adds a menu item to a parent item and gets the item's index:

```
var myMenuItem = myParentMenuItem.addItem({label:"That item"});  
var myIndex = myParentMenuItem.indexOf(myItem);
```

`MenuDataProvider.removeItem()`

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenuDataProvider.removeMenuItem()
```

Parameters

None.

Returns

A reference to the removed Menu item (XML node); undefined if an error occurs.

Description

Method; removes the target item and any child nodes.

Any node or menu item in a `MenuDataProvider` instance can call the methods of the `MenuDataProvider` class.

Example

The following example removes `myMenuItem` from its parent:

```
myMenuItem.removeItem();
```

`MenuDataProvider.removeItemAt()`

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenuDataProvider.removeMenuItemAt(index)
```

Parameters

index The index of the menu item.

Returns

A reference to the removed menu item. This value is `undefined` if there is no item in that position.

Description

Method; removes the child item of the menu item specified by the *index* parameter. If there is no menu item at that index, calling this method has no effect.

Any node or menu item in a `MenuDataProvider` instance can call the methods of the `MenuDataProvider` class.

Example

The following example removes the fourth item:

```
myMenuDataProvider.removeMenuItemAt(3);
```

MenuBar component

The `MenuBar` component lets you create a horizontal menu bar with pop-up menus and commands, just like the menu bars that contain File and Edit menus in common software applications. The `MenuBar` component complements the `Menu` component by providing a clickable interface to show and hide menus that behave as a group for mouse and keyboard interactivity.

The `MenuBar` component lets you create an application menu in a few steps. To build a menu bar, you can either assign an XML data provider to the menu bar that describes a series of menus, or use the `MenuBar.addMenu()` method to add menu instances one at a time.

Each menu in the menu bar is composed of two parts: the menu and the button that causes the menu to open (called the menu activator). These clickable menu activators appear in the menu bar as a text label with inset and outset border highlight states that react to interaction from the mouse and keyboard.

When a menu activator is clicked, the corresponding menu opens below it. The menu stays active until the activator is clicked again, or until a menu item is selected or a click occurs outside the menu area.

In addition to creating menu activators that show and hide menus, the `MenuBar` component creates group behavior among a series of menus. This lets a user scan a large number of command choices by rolling over the series of activators or by using the arrow keys to move through the lists. Mouse and keyboard interactivity work together to let the user jump from menu to menu in the menu bar.

A user cannot scroll through menus on a menu bar. If menus exceed the width of the menu bar, they are masked.

You cannot make the `MenuBar` component accessible to screen readers.

Interacting with the MenuBar component

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

Rolling over a menu activator displays an outset border highlight around the activator label.

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

Key	Description
Down Arrow	Moves the selection down a menu row.
Up Arrow	Moves the selection up a menu row.
Right Arrow	Moves the selection to the next button.
Left Arrow	Moves the selection to the previous button.
Enter/Escape	Closes an open menu.

Note: If a menu is open, you can't press the Tab key to close it. You must either make a selection or close the menu by pressing Escape.

Using the MenuBar component

You can use the MenuBar component to add a set of menus (for example, File, Edit, Special, Window) to the top edge of an application.

MenuBar parameters

You can set the following authoring parameter for each MenuBar component instance in the Property inspector or in the Component inspector:

Labels An array that adds menu activators with the specified labels to the MenuBar component. The default value is [] (an empty array).

You cannot access the Labels parameter using ActionScript. However, you can write ActionScript to control additional options for the MenuBar component using its properties, methods, and events. For more information, see [“MenuBar class” on page 363](#).

Creating an application with the MenuBar component

In this example, you drag a MenuBar component to the Stage, add code to fill the instance with menus, and attach listeners to the menus to respond to menu item selection.

To use a MenuBar component in an application:

1. Select File > New and create a new Flash document.
2. Drag the MenuBar component from the Components panel to the Stage.
3. Position the menu at the top of the Stage for a standard layout.
4. Select the MenuBar instance, and in the Property inspector, enter the instance name **myMenuBar**.

5. In the Actions panel on Frame 1, enter the following code:

```
var menu = myMenuBar.addMenu("File");
menu.addItem({label:"New", instanceName:"newInstance"});
menu.addItem({label:"Open", instanceName:"openInstance"});
menu.addItem({label:"Close", instanceName:"closeInstance"});
```

This code adds a File menu to the MenuBar instance. It then uses a Menu method to add three menu items: New, Open, and Close.

6. In the Actions panel on Frame 1, enter the following code:

```
var listen = new Object();
listen.change = function(evt){
    var menu = evt.menu;
    var item = evt.menuItem
    if (item == menu.newInstance){
        myNew();
        trace(item);
    }else if (item == menu.openInstance){
        myOpen();
        trace(item);
    }
}
menu.addEventListener("change",listen);
```

This code creates a listener object, `listen`, that uses the event object, `evt`, to catch menu item selections.

Note: You must call the `addEventListener()` method to register the listener with the menu instance, not with the menu bar instance.

7. Select Control > Test Movie to test the MenuBar component.

Customizing the MenuBar component

This component sizes itself according to the activator labels that are supplied through the `dataProvider` property or the methods of the MenuBar class. When an activator button is in a menu bar, it remains at a fixed size that is dependent on the font styles and the text length.

Using styles with the MenuBar component

The MenuBar component creates an activator label for each menu in a group. You can use styles to change the look of the activator labels. A MenuBar component supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>color</code>	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).

Style	Theme	Description
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either <code>"normal"</code> or <code>"italic"</code> . The default value is <code>"normal"</code> .
<code>fontWeight</code>	Both	The font weight: either <code>"none"</code> or <code>"bold"</code> . The default value is <code>"none"</code> . All components can also accept the value <code>"normal"</code> in place of <code>"none"</code> during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return <code>"none"</code> .
<code>textDecoration</code>	Both	The text decoration: either <code>"none"</code> or <code>"underline"</code> . The default value is <code>"none"</code> .

The `MenuBar` component also forwards all style settings for `Menu` style properties to the composed `Menu` instances. For a list of `Menu` style properties, see [“Using styles with the Menu component” on page 334](#).

Using skins with the `MenuBar` component

The `MenuBar` component uses three skins to represent its background, uses a movie clip symbol for highlighting individual items, and contains a `Menu` component as the pop-up which itself is skinnable. The `MenuBar` skins are described below. For information on skinning the `Menu` component, see [“Using skins with the Menu component” on page 336](#).

The `MenuBar` component supports the following skin properties.

Property	Description
<code>menuBarBackLeftName</code>	The up state of the pop-up icon.
<code>menuBarBackRightName</code>	The down state of the pop-up icon.
<code>menuBarBackMiddleName</code>	The disabled state of the pop-up icon.

To create movie clip symbols for `MenuBar` skins:

1. Create a new FLA file.
2. Select `File > Import > Open External Library`, and select the `HaloTheme.fla` file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see [“About themes” in Flash Help](#).
3. In the theme’s `Library` panel, expand the `Flash UI Components 2/Themes/MMDefault` folder and drag the `MenuBar Assets` folder to the library for your document.
4. Expand the `MenuBar Assets/Elements` folder in the library of your document.

5. Open the symbols you want to customize for editing.
For example, open the `MenuBarBackLeft` symbol.
6. Customize the symbol as desired.
For example, change the outer edge to blank.
7. Repeat steps 5-6 for all symbols you want to customize.
For example, set the outer edges for the middle and right symbols to black.
8. Click the Back button to return to the main Timeline.
9. Drag a `MenuBar` component to the Stage.
10. Set `MenuBar` properties so that they display items on the bar.
11. Select `Control > Test Movie`.

Note: The border used to highlight individual items in a `MenuBar` component is an instance of `ActivatorSkin` found in the Flash UI Components 2/Themes/MMDefault/Button Assets folder. This symbol can be customized to point to a different class to provide a different border. However, the symbol name cannot be modified, and you cannot use a different symbol for different `MenuBar` instances in a single document.

MenuBar class

Inheritance `MovieClip` > [UIObject class](#) > [UIComponent class](#) > `MenuBar`

ActionScript Class Name `mx.controls.MenuBar`

The methods and properties of the `MenuBar` class let you create a horizontal menu bar with pop-up menus and commands. These methods and properties complement those of the `Menu` class by allowing you to create a clickable interface to show and hide menus that behave as a group for mouse and keyboard interactivity.

Method summary for the MenuBar class

The following table lists methods of the `MenuBar` class.

Method	Description
<code>MenuBar.addMenu()</code>	Adds a menu to the menu bar.
<code>MenuBar.addMenuAt()</code>	Adds a menu at a specified location to the menu bar.
<code>MenuBar.getMenuAt()</code>	Gets a reference to a menu at a specified location.
<code>MenuBar.getMenuEnabledAt()</code>	Returns a Boolean value indicating whether a menu is enabled (<code>true</code>) or not (<code>false</code>).
<code>MenuBar.removeMenuAt()</code>	Removes a menu at a specified location from a menu bar.
<code>MenuBar.setMenuEnabledAt()</code>	A Boolean value indicating whether a menu is can be chosen (<code>true</code>) or not (<code>false</code>).

Methods inherited from the UIObject class

The following table lists the methods the MenuBar class inherits from the UIObject class. When calling these methods from the MenuBar object, use the form `MenuBar.methodName`.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it will be redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the MenuBar class inherits from the UIComponent class. When calling these methods from the MenuBar object, use the form `MenuBar.methodName`.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the MenuBar class

The following table lists properties of the MenuBar class.

Property	Description
<code>MenuBar.dataProvider</code>	The data model for a menu bar.
<code>MenuBar.labelField</code>	A string that determines which attribute of each XMLNode to use as the label text of the menu.
<code>MenuBar.labelFunction</code>	A function that determines what to display in each menu's label.

Properties inherited from the UIObject class

The following table lists the properties the MenuBar class inherits from the UIObject class. When calling these properties from the MenuBar object, use the form `MenuBar.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the UIComponent class

The following table lists the properties the MenuBar class inherits from the UIComponent class. When calling these properties from the MenuBar object, use the form `MenuBar.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the MenuBar class

There are no events exclusive to the MenuBar class.

Events inherited from the UIObject class

The following table lists the events the MenuBar class inherits from the UIObject class. When calling these events from the MenuBar object, use the form `MenuBar.eventName`.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the MenuBar class inherits from the UIComponent class. When calling these events from the MenuBar object, use the form `MenuBar.eventName`.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

MenuBar.addMenu()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
myMenuBar.addMenu(label)
```

Usage 2:

```
myMenuBar.addMenu(label, menuDataProvider)
```

Parameters

label A string indicating the label of the new menu.

menuDataProvider An XML or XMLNode instance that describes the menu and its items. If the value is an XML instance, the instance's first child is used.

Returns

A reference to the new Menu object.

Description

Method; Usage 1 adds a single menu and menu activator at the end of the menu bar and uses the specified label. Usage 2 adds a single menu and menu activator that are defined in the specified XML *menuDataProvider* parameter.

Example

Usage 1: The following example adds a File menu and then uses `Menu.addItem()` to add the menu items `New` and `Open`:

```
var myMenuBar:mx.controls.MenuBar;
var myMenu:mx.controls.Menu;

myMenu = myMenuBar.addMenu("File");
myMenu.addItem({label:"New", instanceName:"newInstance"});
myMenu.addItem({label:"Open", instanceName:"openInstance"})
```

Usage 2: The following example adds a Font menu with the menu items `Bold` and `Italic` that are defined in the `MenuDataProvider` instance `myMenuDP2`:

```
var myMenuDP2 = new XML();
myMenuDP2.addItem({type:"check", label:"Bold", instanceName:"check1"});
myMenuDP2.addItem({type:"check", label:"Italic", instanceName:"check2"});
menu = myMenuBar.addMenu("Font",myMenuDP2);
```

MenuBar.addMenuAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
myMenuBar.addMenuAt(index, label)
```

Usage 2:

```
myMenuBar.addMenuAt(index, label, menuDataProvider)
```

Parameters

index An integer indicating the position where the menu should be inserted. The first position is 0. To append to the end of the menu, call `MenuBar.addMenu(label)`.

label A string indicating the label of the new menu.

menuDataProvider An XML or XMLNode instance that describes the menu. If the value is an XML instance, the instance's first child is used.

Returns

A reference to the new Menu object.

Description

Method; Usage 1 adds a single menu and menu activator at the specified index with the specified label. Usage 2 adds a single menu and a labeled menu activator at the specified index. The content for the menu is defined in the *menuDataProvider* parameter.

Example

Usage 1: The following example places a menu to the left of all MenuBar menus:

```
menu = myMenuBar.addMenuAt(0, "Toreador");
menu.addItem("About Macromedia Flash", instanceName:"aboutInst");
menu.addItem("Preferences", instanceName:"PrefInst");
```

Usage 2: The following example adds an Edit menu with the menu items Undo, Redo, Cut, and Copy, which are defined in the MenuDataProvider instance myMenuDP:

```
var myMenuDP = new XML();
myMenuDP.addItem({label:"Undo", instanceName:"undoInst"});
myMenuDP.addItem({label:"Redo", instanceName:"redoInst"});
myMenuDP.addItem({type:"separator"});
myMenuDP.addItem({label:"Cut", instanceName:"cutInst"});
myMenuDP.addItem({label:"Copy", instanceName:"copyInst"});

myMenuBar.addMenuAt(0, "Edit", myMenuDP);
```

MenuBar.dataProvider

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenuBar.dataProvider
```

Description

Property; the data model for items in a MenuBar component.

`MenuBar.dataProvider` is an XML node object. Setting this property replaces the existing data model of the `MenuBar` component. Whatever child nodes the data provider might have are used as the items for the menu bar itself; any subnodes of these child nodes are used as the items for their respective menus.

The default value is undefined.

Note: All XML or XMLNode instances are automatically given the methods and properties of the `MenuDataProvider` class when they are used with the `MenuBar` component.

Example

The following example imports an XML file and assigns it to the `MenuBar.dataProvider` property:

```
var myMenuBarDP = new XML();
myMenuBarDP.load("http://myServer.myDomain.com/source.xml");
myMenuBarDP.onLoad = function(success){
    if(success){
        myMenuBar.dataProvider = myMenuBarDP;
    } else {
        trace("error loading XML file");
    }
}
```

MenuBar.getMenuAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenuBar.getMenuAt(index)
```

Parameters

index An integer indicating the position of the menu.

Returns

A reference to the menu at the specified index. This value is undefined if there is no menu at that position.

Description

Method; returns a reference to the menu at the specified index.

Example

Because `getMenuAt()` returns an instance, it is possible to add items to a menu at the specified index. In the following example, after using the Labels authoring parameter to create the menu activators File, Edit, and View, the following code adds New and Open items to the File menu at runtime:

```
menu = myMenuBar.getMenuAt(0);
menu.addItem({label:"New",instanceName:"newInst"});
menu.addItem({label:"Open",instanceName:"openInst"});
```

MenuBar.getMenuEnabledAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenuBar.getMenuEnabledAt(index)
```

Parameters

index The index of the menu in the menu bar.

Returns

A Boolean value that indicates whether this menu can be chosen (`true`) or not (`false`).

Description

Method; returns a Boolean value that indicates whether this menu can be chosen (`true`) or not (`false`).

Example

The following example calls the method on the menu in the first position of `myMenuBar`:

```
myMenuBar.getMenuEnabledAt(0);
```

MenuBar.labelField

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenuBar.labelField
```

Description

Property; a string that determines which attribute of each XML node to use as the label text of the menu. This property is also passed to any menus that are created from the menu bar. The default value is "label".

After the `dataProvider` property is set, this property is read-only.

Example

The following example uses the `name` attribute of each node as the label text:

```
myMenuBar.labelField = "name";
```

MenuBar.labelFunction

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenuBar.labelFunction
```

Description

Property; a function that determines what to display in each menu's label text. The function accepts the XML node associated with an item as a parameter and returns a string to be used as label text. This property is passed to any menus created from the menu bar. The default value is undefined.

After the `dataProvider` property is set, this property is read-only.

Example

The following example of a label function builds and returns a custom label from the node attributes:

```
myMenuBar.labelFunction = function(node){  
    var a = node.attributes;  
    return "The Price for " + a.name + " is " + a.price;  
};
```

MenuBar.removeMenuAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenuBar.removeMenuAt(index)
```

Parameters

index The index of the menu to be removed from the menu bar.

Returns

A reference to the menu at the specified index in the menu bar. This value is `undefined` if there is no menu in that position in the menu bar.

Description

Method; removes the menu at the specified index. If there is no menu item at that index, calling this method has no effect.

Example

The following example removes the menu at index 4:

```
myMenuBar.removeMenuAt(4);
```

MenuBar.setMenuEnabledAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myMenuBar.setMenuEnabledAt(index, boolean)
```

Parameters

index The index of the menu item to set in the MenuBar instance.

boolean A Boolean value indicating whether the menu item at the specified index is enabled (`true`) or not (`false`).

Returns

Nothing.

Description

Method; enables the menu at the specified index. If there is no menu at that index, calling this method has no effect.

Example

The following example enables the item at index 3 in the MenuBar object `myMenuBar`:

```
myMenuBar.setMenuEnabledAt(3, true);
```

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.

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).

A stepper can be enabled or disabled in an application. In the disabled state, a stepper doesn't receive mouse or keyboard input. An enabled stepper 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 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 “Creating custom focus navigation” in Flash Help or “[FocusManager class](#)” on page 231.

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

When you add the NumericStepper component to an application, you can use the Accessibility panel to make it accessible to screen readers. First, you must add the following line of code to enable accessibility:

```
mx.accessibility.NumericStepperAccImpl.enableAccessibility();
```

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

Using the NumericStepper component

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

NumericStepper parameters

You can set the following authoring parameters for each NumericStepper instance in the Property inspector or in the Component inspector:

value sets the value displayed in the text area of the stepper. The default value is 0.

minimum sets the minimum value that can be displayed in the stepper. The default value is 0.

maximum sets the maximum value that can be displayed in the stepper. The default value is 10.

stepSize sets the unit by which the stepper increases or decreases with each click. The default value is 1.

You can write ActionScript to control these and additional options for the NumericStepper component using its properties, methods, and events. For more information, see [“NumericStepper class” on page 377](#).

Creating an application with the NumericStepper component

The following procedure explains how to add a NumericStepper component to an application while authoring. In this example, the stepper allows a user to pick a movie rating from 0 to 5 stars with half-star increments.

To create an application with the NumericStepper component:

1. Drag a NumericStepper component from the Components panel to the Stage.
2. In the Property inspector, enter the instance name **starStepper**.
3. In the Property inspector, do the following:
 - Enter 0 for the minimum parameter.
 - Enter 5 for the maximum parameter.
 - Enter .5 for the stepSize parameter.
 - Enter 0 for the value parameter.
4. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code:

```
movieRate = new Object();
movieRate.change = function (eventObject){
    starChart.value = eventObject.target.value;
}
starStepper.addEventListener("change", movieRate);
```

The last line of code adds a change event handler to the starStepper instance. The handler sets the starChart movie clip to display the amount of stars indicated by the starStepper instance. (To see this code work, you must create a starChart movie clip with a value property that displays the stars.)

Customizing the NumericStepper component

You can transform a NumericStepper component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)) or any applicable properties and methods of the NumericStepper class. (See “NumericStepper class” on page 377.)

Resizing the NumericStepper component does not change the size of the down and up arrow buttons. If the stepper is resized to be greater than the default height, the arrow buttons are pinned to the top and bottom of the component. The arrow buttons always appear to the right of the text box.

Using styles with the NumericStepper component

You can set style properties to change the appearance of a NumericStepper instance. If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in Flash Help.

A NumericStepper component supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>color</code>	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is "_sans".
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either "normal" or "italic". The default value is "normal".
<code>fontWeight</code>	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return "none".
<code>textAlign</code>	Both	The text alignment: either "left", "right", or "center". The default value is "center".

Style	Theme	Description
<code>textDecoration</code>	Both	The text decoration: either "none" or "underline". The default value is "none".
<code>repeatDelay</code>	Both	The number of milliseconds of delay between when a user first presses a button and when the action begins to repeat. The default value is 500 (half a second).
<code>repeatInterval</code>	Both	The number of milliseconds between automatic clicks when a user holds the mouse button down on a button. The default value is 35.
<code>symbolColor</code>	Sample	The color of the arrows. The default value is 0x2B333C (dark gray).

Using skins with the NumericStepper component

The NumericStepper component uses skins to represent its up and down button states. To skin the NumericStepper component while authoring, modify skin symbols in the Flash UI Components 2/Themes/MMDefault/Stepper Assets/States folder in the library. For more information, see “About skinning components” in Flash Help.

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 supports the following skin properties:

Property	Description
<code>upArrowUp</code>	The up arrow button's up state. The default value is <code>StepUpArrowUp</code> .
<code>upArrowDown</code>	The up arrow button's pressed state. The default value is <code>StepUpArrowDown</code> .
<code>upArrowOver</code>	The up arrow button's over state. The default value is <code>StepUpArrowOver</code> .
<code>upArrowDisabled</code>	The up arrow button's disabled state. The default value is <code>StepUpArrowDisabled</code> .
<code>downArrowUp</code>	The down arrow button's up state. The default value is <code>StepDownArrowUp</code> .
<code>downArrowDown</code>	The down arrow button's down state. The default value is <code>StepDownArrowDown</code> .
<code>downArrowOver</code>	The down arrow button's over state. The default value is <code>StepDownArrowOver</code> .
<code>downArrowDisabled</code>	The down arrow button's disabled state. The default value is <code>StepDownArrowDisabled</code> .

To create movie clip symbols for NumericStepper skins:

1. Create a new FLA file.
2. Select File > Import > Open External Library, and select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in Flash Help.
3. In the theme’s Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the Stepper Assets folder to the library for your document.
4. Expand the Stepper Assets folder in the library of your document.
5. Expand the Stepper Assets/States folder in the library of your document.
6. Open the symbols you want to customize for editing.
For example, open the StepDownArrowDisabled symbol.
7. Customize the symbol as desired.
For example, change the white inner graphics to a light gray.
8. Repeat steps 6-7 for all symbols you want to customize.
For example, repeat the same change on the up arrow.
9. Click the Back button to return to the main Timeline.
10. Drag a NumericStepper component to the Stage.
This example has customized the disabled skins, so use ActionScript to set the NumericStepper instance to be disabled in order to see the modified skins.
11. Select Control > Test Movie.

Note: The Stepper Assets/States folder also contains a StepTrack symbol, which is used as a spacer between the up and down skins if the total height of the NumericStepper instance is greater than the sum of the two arrow heights. This symbol linkage identifier is not available for modification through a skin property, but the library symbol can be modified provided the linkage identifier remains unchanged.

NumericStepper class

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > NumericStepper

ActionScript Class Name mx.controls.NumericStepper

The properties of the NumericStepper class let you set the following at runtime: the minimum and maximum values displayed in the stepper, the unit by which the stepper increases or decreases in response to a click, and the current value displayed in the stepper.

Setting a property of the NumericStepper class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

The NumericStepper component uses the Focus Manager to override the default Flash Player focus rectangle and draw a custom focus rectangle with rounded corners. For more information, see “Creating custom focus navigation” in Flash Help.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.NumericStepper.version);
```

Note: The code `trace(myNumericStepperInstance.version);` returns `undefined`.

Method summary for the `NumericStepper` class

There are no methods exclusive to the `NumericStepper` class.

Methods inherited from the `UIObject` class

The following table lists the methods the `NumericStepper` class inherits from the `UIObject` class. When calling these methods from the `NumericStepper` object, use the form `NumericStepper.methodName`.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it will be redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the `UIComponent` class

The following table lists the methods the `NumericStepper` class inherits from the `UIComponent` class. When calling these methods from the `NumericStepper` object, use the form `NumericStepper.methodName`.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the NumericStepper class

The following table lists properties of the NumericStepper class.

Property	Description
<code>NumericStepper.maximum</code>	A number indicating the maximum range value.
<code>NumericStepper.minimum</code>	A number indicating the minimum range value.
<code>NumericStepper.nextValue</code>	A number indicating the next sequential value. This property is read-only.
<code>NumericStepper.previousValue</code>	A number indicating the previous sequential value. This property is read-only.
<code>NumericStepper.stepSize</code>	A number indicating the unit of change for each click.
<code>NumericStepper.value</code>	A number indicating the current value of the stepper.

Properties inherited from the UIObject class

The following table lists the properties the NumericStepper class inherits from the UIObject class.

When calling these properties from the NumericStepper object, use the form

`NumericStepper.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `NumericStepper` class inherits from the `UIComponent` class. When calling these properties from the `NumericStepper` object, use the form `NumericStepper.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the `NumericStepper` class

The following table lists the event of the `NumericStepper` class.

Event	Description
<code>NumericStepper.change</code>	Triggered when the value of the stepper changes.

Events inherited from the `UIObject` class

The following table lists the events the `NumericStepper` class inherits from the `UIObject` class. When calling these events from the `NumericStepper` object, use the form `NumericStepper.eventName`.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the `UIComponent` class

The following table lists the events the `NumericStepper` class inherits from the `UIComponent` class. When calling these events from the `NumericStepper` object, use the form `NumericStepper.eventName`.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

NumericStepper.change

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(click){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.change = function(eventObject){  
    ...  
}  
stepperInstance.addEventListener("change", listenerObject)
```

Description

Event; broadcast to all registered listeners when the value of the stepper is changed.

The first usage example uses an `on()` handler and must be attached directly to a `NumericStepper` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the stepper `myStepper`, sends “`_level0.myStepper`” to the Output panel:

```
on(click){  
    trace(this);  
}
```

The second usage example uses a dispatcher/listener event model. A component instance (`stepperInstance`) dispatches an event (in this case, `change`) and the event is handled by a function, also called a *handler*, on a listener object (`listenerObject`) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (`eventObject`) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “`EventDispatcher` class” in Flash Help.

Example

This example, written on a frame of the Timeline, sends a message to the Output panel when a stepper called `myNumericStepper` is changed. The first line of code creates a listener object called `form`. The second line defines a function for the `change` event on the listener object. Inside the function is a `trace()` statement that uses the event object that is automatically passed to the function, in this example `eventObj`, to generate a message. The `target` property of an event object is the component that generated the event—in this example, `myNumericStepper`. The `NumericStepper.value` property is accessed from the event object's `target` property. The last line calls `EventDispatcher.addEventListener()` from `myNumericStepper` and passes it the `change` event and the `form` listener object as parameters.

```
form = new Object();
form.change = function(eventObj){
    // eventObj.target is the component that generated the change event,
    // i.e., the numeric stepper.
    trace("Value changed to " + eventObj.target.value);
}
myNumericStepper.addEventListener("change", form);
```

NumericStepper.maximum

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

stepperInstance.maximum

Description

Property; the maximum range value of the stepper. This property can contain a number of up to three decimal places. The default value is 10.

Example

The following example sets the maximum value of the stepper range to 20:

```
myStepper.maximum = 20;
```

See also

[NumericStepper.minimum](#)

NumericStepper.minimum

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

stepperInstance.minimum

Description

Property; the minimum range value of the stepper. This property can contain a number of up to three decimal places. The default value is 0.

Example

The following example sets the minimum value of the stepper range to 100:

```
myStepper.minimum = 100;
```

See also

[NumericStepper.maximum](#)

NumericStepper.nextValue

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

stepperInstance.nextValue

Description

Property (read-only); the next sequential value. This property can contain a number of up to three decimal places.

Example

The following example sets the `stepSize` property to 1 and the starting value to 4, which would make the value of `nextValue` 5:

```
myStepper.stepSize = 1;  
myStepper.value = 4;  
trace(myStepper.nextValue);
```

See also

[NumericStepper.previousValue](#)

NumericStepper.previousValue

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

stepperInstance.previousValue

Description

Property (read-only); the previous sequential value. This property can contain a number of up to three decimal places.

Example

The following example sets the `stepSize` property to 1 and the starting value to 4, which would make the value of `nextValue` 3:

```
myStepper.stepSize = 1;  
myStepper.value = 4;  
trace(myStepper.previousValue);
```

See also

[NumericStepper.nextValue](#)

NumericStepper.stepSize

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

stepperInstance.stepSize

Description

Property; the unit amount to change from the current value. The default value is 1. This value cannot be 0. This property can contain a number of up to three decimal places.

Example

The following example sets the current `value` property to 2 and the `stepSize` unit to 2. The value of `nextValue` is 4:

```
myStepper.value = 2;  
myStepper.stepSize = 2;  
trace(myStepper.nextValue);
```

NumericStepper.value

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

`stepperInstance.value`

Description

Property; the current value displayed in the text area of the stepper. The value is not assigned if it does not correspond to the stepper's range and step increment as defined in the `stepSize` property. This property can contain a number of up to three decimal places.

Example

The following example sets the current value of the stepper to 10 and sends the value to the Output panel:

```
myStepper.value = 10;
trace(myStepper.value);
```

ProgressBar component

The `ProgressBar` component displays the progress of loading content. 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 add a label to display the progress of the loading content.

By default, components are set to export in the first frame. This means that components are loaded into an application before the first frame is rendered. If you want to create a preloader for an application, you must deselect Export in First Frame in each component's Linkage Properties dialog box (available from the Library options menu). The progress bar, however, should be set to Export in First Frame, because it must appear first, while other content streams into Flash Player.

The `ProgressBar` component contains a left cap, a right cap, and a progress track. The caps are simply the ends of the progress bar, where the progress track visually ends. A live preview of each `ProgressBar` instance reflects changes made to parameters in the Property inspector or Component inspector during authoring. The following parameters are reflected in the live preview: conversion, direction, label, labelPlacement, mode, and source.

Using the ProgressBar component

A progress bar lets you display the progress of content as it loads. This is essential feedback for users as they interact with an application.

There are several modes in which to use the `ProgressBar` component; you set the mode with the mode parameter. The most commonly used modes are event mode and polled mode. These modes use the source parameter to specify a loading process that either emits progress and complete events (event mode), or exposes `getBytesLoaded()` and `getBytesTotal()` methods (polled mode). You can also use the `ProgressBar` component in manual mode by manually setting the maximum, minimum, and indeterminate properties along with calls to the `ProgressBar.setProgress()` method.

ProgressBar parameters

You can set the following authoring parameters for each `ProgressBar` instance in the Property inspector or in the Component inspector:

mode is the mode in which the progress bar operates. This value can be one of the following: `event`, `polled`, or `manual`. The default value is `event`.

source is a string to be converted into an object representing the instance name of the source.

direction indicates the direction toward which the progress bar fills. This value can be `right` or `left`; the default value is `right`.

label is the text indicating the loading progress. This parameter is a string in the format "%1 out of %2 loaded (%3%%)". In this string, %1 is a placeholder for the current bytes loaded, %2 is a placeholder for the total bytes loaded, and %3 is a placeholder for the percent of content loaded. The characters "%%" are a placeholder for the "%" character. If a value for %2 is unknown, it is replaced by two question marks (??). If a value is undefined, the label doesn't display.

labelPlacement indicates the position of the label in relation to the progress bar. This parameter can be one of the following values: `top`, `bottom`, `left`, `right`, `center`. The default value is `bottom`.

conversion is a number by which to divide the %1 and %2 values in the label string before they are displayed. The default value is 1.

You can write ActionScript to control these and additional options for the `ProgressBar` component using its properties, methods, and events. For more information, see [“ProgressBar class” on page 390](#).

Creating an application with the ProgressBar component

The following procedure explains how to add a `ProgressBar` component to an application while authoring. In this example, the progress bar is used in event mode. In event mode, the loading content must emit `progress` and `complete` events that the progress bar uses to display progress. (These events are emitted by the `Loader` component. For more information, see “`Loader` component” in Flash Help.)

To create an application with the `ProgressBar` component in event mode:

1. Drag a `ProgressBar` component from the Components panel to the Stage.
2. In the Property inspector, do the following:
 - Enter the instance name **pBar**.
 - Select `Event` for the `mode` parameter.
3. Drag a `Loader` component from the Components panel to the Stage.
4. In the Property inspector, enter the instance name **loader**.
5. Select the progress bar on the Stage and, in the Property inspector, enter **loader** for the `source` parameter.

6. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code, which loads a JPEG file into the Loader component:

```
loader.autoLoad = false;
loader.contentPath = "http://imagecache2.allposters.com/images/86/
017_PP0240.jpg";
pBar.source = loader;
// loading does not start until load() is invoked
loader.load();
```

In the following example, the progress bar is used in polled mode. In polled mode, the `ProgressBar` uses the `getBytesLoaded()` and `getBytesTotal()` methods of the source object to display its progress.

To create an application with the `ProgressBar` component in polled mode:

1. Drag a `ProgressBar` component from the Components panel to the Stage.
2. In the Property inspector, do the following:
 - Enter the instance name `pBar`.
 - Select `Polled` for the mode parameter.
 - Enter `loader` for the source parameter.
3. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code, which creates a `Sound` object called `loader` and calls `loadSound()` to load a sound into the `Sound` object:

```
var loader:Object = new Sound();
loader.loadSound("http://soundamerica.com/sounds/sound_fx/A-E/air.wav",
true);
```

In the following example, the progress bar is used in manual mode. In manual mode, you must set the `maximum`, `minimum`, and `indeterminate` properties in conjunction with the `setProgress()` method to display progress. You do not set the `source` property in manual mode.

To create an application with the `ProgressBar` component in manual mode:

1. Drag a `ProgressBar` component from the Components panel to the Stage.
2. In the Property inspector, do the following:
 - Enter the instance name `pBar`.
 - Select `Manual` for the mode parameter.
3. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code, which updates the progress bar manually on every file download by using calls to `setProgress()`:

```
for(var:Number i=1; i <= total; i++){
    // insert code to load file
    pBar.setProgress(i, total);
}
```

Customizing the ProgressBar component

You can transform a ProgressBar component horizontally while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use `UIObject.setSize()`.

The progress bar's left cap, right cap, and track graphic are set at a fixed size. When you resize a progress bar, its middle portion is resized to fit between the two caps. If a progress bar is too small, it may not render correctly.

Using styles with the ProgressBar component

You can set style properties to change the appearance of a progress bar instance. If the name of a style property ends in "Color", it is a color style property and behaves differently than noncolor style properties. For more information, see "Using styles to customize component color and text" in Flash Help.

A ProgressBar component supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>color</code>	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is "_sans".
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either "normal" or "italic". The default value is "normal".
<code>fontWeight</code>	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return "none".
<code>textDecoration</code>	Both	The text decoration: either "none" or "underline". The default value is "none".

Style	Theme	Description
barColor	Sample	The foreground color in denoting the percent complete. The default color is white. To set the bar color on a Halo-themed component, set the <code>themeColor</code> style property.
trackColor	Sample	The background color for the bar. The default value is 0x666666 (dark gray).

Using skins with the ProgressBar component

The ProgressBar component uses skins to represent the progress bar track, the completed bar, and an indeterminate bar. To skin the ProgressBar component while authoring, modify symbols in the Flash UI Components 2/Themes/MMDefault/ProgressBar Elements folder. For more information, see “About skinning components” in Flash Help.

The track and bar graphics are each made up of three skins corresponding to the left and right caps and the middle. The caps are used “as is,” and the middle is resized horizontally to fit the width of the ProgressBar instance.

The indeterminate bar is used when the ProgressBar instance’s `indeterminate` property is set to `true`. The skin is resized horizontally to fit the width of the progress bar.

A ProgressBar component supports the following skin properties:

Property	Description
<code>progTrackMiddleName</code>	The expandable middle of the track. The default value is <code>ProgTrackMiddle</code> .
<code>progTrackLeftName</code>	The fixed-size left cap. The default value is <code>ProgTrackLeft</code> .
<code>progTrackRightName</code>	The fixed-size right cap. The default value is <code>ProgTrackRight</code> .
<code>progBarMiddleName</code>	The expandable middle bar graphic. The default value is <code>ProgBarMiddle</code> .
<code>progBarLeftName</code>	The fixed-size left bar cap. The default value is <code>ProgBarLeft</code> .
<code>progBarRightName</code>	The fixed-size right bar cap. The default value is <code>ProgBarRight</code> .
<code>progIndBarName</code>	The indeterminate bar graphic. The default value is <code>ProgIndBar</code> .

To create movie clip symbols for ProgressBar skins:

1. Create a new FLA file.
2. Select File > Import > Open External Library, and select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in Flash Help.
3. In the theme’s Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the ProgressBar Assets folder to the library for your document.
4. Expand the ProgressBar Assets/Elements folder in the library of your document.
5. Open the symbols you want to customize for editing.
For example, open the ProgIndBar symbol.

6. Customize the symbol as desired.

For example, flip the track horizontally.

7. Repeat steps 5-6 for all symbols you want to customize.

8. Click the Back button to return to the main Timeline.

9. Drag a `ProgressBar` component to the Stage.

To view the skins modified in this example, use `ActionScript` to set the `indeterminate` property to `true`.

10. Select `Control > Test Movie`.

ProgressBar class

Inheritance `MovieClip` > [UIObject class](#) > `ProgressBar`

ActionScript Class Name `mx.controls.ProgressBar`

Setting a property of the `ProgressBar` class with `ActionScript` overrides the parameter of the same name set in the Property inspector or Component inspector.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.ProgressBar.version);
```

Note: The code `trace(myProgressBarInstance.version);` returns `undefined`.

Method summary for the ProgressBar class

The following table lists the method of the `ProgressBar` class.

Method	Description
ProgressBar.setProgress()	Sets the state of the progress bar to reflect the amount of progress made when the progress bar is in manual mode

Methods inherited from the UIObject class

The following table lists the methods the `ProgressBar` class inherits from the `UIObject` class.

When calling these methods from the `ProgressBar` object, use the form

`ProgressBar.methodName`.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.

Method	Description
<code>UIObject.invalidate()</code>	Marks the object so it will be redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Property summary for the `ProgressBar` class

The following table lists properties of the `ProgressBar` class.

Property	Description
<code>ProgressBar.conversion</code>	A number used to convert the current bytes loaded value and the total bytes loaded values.
<code>ProgressBar.direction</code>	The direction in which the progress bar fills.
<code>ProgressBar.indeterminate</code>	Indicates whether the size of the loading source is unknown.
<code>ProgressBar.label</code>	The text that accompanies the progress bar.
<code>ProgressBar.labelPlacement</code>	The location of the label in relation to the progress bar.
<code>ProgressBar.maximum</code>	The maximum value of the progress bar in manual mode.
<code>ProgressBar.minimum</code>	The minimum value of the progress bar in manual mode.
<code>ProgressBar.mode</code>	The mode in which the progress bar loads content.
<code>ProgressBar.percentComplete</code>	Read-only; a number indicating the percent loaded.
<code>ProgressBar.source</code>	The content to load.
<code>ProgressBar.value</code>	Read-only; indicates the amount of progress that has been made.

Properties inherited from the `UIObject` class

The following table lists the properties the `ProgressBar` class inherits from the `UIObject` class.

When calling these properties from the `ProgressBar` object, use the form

`ProgressBar.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.

Property	Description
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Event summary for the `ProgressBar` class

The following table lists events of the `ProgressBar` class.

Event	Description
<code>ProgressBar.complete</code>	Triggered when loading is complete.
<code>ProgressBar.progress</code>	Triggered as content loads in manual or polled mode.

Events inherited from the `UIObject` class

The following table lists the events the `ProgressBar` class inherits from the `UIObject` class. When calling these events from the `ProgressBar` object, use the form `ProgressBar.eventName`.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

ProgressBar.complete

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(complete){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.complete = function(eventObject){  
    ...  
}  
pBar.addEventListener("complete", listenerObject)
```

Event object

In addition to the standard event object properties, there are two additional properties defined for the `ProgressBar.complete` event: `current` (the loaded value equals total), and `total` (the total value).

Description

Event; broadcast to all registered listeners when the loading progress has completed.

The first usage example uses an `on()` handler and must be attached directly to a `ProgressBar` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the instance `pBar`, sends “_level0.pBar” to the Output panel:

```
on(complete){  
    trace(this);  
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*pBar*) dispatches an event (in this case, `complete`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

This example creates a form listener object with a complete callback function that sends a message to the Output panel with the value of the pBar instance:

```
form.complete = function(eventObj){
    // eventObj.target is the component that generated the complete event,
    // i.e., the progress bar.
    trace("Current ProgressBar value = " + eventObj.target.value);
}
pBar.addEventListener("complete", form);
```

See also

`EventDispatcher.addEventListener()` in Flash Help

ProgressBar.conversion

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

pBarInstance.conversion

Description

Property; a number that sets a conversion value for the incoming values. It divides the current and total values, floors them, and displays the converted value in the `label` property. The default value is 1.

Note: The floor is the closest integer value that is less than or equal to the specified value. For example, the number 4.6 becomes 4.

Example

The following code displays the value of the loading progress in kilobytes:

```
pBar.conversion = 1024;
```

ProgressBar.direction

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

pBarInstance.direction

Description

Property; indicates the fill direction for the progress bar. The default value is "right".

Example

The following code makes the progress bar fill from right to left:

```
pBar.direction = "left";
```

ProgressBar.indeterminate

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
pBarInstance.indeterminate
```

Description

Property; a Boolean value that indicates whether the progress bar has a striped fill and a loading source of unknown size (`true`), or a solid fill and a loading source of a known size (`false`). For example, you might use this property if you are loading a large data set into a SWF file and do not know the size of the data you are loading.

Example

The following code creates a determinate progress bar with a solid fill that moves from left to right. Drag an instance of the `ProgressBar` component onto the Stage, and enter the instance name `my_pb` in the Property inspector. Drag an instance of the `Loader` component onto the Stage, and enter the instance name `my_ldr` in the Property inspector. Add the following code to Frame 1 of the Timeline:

```
var my_pb:mx.controls.ProgressBar;
var my_ldr:mx.controls.Loader;

var pbListener:Object = new Object();
pbListener.complete = function(evt:Object) {
    evt.target._visible = false;
};
my_pb.addEventListener("complete", pbListener);
my_pb.mode = "polled";
my_pb.indeterminate = true;
my_pb.source = my_ldr;

my_ldr.autoLoad = false;
my_ldr.scaleContent = false;
my_ldr.load("http://www.macromedia.com/software/flex/images/
flex_presentation_eyes.jpg");
```

ProgressBar.label

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

pBarInstance.label

Description

Property; text that indicates the loading progress. This property is a string in the format "%1 out of %2 loaded (%3%)". In this string, %1 is a placeholder for the current bytes loaded, %2 is a placeholder for the total bytes loaded, and %3 is a placeholder for the percentage of content loaded. (The characters %% allow Flash to display a single % character.) If a value for %2 is unknown, it is replaced by ??. If a value is undefined, the label doesn't display. The default value is "LOADING %3%".

Example

The following code lets your application display progress bar text that reads "3 files loaded," "4 files loaded," and so on as the files load:

```
pBar.label = "%1 files loaded";
```

See also

[ProgressBar.labelPlacement](#)

ProgressBar.labelPlacement

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

pBarInstance.labelPlacement

Description

Property; sets the placement of the label in relation to the progress bar. The possible values are "left", "right", "top", "bottom", and "center".

Example

The following code specifies that the text label appears above the progress bar:

```
pBar.label = "%1 out of %2 loaded (%3%)";  
pBar.labelPlacement = "top";
```

See also

[ProgressBar.label](#)

ProgressBar.maximum

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

pBarInstance.maximum

Description

Property; the largest value for the progress bar when the [ProgressBar.mode](#) property is set to "manual".

Example

The following code sets the `maximum` property to the total frames of a Flash application that's loading:

```
pBar.maximum = _totalframes;
```

See also

[ProgressBar.minimum](#), [ProgressBar.mode](#)

ProgressBar.minimum

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

pBarInstance.minimum

Description

Property; the smallest value for the progress bar when the [ProgressBar.mode](#) property is set to "manual".

Example

The following code sets the minimum value for the progress bar:

```
pBar.minimum = 0;
```

See also

[ProgressBar.maximum](#), [ProgressBar.mode](#)

ProgressBar.mode

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

pBarInstance.mode

Description

Property; the mode in which the progress bar loads content. This value can be "event", "polled", or "manual".

Event mode and polled mode are the most common modes. In event mode, the `source` property specifies loading content that emits `progress` and `complete` events; you should use a `Loader` object in this mode. In polled mode, the `source` property specifies loading content (such as a `MovieClip` object) that exposes `getBytesLoaded()` and `getBytesTotal()` methods. Any object that exposes these methods can be used as a source in polled mode (including a custom object or the root `Timeline`).

You can also use the `ProgressBar` component in manual mode by manually setting the `maximum`, `minimum`, and `indeterminate` properties and making calls to the `ProgressBar.setProgress()` method.

Example

The following code sets the progress bar to event mode. Drag an instance of the `ProgressBar` component onto the Stage, and enter the instance name `my_pb` in the Property inspector. Drag an instance of the `Loader` component onto the Stage, and enter an instance name `my_ldr` in the Property inspector. Add the following code to Frame 1 of the Timeline:

```
var my_pb:mx.controls.ProgressBar;
var my_ldr:mx.controls.Loader;

var pbListener:Object = new Object();
pbListener.complete = function(evt:Object) {
    evt.target._visible = false;
};

my_pb.addEventListener("complete", pbListener);
my_pb.mode = "polled";
my_pb.indeterminate = true;
my_pb.source = my_ldr;

my_ldr.autoLoad = false;
my_ldr.scaleContent = false;
my_ldr.load("http://www.macromedia.com/software/flex/images/
flex_presentation_eyes.jpg");
```

ProgressBar.percentComplete

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

pBarInstance.percentComplete

Description

Property (read-only); tells what percentage of the content has been loaded. This value is floored. (The floor is the closest integer value that is less than or equal to the specified value. For example, the number 7.8 becomes 7.) The following formula is used to calculate the percentage:

$$100 * (\text{value} - \text{minimum}) / (\text{maximum} - \text{minimum})$$

Example

The following code sends the value of the `percentComplete` property to the Output panel:

```
trace("percent complete = " + pBar.percentComplete);
```

ProgressBar.progress

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(progress){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.progress = function(eventObject){  
    ...  
}  
pBarInstance.addEventListener("progress", listenerObject)
```

Event object

In addition to the standard event object properties, there are two additional properties defined for the `ProgressBar.progress` event: `current` (the loaded value equals total), and `total` (the total value).

Description

Event; broadcast to all registered listeners whenever the value of a progress bar changes. This event is broadcast only when `ProgressBar.mode` is set to "manual" or "polled".

The first usage example uses an `on()` handler and must be attached directly to a `ProgressBar` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the instance `myPBar`, sends “_level0.myPBar” to the Output panel:

```
on(progress){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*pBarInstance*) dispatches an event (in this case, `progress`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

This example creates a listener object, `form`, and defines a `progress` event handler on it. The `form` listener is registered to the `pBar` instance in the last line of code. When the `progress` event is triggered, `pBar` broadcasts the event to the `form` listener, which calls the `progress` callback function.

```
var form:Object = new Object();
form.progress = function(eventObj){
    // eventObj.target is the component that generated the progress event,
    // i.e., the progress bar.
    trace("Current progress value = " + eventObj.target.value);
}
pBar.addEventListener("progress", form);
```

See also

`EventDispatcher.addEventListener()` in Flash Help

ProgressBar.setProgress()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
pBarInstance.setProgress(completed, total)
```

Parameters

completed A number indicating the amount of progress that has been made. You can use the `ProgressBar.label` and `ProgressBar.conversion` properties to display the number in percentage form or any units you choose, depending on the source of the progress bar.

total A number indicating the total progress that must be made to reach 100%.

Returns

A number indicating the amount of progress that has been made.

Description

Method; sets the state of the progress bar to reflect the amount of progress made when the `ProgressBar.mode` property is set to "manual". You can call this method to make the bar reflect the state of a process other than loading. For example, you might want to explicitly set the progress bar to zero progress.

The *completed* parameter is assigned to the `value` property and the *total* parameter is assigned to the `maximum` property. The `minimum` property is not altered.

Example

The following code calls `setProgress()` according to the progress of a Flash application's Timeline:

```
pBar.setProgress(_currentFrame, _totalFrames);
```

ProgressBar.source

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
pBarInstance.source
```

Description

Property; a reference to the instance to be loaded whose loading process will be displayed. The loading content should emit a `progress` event from which the current and total values are retrieved. This property is used only when `ProgressBar.mode` is set to "event" or "polled". The default value is `undefined`.

The `ProgressBar` component can be used with content within an application, including `_root`.

Example

This example sets the `pBar` instance to display the loading progress of a Loader component with the instance name `loader`:

```
pBar.source = loader;
```

See also

[ProgressBar.mode](#)

ProgressBar.value

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

pBarInstance.value

Description

Property (read-only); indicates the amount of progress that has been made. This property is a number between the value of [ProgressBar.minimum](#) and [ProgressBar.maximum](#). The default value is 0.

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 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 control it:

Key	Description
Up Arrow/Right Arrow	The selection moves to the previous radio button within the radio button group.
Down Arrow/Left 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 “Creating custom focus navigation” in Flash Help or [“FocusManager class” on page 231](#).

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 will appear selected at runtime. For more information, see [“RadioButton parameters” on page 403](#).

When you add the `RadioButton` component to an application, you can use the Accessibility panel to make it accessible to screen readers. First, you must add the following line of code to enable accessibility:

```
mx.accessibility.RadioButtonAccImpl.enableAccessibility();
```

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

Using the `RadioButton` component

A radio button is a fundamental part of any form or web application. 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.

RadioButton parameters

You can set the following authoring parameters for each `RadioButton` component instance in the Property inspector or in the Component inspector:

label sets the value of the text on the button; the default value is `Radio Button`.

data is the value associated with the radio button. There is no default value.

groupName is the group name of the radio button. The default value is `radioGroup`.

selected sets the initial value of the radio button to selected (`true`) or unselected (`false`). A selected radio button displays a dot inside it. Only one radio button in a group can have a selected value of `true`. If more than one radio button in a group is set to `true`, the radio button that is instantiated last is selected. The default value is `false`.

labelPlacement orients the label text on the button. This parameter can be one of four values: `left`, `right`, `top`, or `bottom`; the default value is `right`. For more information, see [`RadioButton.labelPlacement`](#).

You can write ActionScript to set additional options for `RadioButton` instances using the methods, properties, and events of the `RadioButton` class. For more information, see [“RadioButton class” on page 407](#).

Creating an application with the `RadioButton` component

The following procedure explains how to add `RadioButton` components to an application while authoring. In this example, the radio buttons are used to present the yes-or-no question “Are you a Flashist?”. The data from the radio group is displayed in a `TextArea` component with the instance name `theVerdict`.

To create an application with the `RadioButton` component:

1. Drag two `RadioButton` components from the Components panel to the Stage.
2. Select one of the radio buttons. In the Component inspector, do the following:
 - Enter **Yes** for the label parameter.
 - Enter **Flashist** for the data parameter.
3. Select the other radio button. In the Component inspector, do the following:
 - Enter **No** for the label parameter.
 - Enter **Anti-Flashist** for the data parameter.
4. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code:

```
flashistListener = new Object();
flashistListener.click = function (evt){
    theVerdict.text = evt.target.selection.data
}
radioGroup.addEventListener("click", flashistListener);
```

The last line of code adds a `click` event handler to the `radioGroup` radio button group. The handler sets the `text` property of `theVerdict` (a `TextArea` instance) to the value of the `data` property of the selected radio button in the `radioGroup` radio button group. For more information, see [RadioButton.click](#).

Customizing the `RadioButton` component

You can transform a `RadioButton` component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the `Modify > Transform` commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)).

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.

Using styles with the `RadioButton` component

You can set style properties to change the appearance of a `RadioButton`. If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in Flash Help.

A `RadioButton` component uses the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>color</code>	Both	The text color. The default value is <code>Ox0B333C</code> for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is <code>Ox848384</code> (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either "normal" or "italic". The default value is "normal".
<code>fontWeight</code>	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return "none".
<code>textDecoration</code>	Both	The text decoration: either "none" or "underline". The default value is "none".
<code>symbolBackgroundColor</code>	Sample	The background color of the radio button. The default value is <code>OxFFFFFFFF</code> (white).
<code>symbolBackgroundDisabledColor</code>	Sample	The background color of the radio button when disabled. The default value is <code>OxEFEEEEF</code> (light gray).
<code>symbolBackgroundPressedColor</code>	Sample	The background color of the radio button when pressed. The default value is <code>OxFFFFFFFF</code> (white).
<code>symbolColor</code>	Sample	The color of the dot in the radio button. The default value is <code>Ox000000</code> (black).
<code>symbolDisabledColor</code>	Sample	The color of the dot in the radio button when the component is disabled. The default value is <code>Ox848384</code> (dark gray).

Using skins with the RadioButton component

You can skin the RadioButton component while authoring by modifying the component's symbols in the library. The skins for the RadioButton component are located in the following folder in the library of HaloTheme.fla or SampleTheme.fla: Flash UI Components 2/Themes/MMDefault/RadioButton Assets/States. See “About skinning components” in Flash Help.

If a radio button is enabled and unselected, it displays its rollover state when a user moves the pointer over it. When a user clicks an unselected radio button, the radio button receives input focus and displays its false pressed state. When a user releases the mouse, the radio button displays its true state and the previously selected radio button in the group returns to its false state. If a user moves the pointer off a radio button while pressing the mouse, the radio button's appearance returns to its false state and it retains input focus.

If a radio button or radio button group is disabled, it displays its disabled state, regardless of user interaction.

A RadioButton component uses the following skin properties:

Name	Description
falseUpIcon	The unselected state. The default value is <code>RadioFalseUp</code> .
falseDownIcon	The pressed-unselected state. The default value is <code>RadioFalseDown</code> .
falseOverIcon	The over-unselected state. The default value is <code>RadioFalseOver</code> .
falseDisabledIcon	The disabled-unselected state. The default value is <code>RadioFalseDisabled</code> .
trueUpIcon	The selected state. The default value is <code>RadioTrueUp</code> .
trueDisabledIcon	The disabled-selected state. The default value is <code>RadioTrueDisabled</code> .

Each of these skins correspond to the icon indicating the RadioButton state. The RadioButton does not have a border or background.

To create movie clip symbols for RadioButton skins:

1. Create a new FLA file.
2. Select File > Import > Open External Library, and select the HaloTheme.fla file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in Flash Help.
3. In the theme's Library panel, expand the Flash UI Components 2/Themes/MMDefault folder and drag the RadioButton Assets folder to the library for your document.
4. Expand the RadioButton Assets/States folder in the library of your document.
5. Open the symbols you want to customize for editing.
For example, open the RadioFalseDisabled symbol.
6. Customize the symbol as desired.
For example, change the inner white circle to a light gray.

7. Repeat steps 5-6 for all symbols you want to customize.
For example, repeat the color change for the inner circle of the RadioTrueDisabled symbol.
8. Click the Back button to return to the main Timeline.
9. Drag a RadioButton component to the Stage.
For this example, drag two instances to show the two new skin symbols.
10. Set the RadioButton instance properties as desired.
For this example, set one RadioButton to selected, and use ActionScript to set both RadioButton instances to disabled.
11. Select Control > Test Movie.

RadioButton class

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > [SimpleButton class](#) > [Button component](#) > RadioButton

ActionScript Package Name mx.controls.RadioButton

The properties of the RadioButton class allow you at runtime to create a text label and position it in relation to the radio button. You can also assign data values to radio buttons, assign them to groups, and select them based on data value or instance name.

Setting a property of the RadioButton class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

The RadioButton component uses the Focus Manager to override the default Flash Player focus rectangle and draw a custom focus rectangle with rounded corners. For information about creating focus navigation, see “Creating custom focus navigation” in Flash Help.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.RadioButton.version);
```

Note: The code `trace(myRadioButtonInstance.version);` returns `undefined`.

Method summary for the RadioButton class

There are no methods exclusive to the RadioButton class.

Methods inherited from the UIObject class

The following table lists the methods the RadioButton class inherits from the UIObject class. When calling these methods from the RadioButton object, use the form *RadioButtonInstance.methodName*.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.

Method	Description
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it will be redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the `RadioButton` class inherits from the `UIComponent` class. When calling these methods from the `RadioButton` object, use the form *RadioButtonInstance.methodName*.

Method	Description
UIComponent.getFocus()	Returns a reference to the object that has focus.
UIComponent.setFocus()	Sets focus to the component instance.

Property summary for the RadioButton class

The following table lists properties of the `RadioButton` class.

Property	Description
RadioButton.data	The value associated with a radio button instance.
RadioButton.groupName	The group name for a radio button group instance or radio button instance.
RadioButton.label	The text that appears next to a radio button.
RadioButton.labelPlacement	The orientation of the label text in relation to a radio button or radio button group.
RadioButton.selected	Selects the radio button, and deselects the previously selected radio button. This property can be used with a <code>RadioButton</code> instance or a <code>RadioButtonGroup</code> instance.
RadioButton.selectedData	Selects the radio button with the specified data value in a radio button group.
RadioButton.selection	A reference to the currently selected radio button in a radio button group. This property can be used with a <code>RadioButton</code> instance or a <code>RadioButtonGroup</code> instance.

Properties inherited from the `UIObject` class

The following table lists the properties the `RadioButton` class inherits from the `UIObject` class. When accessing these properties from the `RadioButton` object, use the form *RadioButtonInstance.propertyName*.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `RadioButton` class inherits from the `UIComponent` class. When accessing these properties from the `RadioButton` object, use the form *RadioButtonInstance.propertyName*.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Properties inherited from the SimpleButton class

The following table lists the properties `RadioButton` class inherits from the `SimpleButton` class. When accessing these properties from the `RadioButton` object, use the form `RadioButtonInstance.propertyName`.

Property	Description
<code>SimpleButton.emphasized</code>	Indicates whether a button has the appearance of a default push button.
<code>SimpleButton.emphasizedStyleDeclaration</code>	The style declaration when the <code>emphasized</code> property is set to <code>true</code> .
<code>SimpleButton.selected</code>	A Boolean value indicating whether the button is selected (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .
<code>SimpleButton.toggle</code>	A Boolean value indicating whether the button behaves as a toggle switch (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .

Properties inherited from the Button class

The following table lists the properties the `RadioButton` class inherits from the `Button` class. When accessing these properties from the `RadioButton` object, use the form `RadioButtonInstance.propertyName`.

Property	Description
<code>Button.icon</code>	Specifies an icon for a button instance.
<code>Button.label</code>	Specifies the text that appears in a button.
<code>Button.labelPlacement</code>	Specifies the orientation of the label text in relation to an icon.

Event summary for the RadioButton class

The following table lists the event of the `RadioButton` class.

Event	Description
<code>RadioButton.click</code>	Triggered when the mouse is clicked over a radio button or radio button group.

Events inherited from the UIObject class

The following table lists the events the `RadioButton` class inherits from the `UIObject` class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.

Event	Description
UIObject.resize	Broadcast when an object has been resized.
UIObject.reveal	Broadcast when an object's state changes from invisible to visible.
UIObject.unload	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the RadioButton class inherits from the UIComponent class.

Event	Description
UIComponent.focusIn	Broadcast when an object receives focus.
UIComponent.focusOut	Broadcast when an object loses focus.
UIComponent.keyDown	Broadcast when a key is pressed.
UIComponent.keyUp	Broadcast when a key is released.

Events inherited from the SimpleButton class

The following table lists the event the RadioButton class inherits from the SimpleButton class.

Event	Description
SimpleButton.click	Broadcast when the mouse is clicked (released) over a button or if the button has focus and the Spacebar is pressed.

RadioButton.click

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(click){
    ...
}
```

Usage 2:

```
listenerObject = new Object();
listenerObject.click = function(eventObject){
    ...
}
radioButtonGroup.addEventListener("click", listenerObject)
```

Description

Event; broadcast to all registered listeners when the mouse is clicked (pressed and released) over the radio button or if the radio button is selected by means of the arrow keys. The event is also broadcast if the Spacebar or arrow keys are pressed when a radio button group has focus, but none of the radio buttons in the group are selected.

The first usage example uses an `on()` handler and must be attached directly to a `RadioButton` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the radio button `myRadioButton`, sends “_level0.myRadioButton” to the Output panel:

```
on(click){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*radioButtonInstance*) dispatches an event (in this case, `click`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

This example, written on a frame of the Timeline, sends a message to the Output panel when a radio button in `radioGroup` is clicked. The first line of code creates a listener object called `form`. The second line defines a function for the `click` event on the listener object. Inside the function is a `trace()` statement that uses the event object (`eventObj`) that is automatically passed to the function to generate a message. The `target` property of an event object is the component that generated the event. You can access instance properties from the `target` property (in this example, the `RadioButton.selection` property is accessed). The last line calls `EventDispatcher.addEventListener()` from `radioGroup` and passes it the `click` event and the `form` listener object as parameters.

```
form = new Object();
form.click = function(eventObj){
    trace("The selected radio instance is " + eventObj.target.selection);
}
radioGroup.addEventListener("click", form);
```

The following code also sends a message to the Output panel when `radioButtonInstance` is clicked. The `on()` handler must be attached directly to `radioButtonInstance`.

```
on(click){
    trace("radio button component was clicked");
}
```

RadioButton.data

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

radioButtonInstance.data

Description

Property; specifies the data to associate with a RadioButton instance. Setting this property overrides the data parameter value set during authoring. The `data` property can be of any data type.

Example

The following example assigns the data value "#FF00FF" to the `radioOne` radio button instance:

```
radioOne.data = "#FF00FF";
```

RadioButton.groupName

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

radioButtonInstance.groupName

radioButtonGroup.groupName

Description

Property; sets the group name for a radio button instance or group. You can use this property to get or set a group name for a radio button instance or for a radio button group. Calling this method overrides the `groupName` parameter value set during authoring. The default value is "radioGroup".

Example

The following example sets the group name of a radio button instance to `colorChoice` and then changes the group name to `sizeChoice`. To test this example, place a radio button on the Stage, name the instance name `myRadioButton`, and enter the following code on Frame 1:

```
myRadioButton.groupName = "colorChoice";  
trace(myRadioButton.groupName);  
colorChoice.groupName = "sizeChoice";  
trace(colorChoice.groupName);
```

RadioButton.label

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
radioButtonInstance.label
```

Description

Property; specifies the text label for the radio button. By default, the label appears to the right of the radio button. Calling this method overrides the label parameter specified during authoring. If the label text is too long to fit within the bounding box of the component, the text is clipped.

Example

The following example sets the `label` property of the instance `radioButton`:

```
radioButton.label = "Remove from list";
```

RadioButton.labelPlacement

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
radioButtonInstance.labelPlacement
```

```
radioButtonGroup.labelPlacement
```

Description

Property; a string that indicates the position of the label in relation to a radio button. You can set this property for an individual instance or for a radio button group. If you set the property for a group, the label is placed in the appropriate position for each radio button in the group.

The following are the four possible values:

- "right" The radio button is pinned to the upper left corner of the bounding area. The label is placed to the right of the radio button.
- "left" The radio button is pinned to the upper right corner of the bounding area. The label is placed to the left of the radio button.
- "bottom" The label is placed below the radio button. The radio button and label grouping are centered horizontally and vertically. If the bounding box of the radio button isn't large enough, the label is clipped.

- "top" The label is placed above the radio button. The radio button and label grouping are centered horizontally and vertically. If the bounding box of the radio button isn't large enough, the label is clipped.

Example

The following code places the label to the left of each radio button in `radioGroup`:

```
radioGroup.labelPlacement = "left";
```

RadioButton.selected

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
radioButtonInstance.selected
```

```
radioButtonGroup.selected
```

Description

Property; a Boolean value that sets the state of the radio button to selected (`true`) and deselects the previously selected radio button, or sets the radio button to deselected (`false`).

Example

The first line of code sets the `mcButton` instance to `true`. The second line of code returns the value of the `selected` property.

```
mcButton.selected = true;  
trace(mcButton.selected);
```

RadioButton.selectedData

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
radioButtonGroup.selectedData
```

Description

Property; selects the radio button with the specified data value and deselects the previously selected radio button. If the `data` property is not specified for a selected instance, the `label` value of the selected instance is selected and returned. The `selectedData` property can be of any data type.

Example

The following example selects the radio button with the value "#FF00FF" from the radio group `colorGroup` and sends the value to the Output panel:

```
colorGroup.selectedData = "#FF00FF";  
trace(colorGroup.selectedData);
```

RadioButton.selection

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
radioButtonInstance.selection  
radioButtonGroup.selection
```

Description

Property; behaves differently depending on whether you get or set the property. If you get the property, it returns the object reference of the currently selected radio button in a radio button group. If you set the property, it selects the specified radio button (passed as an object reference) in a radio button group and deselects the previously selected radio button.

Example

The following example selects the radio button with the instance name `color1` and sends its instance name to the Output panel:

```
colorGroup.selection = color1;  
trace(colorGroup.selection._name)
```

RoundIconButton component

The `RoundIconButton` component lets you create a simple, round push button with a custom icon or a plus (+) sign or a minus (-) sign.



RoundIconButton component with a custom icon

Using the RoundIconButton component

Round icon buttons look pressable, and each one has an icon on its face. Round icon buttons typically perform an action when they are clicked by the user.

The default icon for the `RoundIconButton` component is a plus (+) sign or a minus (-) sign that you can use in your application to perform tasks that, for example, allow users to add items to or remove items from a list.

RoundIconButton parameters

You can set the following parameters for each RoundIconButton component instance:

Built-In Icon has three possible values: `icon_add`, an icon with a plus (+) sign; `icon_remove`, an icon with a minus (-) sign; or `Custom`.

Custom Icon is the symbol ID of the image file to be used for a custom icon; for example, `logo.jpg`.

You can set additional options and functionality for instances of this component by using its methods.

About RoundIconButton states

The RoundIconButton component has the following states: `active`, `over`, `press`, `default`, `default over`, `disabled`. The component in all states has a green border to indicate its status to the user, except for the `active` and `disabled` states. The `disabled` button is dimmed and is unavailable to the user.

Method summary for the MRoundIconButton component

The following table summarizes the methods for the MRoundIconButton class:

Method	Description
<code>MRoundIconButton.setEnabled()</code>	Returns <code>true</code> if enabled and <code>false</code> if disabled.
<code>MRoundIconButton.getIcon()</code>	Returns an instance of the icon inside the button.
<code>MRoundIconButton.setChangeHandler()</code>	Assigns a function that is called every time the push button is released (toggle state is <code>false</code>).
<code>MRoundIconButton.setClickHandler()</code>	Specifies a function that is called when the component is clicked.
<code>MRoundIconButton.setEnabled()</code>	Enables or disables the push button.
<code>MRoundIconButton.setIcon()</code>	Sets the icon that is displayed in the push button.

MRoundIconButton.setEnabled()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myRoundIconButton.setEnabled()
```

Parameters

None.

Returns

A Boolean value.

Description

Method; indicates whether the `RoundIconButton` instance is enabled (`true`) or disabled (`false`).

Example

The following example returns the enabled state of the `RoundIconButton1` instance to the Output panel:

```
trace(RoundIconButton1.getEnabled());
```

MRoundIconButton.getIcon()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myRoundIconButton.getIcon()
```

Parameters

None.

Returns

A reference to the movie clip in `RoundIconButton`.

Description

Method; returns an instance of the icon that is displayed in the icon button.

Example

The following example retrieves a reference to the movie clip inside the `RoundIconButton1` object, stores the reference in a variable, and sets the `rotation` property to 45:

```
var icon = RoundIconButton1.getIcon();  
icon._rotation=45;
```

MRoundIconButton.setChangeHandler()**Availability**

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myRoundIconButton.setChangeHandler(callback)
```

Parameters

callback The string name of the function that is called. The function that it calls should reside on the same Timeline as the RoundIconButton component.

Returns

Nothing.

Description

Method; specifies a change handler function to call when the icon button is released. The function always accepts, as a parameter, the instance of the component that has changed. Calling this method overrides the Change Handler parameter value specified in the Property inspector.

Example

The following example sets a change handler function for the RoundIconButton1 object:

```
RoundIconButton1.setChangeHandler("On");
```

MRoundIconButton.setClickHandler()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myRoundIconButton.setClickHandler(function)
```

Parameters

function The string name of the function that is called. The function that it calls should reside on the same Timeline as the RoundIconButton component.

Returns

Nothing.

Description

Method; specifies a click handler function to call when the user clicks the icon button. The function always accepts, as a parameter, the instance of the component that has changed.

Example

The following example sets a click handler for the RoundIconButton1 object:

```
RoundIconButton1.setClickHandler("On");
```

MRoundIconButton.setEnabled()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myRoundIconButton.setEnabled(state)
```

Parameters

state A Boolean value: `true` enables the icon button; `false` disables it. The default value is `true`.

Returns

Nothing.

Description

Method; specifies whether the icon button is enabled (`true`) or disabled (`false`). If an icon button is disabled, it does not accept mouse or keyboard interaction from the user.

Example

The following example disables the button:

```
myRoundIconButton.setEnabled(false);
```

MRoundIconButton.setIcon()

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myRoundIconButton.setIcon(linkage)
```

Parameters

linkage A string listing the linkage name or instance reference of the movie clip target.

Returns

Nothing.

Description

Method; sets the icon for the icon button. If the icon movie clip is composed of frames labeled “_up,” “_over,” “_down,” and “disabled,” the content of each frame is displayed with the corresponding button state. Calling this method overrides the Icon parameter value set in authoring.

Example

The following example applies the icon movie clip that has the linkage named `foo`:

```
RoundIconButton1.setIcon("foo");
```

ScrollPane component

The ScrollPane component displays movie clips, JPEG files, and SWF files in a scrollable area. By using a scroll pane, you can limit the amount of screen area occupied by these media types. The scroll pane can display content that is loaded from a local disk or from the Internet. You can set this content while authoring and at runtime by using ActionScript.

Once the scroll pane has focus, if its content has valid tab stops, those markers receive focus. After the last tab stop in the content, focus shifts to the next component. The vertical and horizontal scroll bars in the scroll pane never receive focus.

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

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

For more information about controlling focus, see “Creating custom focus navigation” in Flash Help or “[FocusManager class](#)” on page 231.

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

Using the ScrollPane component

You can use a scroll pane to display any 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 scroll pane.

You can set up a scroll pane to allow users to drag the content within the pane by setting the `scrollDrag` parameter to `true`; a pointing hand appears on the content. Unlike most other components, events are broadcast when the mouse button is pressed and continue broadcasting until the button is released. If the contents of a scroll pane have valid tab stops, you must set `scrollDrag` to `false`; otherwise each mouse interaction with the contents will invoke scroll dragging.

ScrollPane parameters

You can set the following authoring parameters for each `ScrollPane` instance in the Property inspector or in the Component inspector:

contentPath indicates the content to load into the scroll pane. This value can be a relative path to a local SWF or JPEG file, or a relative or absolute path to a file on the Internet. It can also be the linkage identifier of a movie clip symbol in the library that is set to Export for ActionScript.

hLineScrollSize indicates the number of units a horizontal scroll bar moves each time an arrow button is clicked. The default value is 5.

hPageScrollSize indicates the number of units a horizontal scroll bar moves each time the track is clicked. The default value is 20.

hScrollPolicy displays the horizontal scroll bars. The value can be `on`, `off`, or `auto`. The default value is `auto`.

scrollDrag is a Boolean value that determines whether scrolling occurs (`true`) or not (`false`) when a user drags on the content within the scroll pane. The default value is `false`.

vLineScrollSize indicates the number of units a vertical scroll bar moves each time a scroll arrow is clicked. The default value is 5.

vPageScrollSize indicates the number of units a vertical scroll bar moves each time the scroll bar track is clicked. The default value is 20.

vScrollPolicy displays the vertical scroll bars. The value can be `on`, `off`, or `auto`. The default value is `auto`.

You can write ActionScript to control these and additional options for a `ScrollPane` component using its properties, methods, and events. For more information, see [“UI ScrollBar class” on page 561](#).

Creating 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 scroll pane loads a SWF file that contains a logo.

To create an application with the ScrollPane component:

1. Drag a `ScrollPane` component from the Components panel to the Stage.
2. In the Property inspector, enter the instance name `myScrollPane`.
3. In the Property inspector, enter `logo.swf` for the `contentPath` parameter.

4. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code:

```
scrollListener = new Object();
scrollListener.scroll = function (evt){
    txtPosition.text = myScrollPane.vPosition;
}
myScrollPane.addEventListener("scroll", scrollListener);
completeListener = new Object;
completeListener.complete = function() {
    trace("logo.swf has completed loading.");
}
myScrollPane.addEventListener("complete", completeListener);
```

The first block of code is a `scroll` event handler on the `myScrollPane` instance that displays the value of the `vPosition` property in a `TextField` instance called `txtPosition`. The second block of code creates an event handler for the `complete` event that sends a message to the Output panel.

Customizing the ScrollPane component

You can transform a `ScrollPane` component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the `Modify > Transform` commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)) or any applicable properties and methods of the `ScrollPane` class.

Bear in mind these points about the `ScrollPane` component:

- The `ScrollPane` places the registration point of its content 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 buttons remain the same size. The scroll track and scroll box (thumb) expand or contract, and their hit areas are resized.

Using styles with the ScrollPane component

The `ScrollPane` supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>border styles</code>	Both	The <code>ScrollPane</code> component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See "RectBorder class" in Flash Help. The default border style is "inset".

Style	Theme	Description
<code>scrollTrackColor</code>	Sample	The background color for the scroll track. The default value is <code>0xCCCCCC</code> (light gray).
<code>symbolColor</code>	Sample	The color of the check mark. The default value is <code>0x000000</code> (black).
<code>symbolDisabledColor</code>	Sample	The color of the disabled check mark. The default value is <code>0x848384</code> (dark gray).

Using skins with the ScrollPane component

The ScrollPane component uses an instance of RectBorder for its border and scroll bars for scroll assets. For more information about skinning these visual elements, see “RectBorder class” in Flash Help and “Using skins with the ScrollPane component” on page 424.

ScrollPane class

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > View > ScrollView > ScrollPane

ActionScript Class Name mx.containers.ScrollPane

The properties of the ScrollPane class let you do the following at runtime: set the content, monitor the loading progress, and adjust the scroll amount.

Setting a property of the ScrollPane class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

You can set up a scroll pane so that users can drag the content within the pane. To do this, set the `scrollDrag` property to `true`; a pointing hand appears on the content. Unlike most other components, events are broadcast when the mouse button is pressed and continue broadcasting until the button is released. If the contents of a scroll pane have valid tab stops, you must set `scrollDrag` to `false`; otherwise, each mouse interaction with the contents will invoke scroll dragging.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.containers.ScrollPane.version);
```

Note: The code `trace(myScrollPaneInstance.version);` returns `undefined`.

Method summary for the ScrollPane class

The following table lists methods of the ScrollPane class.

Method	Description
ScrollPane.getBytesLoaded()	Returns the number of bytes of content loaded.
ScrollPane.getBytesTotal()	Returns the total number of bytes of content to be loaded.
ScrollPane.refreshPane()	Reloads the contents of the scroll pane.

Methods inherited from the UIObject class

The following table lists the methods the ScrollPane class inherits from the UIObject class. When calling these methods from the ScrollPane object, use the form

ScrollPaneInstance.methodName.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it will be redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the ScrollPane class inherits from the UIComponent class. When calling these methods from the ScrollPane object, use the form

ScrollPaneInstance.methodName.

Method	Description
UIComponent.getFocus()	Returns a reference to the object that has focus.
UIComponent.setFocus()	Sets focus to the component instance.

Property summary for the ScrollPane class

The following table lists properties of the ScrollPane class.

Method	Description
ScrollPane.content	A reference to the content loaded into the scroll pane.
ScrollPane.contentPath	An absolute or relative URL of the SWF or JPEG file to load into the scroll pane.
ScrollPane.hLineScrollSize	The amount of content to scroll horizontally when a scroll arrow is clicked.
ScrollPane.hPageScrollSize	The amount of content to scroll horizontally when the scroll track is clicked.

Method	Description
<code>ScrollPane.hPosition</code>	The horizontal pixel position of the scroll pane's horizontal scroll bar.
<code>ScrollPane.hScrollPolicy</code>	The status of the horizontal scroll bar. It can be always on ("on"), always off ("off"), or on when needed ("auto"). The default value is "auto".
<code>ScrollPane.scrollDrag</code>	Indicates whether scrolling occurs (<code>true</code>) or not (<code>false</code>) when a user drags on content within the scroll pane. The default value is <code>false</code> .
<code>ScrollPane.vLineScrollSize</code>	The amount of content to scroll vertically when a scroll arrow is clicked.
<code>ScrollPane.vPageScrollSize</code>	The amount of content to scroll vertically when the scroll track is clicked.
<code>ScrollPane.vPosition</code>	The pixel position of the scroll pane's vertical scroll bar.
<code>ScrollPane.vScrollPolicy</code>	The status of the vertical scroll bar. It can be always on ("on"), always off ("off"), or on when needed ("auto"). The default value is "auto".

Properties inherited from the UIObject class

The following table lists the properties the ScrollPane class inherits from the UIObject class. When accessing these properties from the ScrollPane object, use the form *ScrollPaneInstance.propertyName*.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `ScrollPane` class inherits from the `UIComponent` class. When accessing these properties from the `ScrollPane` object, use the form `ScrollPaneInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the `ScrollPane` class

The following table lists events of the `ScrollPane` class.

Event	Description
<code>ScrollPane.complete</code>	Broadcast when the scroll pane content is loaded.
<code>ScrollPane.progress</code>	Broadcast while the scroll pane content is loading.
<code>ScrollPane.scroll</code>	Broadcast when the scroll bar is clicked.

Events inherited from the `UIObject` class

The following table lists the events the `ScrollPane` class inherits from the `UIObject` class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the `UIComponent` class

The following table lists the events the `ScrollPane` class inherits from the `UIComponent` class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

ScrollPane.complete

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(complete){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.complete = function(eventObject){  
    ...  
}  
scrollPaneInstance.addEventListener("complete", listenerObject)
```

Description

Event; broadcast to all registered listeners when the content has finished loading.

The first usage example uses an `on()` handler and must be attached directly to a `ScrollPane` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `ScrollPane` instance `myScrollPaneComponent`, sends “_level0.myScrollPaneComponent” to the Output panel:

```
on(complete){  
    trace(this);  
}
```

The second usage example uses a dispatcher/listener event model. A component instance (`scrollPaneInstance`) dispatches an event (in this case, `complete`) and the event is handled by a function, also called a *handler*, on a listener object (`listenerObject`) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (`eventObject`) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following example creates a listener object with a complete event handler for the `ScrollPane` instance:

```
form.complete = function(eventObj){  
    // insert code to handle the event  
}  
ScrollPane.addEventListener("complete",form);
```

ScrollPane.content

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

ScrollPaneInstance.content

Description

Property (read-only); a reference to the content of the scroll pane. The value is undefined until the load begins.

Example

This example sets the `mcLoaded` variable to the value of the `content` property:

```
var mcLoaded = ScrollPane.content;
```

See also

[ScrollPane.contentPath](#)

ScrollPane.contentPath

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

ScrollPaneInstance.contentPath

Description

Property; a string that indicates an absolute or relative URL of the SWF or JPEG file to load into the scroll pane. A relative path must be relative to the SWF file that loads the content.

If you load content using a relative URL, the loaded content must be relative to the location of the SWF file that contains the scroll pane. For example, an application using a ScrollPane component that resides in the directory `/scrollpane/nav/example.swf` could load contents from the directory `/scrollpane/content/flash/logo.swf` by using the following `contentPath` property:
`"../content/flash/logo.swf"`

Example

The following example tells the scroll pane to display the contents of an image from the Internet:

```
scrollPane.contentPath = "http://imagecache2.allposters.com/images/43/  
033_302.jpg";
```

The following example tells the scroll pane to display the contents of a symbol from the library:

```
scrollPane.contentPath = "movieClip_Name";
```

The following example tells the scroll pane to display the contents of the local file `logo.swf`:

```
scrollPane.contentPath = "logo.swf";
```

See also

[ScrollPane.content](#)

ScrollPane.getBytesLoaded()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
ScrollPaneInstance.getBytesLoaded()
```

Parameters

None.

Returns

The number of bytes loaded in the scroll pane.

Description

Method; returns the number of bytes loaded in the ScrollPane instance. You can call this method at regular intervals while loading content to check its progress.

Example

This example creates a ScrollPane instance called `scrollPane`. It then defines a listener object called `loadListener` with a progress event handler that calls `getBytesLoaded()` to help determine the progress of the load:

```
createClassObject(mx.containers.ScrollPane, "scrollPane", 0);  
loadListener = new Object();
```

```

loadListener.progress = function(eventObj){
    // eventObj.target is the component that generated the change event
    var bytesLoaded = scrollPane.getBytesLoaded();
    var bytesTotal = scrollPane.getBytesTotal();
    var percentComplete = Math.floor(bytesLoaded/bytesTotal);

    if (percentComplete < 5) // loading begins
    {
        trace(" Starting loading contents from Internet");
    }
    else if(percentComplete = 50) // 50% complete
    {
        trace(" 50% contents downloaded ");
    }
}
scrollPane.addEventListener("progress", loadListener);
scrollPane.contentPath = "http://www.geocities.com/hcls_matrix/Images/
homeview5.jpg";

```

ScrollPane.getBytesTotal()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

scrollPaneInstance.getBytesTotal()

Parameters

None.

Returns

A number.

Description

Method; returns the total number of bytes to be loaded into the ScrollPane instance.

See also

[ScrollPane.getBytesLoaded\(\)](#)

ScrollPane.hLineScrollSize

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
scrollPaneInstance.hLineScrollSize
```

Description

Property; the number of pixels to move the content when an arrow in the horizontal scroll bar is clicked. The default value is 5.

Example

This example increases the horizontal scroll unit to 10:

```
scrollPane.hLineScrollSize = 10;
```

ScrollPane.hPageScrollSize

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
scrollPaneInstance.hPageScrollSize
```

Description

Property; the number of pixels to move the content when the track in the horizontal scroll bar is clicked. The default value is 20.

Example

This example increases the horizontal page scroll unit to 30:

```
scrollPane.hPageScrollSize = 30;
```

ScrollPane.hPosition

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
scrollPaneInstance.hPosition
```

Description

Property; the pixel position of the scroll pane's horizontal scroll box (thumb). The 0 position is at the extreme left end of the scroll track, which causes the left edge of the scroll pane content to be visible in the scroll pane.

Example

This example positions the scroll bar at pixel 20:

```
scrollPane.hPosition = 20;
```

ScrollPane.hScrollPolicy

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
scrollPaneInstance.hScrollPolicy
```

Description

Property; determines whether the horizontal scroll bar is always present ("on"), is never present ("off"), or appears automatically according to the size of the image ("auto"). The default value is "auto".

Example

The following code turns scroll bars on all the time:

```
scrollPane.hScrollPolicy = "on";
```

ScrollPane.progress

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(progress){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.progress = function(eventObject){  
    ...  
}  
scrollPaneInstance.addEventListener("progress", listenerObject)
```

Description

Event; broadcast to all registered listeners while content is loading. The progress event is not always broadcast; the `complete` event may be broadcast without any `progress` events being dispatched. This can happen especially if the loaded content is a local file. Your application triggers the `progress` event when the content starts loading by setting the value of the `contentPath` property.

The first usage example uses an `on()` handler and must be attached directly to a `ScrollPane` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `ScrollPane` component instance `mySPComponent`, sends “_level0.mySPComponent” to the Output panel:

```
on(progress){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*ScrollPaneInstance*) dispatches an event (in this case, `progress`) and the event is handled by a function, also called a *handler*, on a listener object (*ListenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*EventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following code creates a `ScrollPane` instance called `scrollPane`. It then creates a listener object with an event handler for the `progress` event that sends a message to the Output panel about how much content has loaded.

```
createClassObject(mx.containers.ScrollPane, "scrollPane", 0);
loadListener = new Object();
loadListener.progress = function(eventObj){
    // eventObj.target is the component that generated the progress event
    // --in this case, scrollPane
    trace("logo.swf has loaded " + scrollPane.getBytesLoaded() + " Bytes.");
    // track loading progress
}
scrollPane.addEventListener("complete", loadListener);
scrollPane.contentPath = "logo.swf";
```

ScrollPane.refreshPane()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
scrollPaneInstance.refreshPane()
```

Parameters

None.

Returns

Nothing.

Description

Method; refreshes the scroll pane after content is loaded. This method reloads the content. You could use this method if, for example, you've loaded a form into a scroll pane and an input property (for example, a text field) has been changed by ActionScript. In this case, you would call `refreshPane()` to reload the same form with the new values for the input properties.

Example

The following example refreshes the scroll pane instance `sp`:

```
sp.refreshPane();
```

ScrollPane.scroll

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(scroll){  
  ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.scroll = function(eventObject){  
  ...  
}  
scrollPaneInstance.addEventListener("scroll", listenerObject)
```

Event object

In addition to the standard event object properties, there are two additional properties defined for the `scroll` event: a `type` property whose value is "scroll", and a `direction` property whose value can be "vertical" or "horizontal".

In addition to the standard event object properties, there are two additional properties defined for the `ProgressBar.progress` event: `current` (the loaded value equals `total`), and `total` (the total value).

Description

Event; broadcast to all registered listeners when a user clicks the scroll bar buttons, scroll box (thumb), or scroll track. Unlike other events, the `scroll` event is broadcast when a user presses the mouse button on the scroll bar and continues broadcasting until the mouse is released.

The first usage example uses an `on()` handler and must be attached directly to a `ScrollPane` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the instance `sp`, sends “_level0.sp” to the Output panel:

```
on(scroll){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*ScrollPaneInstance*) dispatches an event (in this case, `scroll`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

This example creates a form listener object with a `scroll` callback function that’s registered to the `spInstance` instance. You must fill `spInstance` with content.

```
spInstance.contentPath = "mouse3.jpg";
form = new Object();
form.scroll = function(eventObj){
    trace("ScrollPane scrolled");
}
spInstance.addEventListener("scroll", form);
```

See also

[EventDispatcher.addEventListener\(\) in Flash Help](#)

ScrollPane.scrollDrag

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
ScrollPaneInstance.scrollDrag
```

Description

Property; a Boolean value that indicates whether scrolling occurs (*true*) or not (*false*) when a user drags within the scroll pane. The default value is *false*.

Example

This example causes the content to scroll when the user drags within the scroll pane:

```
ScrollPane.scrollDrag = true;
```

ScrollPane.vLineScrollSize

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
ScrollPaneInstance.vLineScrollSize
```

Description

Property; the number of pixels to move the content in the display area when the user clicks a scroll arrow in a vertical scroll bar. The default value is 5.

Example

This code causes the content in the display area to move 10 pixels when the vertical scroll arrows are clicked:

```
ScrollPane.vLineScrollSize = 10;
```

ScrollPane.vPageScrollSize

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
scrollPaneInstance.vPageScrollSize
```

Description

Property; the number of pixels to move the content in the display area when the user clicks the track in a vertical scroll bar. The default value is 20.

Example

This code causes the content in the display area to move 30 pixels when the vertical scroll track is clicked:

```
scrollPane.vPageScrollSize = 30;
```

ScrollPane.vPosition**Availability**

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
scrollPaneInstance.vPosition
```

Description

Property; the pixel position of the scroll pane's vertical scroll bar. The default value is 0.

ScrollPane.vScrollPolicy**Availability**

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
scrollPaneInstance.vScrollPolicy
```

Description

Property; determines whether the vertical scroll bar is always present ("on"), is never present ("off"), or appears automatically according to the size of the image ("auto"). The default value is "auto".

Example

The following code turns vertical scroll bars on all the time:

```
scrollPane.vScrollPolicy = "on";
```

SimpleButton class

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > SimpleButton

ActionScript Class Name mx.controls.SimpleButton

The properties of the SimpleButton class let you control the following at runtime:

- Whether a button has the emphasized look of a default push button
- Whether the button acts as a push button or as a toggle switch
- Whether a button is selected

Method summary for the SimpleButton class

There are no methods exclusive to the SimpleButton class.

Methods inherited from the UIObject class

The following table lists the methods the SimpleButton class inherits from the UIObject class. When calling these methods from the SimpleButton class, use the form *buttonInstance.methodName*.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it will be redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the SimpleButton class inherits from the UIComponent class. When calling these methods from the SimpleButton object, use the form *buttonInstance.methodName*.

Method	Description
UIComponent.getFocus()	Returns a reference to the object that has focus.
UIComponent.setFocus()	Sets focus to the component instance.

Property summary for the SimpleButton class

The following table lists properties of the SimpleButton class.

Property	Description
<code>SimpleButton.emphasized</code>	Indicates whether a button has the appearance of a default push button.
<code>SimpleButton.emphasizedStyleDeclaration</code>	The style declaration when the <code>emphasized</code> property is set to <code>true</code> .
<code>SimpleButton.selected</code>	A Boolean value indicating whether the button is selected (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .
<code>SimpleButton.toggle</code>	A Boolean value indicating whether the button behaves as a toggle switch (<code>true</code>) or not (<code>false</code>). The default value is <code>false</code> .

Properties inherited from the UIObject class

The following table lists the properties the SimpleButton class inherits from the UIObject class. When accessing these properties from the SimpleButton object, use the form *buttonInstance.propertyName*.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `SimpleButton` class inherits from the `UIComponent` class. When accessing these properties from the `SimpleButton` object, use the form `buttonInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the `SimpleButton` class

The following table lists the event of the `SimpleButton` class.

Event	Description
<code>SimpleButton.click</code>	Broadcast when a button is clicked.

Events inherited from the `UIObject` class

The following table lists the events the `SimpleButton` class inherits from the `UIObject` class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the `UIComponent` class

The following table lists the events the `SimpleButton` class inherits from the `UIComponent` class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

SimpleButton.click

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(click){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.click = function(eventObject){  
    ...  
}  
buttonInstance.addEventListener("click", listenerObject)
```

Description

Event; broadcast to all registered listeners when the mouse is clicked (released) over the button or if the button has focus and the Spacebar is pressed.

The first usage example uses an `on()` handler and must be attached directly to a Button component instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the Button component instance `myButtonComponent`, sends “_level0.myButtonComponent” to the Output panel:

```
on(click){  
    trace(this);  
}
```

The behavior of `this` is different when used inside an `on()` handler attached to a regular Flash button symbol. In that situation, `this` refers to the Timeline that contains the button. For example, the following code, attached to the button symbol instance `myButton`, sends “_level0” to the Output panel:

```
on(release){  
    trace(this);  
}
```

Note: The built-in ActionScript Button object doesn't have a `click` event; the closest event is `release`.

The second usage example uses a dispatcher/listener event model. A component instance (*buttonInstance*) dispatches an event (in this case, `click`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event occurs. When the event occurs, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call `addEventListener()` (see `EventDispatcher.addEventListener()` in Flash Help) on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

This example, written on a frame of the Timeline, sends a message to the Output panel when a button called `buttonInstance` is clicked. The first line specifies that the button act like a toggle switch. The second line creates a listener object called `form`. The third line defines a function for the `click` event on the listener object. Inside the function is a `trace()` statement that uses the event object that is automatically passed to the function (in this example, `eventObj`) to generate a message. The `target` property of an event object is the component that generated the event (in this example, `buttonInstance`). The `SimpleButton.selected` property is accessed from the event object’s `target` property. The last line calls `addEventListener()` from `buttonInstance` and passes it the `click` event and the `form` listener object as parameters.

```
buttonInstance.toggle = true;
form = new Object();
form.click = function(eventObj){
    trace("The selected property has changed to " + eventObj.target.selected);
}
buttonInstance.addEventListener("click", form);
```

The following code also sends a message to the Output panel when `buttonInstance` is clicked. The `on()` handler must be attached directly to `buttonInstance`.

```
on(click){
    trace("button component was clicked");
}
```

SimpleButton.emphasized

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

buttonInstance.emphasized

Description

Property; indicates whether the button is in an emphasized state (`true`) or not (`false`). The emphasized state is equivalent to the appearance of a default push button. In general, use the `FocusManager.defaultPushButton` property instead of setting the emphasized property directly. The default value is `false`.

If you aren't using `FocusManager.defaultPushButton`, you might just want to set a button to the emphasized state, or use the emphasized state to change text from one color to another. The following example sets the emphasized property for the button instance `myButton`:

```
_global.styles.foo = new CSSStyleDeclaration();
_global.styles.foo.color = 0xFF0000;
SimpleButton.emphasizedStyleDeclaration = "neutralStyle";
myButton.emphasized = true;
```

See also

[SimpleButton.emphasizedStyleDeclaration](#)

SimpleButton.emphasizedStyleDeclaration

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
buttonInstance.emphasizedStyleDeclaration
```

Description

Property (static); a string indicating the style declaration that formats a button when the emphasized property is set to `true`.

The `emphasizedStyleDeclaration` property is a static property of the `SimpleButton` class. Therefore, you must access it directly from `SimpleButton`, rather than from a `buttonInstance`, as in the following:

```
SimpleButton.emphasizedStyleDeclaration = "3dEmphStyle";
```

See also

[SimpleButton.emphasized](#)

SimpleButton.selected

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

`buttonInstance.selected`

Description

Property; a Boolean value that indicates whether the button is selected (`true`) or not (`false`). The default value is `false`.

SimpleButton.toggle

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

`buttonInstance.toggle`

Description

Property; a Boolean value that indicates whether the button acts as a toggle switch (`true`) or not (`false`). The default value is `false`.

If a button acts as a toggle switch, it stays pressed until you click it again to release it.

TextInput component

The TextInput component is a single-line text component that is a wrapper for the native ActionScript TextField object. You can use styles to customize the TextInput component; when an instance is disabled, its contents appear in a color represented by the `disabledColor` style. A TextInput component can also be formatted with HTML, or as a password field that disguises the text.

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 and right.
Shift+Tab	Moves focus to the previous object.
Tab	Moves focus to the next object.

For more information about controlling focus, see “Creating custom focus navigation” in Flash Help or “[FocusManager class](#)” on page 231.

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.

Using the `TextInput` component

You can use a `TextInput` component wherever you need a single-line text field. 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 if 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.

`TextInput` parameters

You can set the following authoring parameters for each `TextInput` component instance in the Property inspector or in the Component inspector:

text specifies the contents of the `TextInput` component. You cannot enter carriage returns in the Property inspector or Component inspector. The default value is "" (an empty string).

editable indicates whether the `TextInput` component is editable (`true`) or not (`false`). The default value is `true`.

password indicates whether the field is a password field (`true`) or not (`false`). The default value is `false`.

You can write ActionScript to control these and additional options for the `TextInput` component using its properties, methods, and events. For more information, see [“`TextInput` class” on page 449](#).

Creating an application with the `TextInput` component

The following procedure explains how to add a `TextInput` component to an application while authoring. In this example, the component is a password field with an event listener that determines if the proper number of characters has been entered.

To create an application with the `TextInput` component:

1. Drag a `TextInput` component from the Components panel to the Stage.
2. In the Property inspector, do the following:
 - Enter the instance name **passwordField**.
 - Leave the text parameter blank.
 - Set the editable parameter to `true`.
 - Set the password parameter to `true`.

3. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code:

```
textListener = new Object();
textListener.handleEvent = function (evt){
    if (evt.type == "enter"){
        trace("You must enter at least 8 characters");
    }
}
passwordField.addEventListener("enter", textListener);
```

This code sets up an `enter` event handler on the `TextInput` `passwordField` instance that verifies that the user entered the proper number of characters.

4. Once text is entered in the `passwordField` instance, you can get its value as follows:

```
var login = passwordField.text;
```

Customizing the TextInput component

You can transform a `TextInput` component horizontally while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use `UIObject.setSize()` or any applicable properties and methods of the `TextInput` class.

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.

Using styles with the TextInput component

The `TextInput` component has its `backgroundColor` and `borderStyle` style properties defined on a class style declaration. Class styles override global styles; therefore, if you want to set the `backgroundColor` and `borderStyle` style properties, you must create a different custom style declaration or define it on the instance.

A `TextInput` component supports the following styles:

Style	Theme	Description
<code>backgroundColor</code>		The background color. The default color is white.
<i>border styles</i>	Both	The <code>TextArea</code> component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See "RectBorder class" in Flash Help.
		The default border style is "inset".
<code>marginLeft</code>	Both	A number indicating the left margin for text. The default value is 0.
<code>marginRight</code>	Both	A number indicating the right margin for text. The default value is 0.

Style	Theme	Description
color	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
disabledColor	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
embedFonts	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
fontFamily	Both	The font name for text. The default value is <code>"_sans"</code> .
fontSize	Both	The point size for the font. The default value is 10.
fontStyle	Both	The font style: either <code>"normal"</code> or <code>"italic"</code> . The default value is <code>"normal"</code> .
fontWeight	Both	The font weight: either <code>"none"</code> or <code>"bold"</code> . The default value is <code>"none"</code> . All components can also accept the value <code>"normal"</code> in place of <code>"none"</code> during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return <code>"none"</code> .
textAlign	Both	The text alignment: either <code>"left"</code> , <code>"right"</code> , or <code>"center"</code> . The default value is <code>"left"</code> .
textIndent	Both	A number indicating the text indent. The default value is 0.
textDecoration	Both	The text decoration: either <code>"none"</code> or <code>"underline"</code> . The default value is <code>"none"</code> .

The `TextArea` and `TextInput` components both use the same styles and are often used in the same manner. Thus, by default they share the same class-level style declaration. For example, the following code sets a style on the `TextArea` declaration but it affects both `TextArea` and `TextInput` components.

```
_global.styles.TextArea.setStyle("disabledColor", 0xBBBBFF);
```

To separate the components and provide class-level styles for one and not the other, create a new style declaration.

```
import mx.styles.CSSStyleDeclaration;
_global.styles.TextInput = new CSSStyleDeclaration();
_global.styles.TextInput.setStyle("disabledColor", 0xFFBBB);
```

Notice how this example does not check if `_global.styles.TextInput` existed before overwriting it; in this example, you know it exists and you want to overwrite it.

Using skins with the `TextInput` component

The `TextArea` component uses an instance of `RectBorder` for its border. For more information about skinning these visual elements, see “`RectBorder` class” in Flash Help.

TextInput class

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > TextInput

ActionScript Class Name mx.controls.TextInput

The properties of the TextInput class let you set the text content, formatting, and horizontal position at runtime. You can also indicate whether the field is editable, and whether it is a “password” field. You can also restrict the characters that a user can enter.

Setting a property of the TextInput class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

The TextInput component uses the Focus Manager to override the default Flash Player focus rectangle and draw a custom focus rectangle with rounded corners. For more information, see “[FocusManager class](#)” on page 231.

The TextInput component supports CSS styles and any additional HTML styles supported by Flash Player. For information about CSS support, see the W3C specification at www.w3.org/TR/REC-CSS2/.

You can manipulate the text string by using the string returned by the text object.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.TextInput.version);
```

Note: The code `trace(myTextInputInstance.version);` returns `undefined`.

Method summary for the TextInput class

There are no methods exclusive to the TextInput class.

Methods inherited from the UIObject class

The following table lists the methods the TextInput class inherits from the UIObject class. When calling these methods from the TextInput object, use the form *TextInputInstance.methodName*.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it will be redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.

Method	Description
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the `UIComponent` class

The following table lists the methods the `TextInput` class inherits from the `UIComponent` class. When calling these methods from the `TextInput` object, use the form *TextInputInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the `TextInput` class

The following table lists properties of the `TextInput` class.

Property	Description
<code>TextInput.editable</code>	A Boolean value indicating whether the field is editable (<code>true</code>) or not (<code>false</code>).
<code>TextInput.hPosition</code>	The horizontal scrolling position of the text in the field.
<code>TextInput.length</code>	The number of characters in a <code>TextInput</code> component. This property is read-only.
<code>TextInput.maxChars</code>	The maximum number of characters that a user can enter in the text field.
<code>TextInput.maxHPosition</code>	The maximum possible value for <code>TextField.hPosition</code> . This property is read-only.
<code>TextInput.password</code>	A Boolean value that indicates whether the text field is a password field that hides the entered characters.
<code>TextInput.restrict</code>	Indicates which characters a user can enter in a text field.
<code>TextInput.text</code>	Sets the text content of a <code>TextInput</code> component.

Properties inherited from the `UIObject` class

The following table lists the properties the `TextInput` class inherits from the `UIObject` class. When accessing these properties from the `TextInput` object, use the form `TextInputInstance.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `TextInput` class inherits from the `UIComponent` class. When accessing these properties from the `TextInput` object, use the form `TextInputInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the `TextInput` class

The following table lists events of the `TextInput` class.

Event	Description
<code>TextInput.change</code>	Broadcast when the <code>TextInput</code> field changes.
<code>TextInput.enter</code>	Broadcast when the Enter key is pressed.

Events inherited from the UIObject class

The following table lists the events the `TextInput` class inherits from the `UIObject` class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the `TextInput` class inherits from the `UIComponent` class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

TextInput.change

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(change){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.change = function(eventObject){  
    ...  
}  
textInputInstance.addEventListener("change", listenerObject)
```

Description

Event; notifies listeners that text has changed. This event is broadcast after the text has changed. This event cannot be used to prevent certain characters from being added to the component's text field; for that purpose, use `TextInput.restrict`. This event is triggered only by user input, not by programmatic change.

The first usage example uses an `on()` handler and must be attached directly to a `TextInput` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the instance `myTextInput`, sends “_level0.myTextInput” to the Output panel:

```
on(change){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*textInputInstance*) dispatches an event (in this case, *change*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

This example sets a flag in the application that indicates if contents in the `TextInput` field have changed:

```
form.change = function(eventObj){
    // note: eventObj.target refers to the component that generated the change
    // event, i.e., the TextInput component.
    myFormChanged.visible = true; // set a change indicator if the contents
    changed;
}
myInput.addEventListener("change", form);
```

See also

`EventDispatcher.addEventListener()` in Flash Help

TextInput.editable

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

`textInputInstance.editable`

Description

Property; a Boolean value that indicates whether the component is editable (`true`) or not (`false`). The default value is `true`.

TextInput.enter

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(enter){
    ...
}
```

Usage 2:

```
listenerObject = new Object();
listenerObject.enter = function(eventObject){
    ...
}
textInputInstance.addEventListener("enter", listenerObject)
```

Description

Event; notifies listeners that the Enter key has been pressed.

The first usage example uses an `on()` handler and must be attached directly to a `TextInput` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the instance `myTextInput`, sends “`_level0.myTextInput`” to the Output panel:

```
on(enter){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*textInputInstance*) dispatches an event (in this case, `enter`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

This example sets a flag in the application that indicates if contents in the `TextInput` field have changed:

```
form.enter = function(eventObj){
    // note: eventObj.target refers the component that generated the enter event,
    // i.e., the TextInput component.
    myFormChanged.visible = true;
    // set a change indicator if the user presses Enter;
}
myInput.addEventListener("enter", form);
```

See also

`EventDispatcher.addEventListener()` in Flash Help

TextInput.hPosition

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

textInputInstance.hPosition

Description

Property; defines the horizontal position of the text in the field. The default value is 0.

Example

The following code displays the leftmost character in the field:

```
myTextInput.hPosition = 0;
```

TextInput.length

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

inputInstance.length

Description

Property (read-only); a number that indicates the number of characters in a `TextInput` component. A character such as tab ("`\t`") counts as one character. The default value is 0.

Example

The following code determines the number of characters in the `myTextInput` string and copies it to the `length` variable:

```
var length = myTextInput.length;
```

TextInput.maxChars

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

textInputInstance.maxChars

Description

Property; the maximum number of characters that the text field can contain. A script may insert more text than the `maxChars` property allows; this property indicates only how much text a user can enter. If this property is `null`, there is no limit to the amount of text a user can enter. The default value is `null`.

Example

The following example limits the number of characters a user can enter to 255:

```
myTextInput.maxChars = 255;
```

TextInput.maxHPosition

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

textInputInstance.maxHPosition

Description

Property (read-only); indicates the maximum value of `TextInput.hPosition`. The default value is 0.

Example

The following code scrolls to the far right:

```
myTextInput.hPosition = myTextInput.maxHPosition;
```

TextInput.password

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

textInputInstance.password

Description

Property; a Boolean value indicating whether the text field is a password field (`true`) or not (`false`). If this property is `true`, the text field is a password text field and hides the input characters. If this property is `false`, the text field is not a password text field. The default value is `false`.

Example

The following code makes the text field a password field that displays all characters as asterisks (*):

```
myTextInput.password = true;
```

TextInput.restrict

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
textInputInstance.restrict
```

Description

Property; indicates the set of characters that a user can enter in the text field. The default value is undefined. If this property is `null` or an empty string (""), a user can enter any character. If this property is a string of characters, the user can enter only characters in the string; the string is scanned from left to right. You can specify a range by using a dash (-).

If the string begins with `^`, all characters that follow the `^` are considered unacceptable characters. If the string does not begin with `^`, the characters in the string are considered acceptable. The `^` can also be used as a toggle between acceptable and unacceptable characters.

For example, the following code allows A-Z except X and Q:

```
Ta.restrict = "A-Z^XQ";
```

You can use the backslash (`\`) to enter a hyphen (-), caret (^), or backslash (`\`) character, as shown here:

```
\^  
\-  
\\
```

When you enter the `\` character in the Actions panel within double quotation marks, it has a special meaning for the Actions panel's double-quote interpreter. It signifies that the character following the `\` should be treated as is. For example, you could use the following code to enter a single quotation mark:

```
var leftQuote = "\'";
```

The Actions panel's `restrict` interpreter also uses `\` as an escape character. Therefore, you may think that the following should work:

```
myText.restrict = "0-9\-\^\\";
```

However, since this expression is surrounded by double quotation marks, the following value is sent to the `restrict` interpreter: `0-9-^\`, and the `restrict` interpreter doesn't understand this value.

Because you must enter this expression within double quotation marks, you must not only provide the expression for the `restrict` interpreter, but you must also escape the Actions panel's built-in interpreter for double quotation marks. To send the value `0-9\-\^\` to the `restrict` interpreter, you must enter the following code:

```
myText.restrict = "0-9\\-\\^\\\"";
```

The `restrict` property restricts only user interaction; a script may put any text into the text field. This property does not synchronize with the Embed Font Outlines check boxes in the Property inspector.

Example

In the following example, the first line of code limits the text field to uppercase letters, numbers, and spaces. The second line of code allows all characters except lowercase letters.

```
my_txt.restrict = "A-Z 0-9";  
my_txt.restrict = "^a-z";
```

The following code allows a user to enter the characters “0 1 2 3 4 5 6 7 8 9 - ^ \” in the instance `myText`. You must use a double backslash to escape the characters -, ^, and \. The first \ escapes the double quotation marks, and the second \ tells the interpreter that the next character should not be treated as a special character.

```
myText.restrict = "0-9\\-\\^\\\\";
```

TextInput.text

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

textInputInstance.text

Description

Property; the text contents of a `TextInput` component. The default value is "" (an empty string).

Example

The following code places a string in the `myTextInput` instance, and then traces that string to the Output panel:

```
myTextInput.text = "The Royal Nonesuch";  
trace(myTextInput.text); // traces "The Royal Nonesuch"
```

TextArea component

The `TextArea` component wraps the native ActionScript `TextField` object. You can use styles to customize the `TextArea` component; when an instance is disabled, its contents display in a color represented by the `disabledColor` style. A `TextArea` component can also be formatted with HTML, or as a password field that disguises the text.

A `TextArea` 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 `TextArea` instance has focus, you can use the following keys to control it:

Key	Description
Arrow keys	Move the insertion point one line up, down, left, or right.
Page Down	Moves one screen down.
Page Up	Moves one screen up.
Shift+Tab	Moves focus to the previous object.
Tab	Moves focus to the next object.

For more information about controlling focus, see “Creating custom focus navigation” in Flash Help or “[FocusManager class](#)” on page 231.

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.

When you add the `TextArea` component to an application, you can use the Accessibility panel to make it accessible to screen readers.

Using the `TextArea` component

You can use a `TextArea` component wherever you need a multiline text field. If you need a single-line text field, use the [TextInput component](#). For example, you could use a `TextArea` component as a comment field in a form. You could set up a listener that checks if 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.

`TextArea` parameters

You can set the following authoring parameters for each `TextArea` component instance in the Property inspector or in the Component inspector:

text indicates the contents of the `TextArea` component. You cannot enter carriage returns in the Property inspector or Component inspector. The default value is "" (an empty string).

html indicates whether the text is formatted with HTML (`true`) or not (`false`). The default value is `false`.

editable indicates whether the `TextArea` component is editable (`true`) or not (`false`). The default value is `true`.

wordWrap indicates whether the text wraps (`true`) or not (`false`). The default value is `true`.

You can write ActionScript to control these and additional options for the `TextArea` component using its properties, methods, and events. For more information, see [“TextArea class” on page 463](#).

Creating an application with the `TextArea` component

The following procedure explains how to add a `TextArea` component to an application while authoring. In this example, the component is a Comment field with an event listener that determines if a user has entered text.

To create an application with the `TextArea` component:

1. Drag a `TextArea` component from the Components panel to the Stage.
2. In the Property inspector, enter the instance name **comment**.
3. In the Property inspector, set parameters as you wish. However, leave the text parameter blank, the editable parameter set to `true`, and the password parameter set to `false`.
4. Select Frame 1 in the Timeline, open the Actions panel, and enter the following code:

```
textListener = new Object();
textListener.handleEvent = function (evt){
    if (comment.length < 1) {
        Alert(_root, "Error", "You must enter at least a comment in this field",
            mxModal | mxOK);
    }
}
comment.addEventListener("focusOut", textListener);
```

This code sets up a `focusOut` event handler on the `TextArea` `comment` instance that verifies that the user typed something in the text field.

5. Once text is entered in the comment instance, you can get its value as follows:

```
var login = comment.text;
```

Customizing the `TextArea` component

You can transform a `TextArea` component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use `UIObject.setSize()` or any applicable properties and methods of the [`TextArea` class](#).

When a `TextArea` component is resized, the border is resized to the new bounding box. The scroll bars are placed on the bottom and right edges if they are required. The text field is then resized within the remaining area; there are no fixed-size elements in a `TextArea` component. If the `TextArea` component is too small to display the text, the text is clipped.

Using styles with the `TextArea` component

The `TextArea` component supports one set of component styles for all text in the field. However, you can also display HTML that is compatible with Flash Player HTML rendering. To display HTML text, set `TextArea.html` to `true`.

The `TextArea` component has its `backgroundColor` and `borderStyle` style properties defined on a class style declaration. Class styles override global styles; therefore, if you want to set the `backgroundColor` and `borderStyle` style properties, you must create a different custom style declaration on the instance.

If the name of a style property ends in “Color”, it is a color style property and behaves differently than noncolor style properties. For more information, see “Using styles to customize component color and text” in Flash Help.

A `TextArea` component supports the following styles:

Style	Theme	Description
<code>backgroundColor</code>	Both	The background color. The default color is white.
<i>border styles</i>	Both	The <code>TextArea</code> component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See “ <code>RectBorder</code> class” in Flash Help. The default border style is “inset”.
<code>marginLeft</code>	Both	A number indicating the left margin for text. The default value is 0.
<code>marginRight</code>	Both	A number indicating the right margin for text. The default value is 0.
<code>color</code>	Both	The text color. The default value is 0x0B333C for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is 0x848384 (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is “_sans”.
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either “normal” or “italic”. The default value is “normal”.
<code>fontWeight</code>	Both	The font weight: either “none” or “bold”. The default value is “none”. All components can also accept the value “normal” in place of “none” during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return “none”.
<code>textAlign</code>	Both	The text alignment: either “left”, “right”, or “center”. The default value is “left”.
<code>textIndent</code>	Both	A number indicating the text indent. The default value is 0.
<code>textDecoration</code>	Both	The text decoration: either “none” or “underline”. The default value is “none”.

The `TextArea` and `TextInput` components use exactly the same styles and are often used in the same manner. Thus, by default they share the same class-level style declaration. For example, the following code sets a style on the `TextInput` declaration, but it affects both `TextInput` and `TextArea` components.

```
_global.styles.TextInput.setStyle("disabledColor", 0xBBBBFF);
```

To separate the components and provide class-level styles for one and not the other, create a new style declaration.

```
import mx.styles.CSSStyleDeclaration;
_global.styles.TextArea = new CSSStyleDeclaration();
_global.styles.TextArea.setStyle("disabledColor", 0xFFBBBB);
```

This example does not check if `_global.styles.TextArea` existed before overwriting it; it assumes you know it exists and want to overwrite it.

Using skins with the `TextArea` component

The `TextArea` component uses an instance of `RectBorder` for its border and scroll bars for scrolling images. For more information about skinning these visual elements, see “[RectBorder class](#)” in Flash Help and “[Using skins with the `TextArea` component](#)” on page 463.

TextArea class

Inheritance `MovieClip` > [UIObject class](#) > [UIComponent class](#) > `View` > `ScrollView` > `TextArea`

ActionScript Class Name `mx.controls.TextArea`

The properties of the `TextArea` class let you set the text content, formatting, and horizontal and vertical position at runtime. You can also indicate whether the field is editable, and whether it is a “password” field. You can also restrict the characters that a user can enter.

Setting a property of the `TextArea` class with ActionScript overrides the parameter of the same name set in the Property inspector or Component inspector.

The `TextArea` component overrides the default Flash Player focus rectangle and draws a custom focus rectangle with rounded corners.

The `TextArea` component supports CSS styles and any additional HTML styles supported by Flash Player.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.TextArea.version);
```

Note: The code `trace(myTextAreaInstance.version);` returns `undefined`.

Method summary for the `TextArea` class

There are no methods exclusive to the `TextArea` class.

Methods inherited from the UIObject class

The following table lists the methods the `TextArea` class inherits from the `UIObject` class. When calling these methods from the `TextArea` object, use the form `TextAreaInstance.methodName`.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it will be redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the `TextArea` class inherits from the `UIComponent` class. When calling these methods from the `TextArea` object, use the form `TextAreaInstance.methodName`.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the TextArea class

The following table lists properties of the `TextArea` class.

Property	Description
<code>TextArea.editable</code>	A Boolean value indicating whether the field is editable (<code>true</code>) or not (<code>false</code>).
<code>TextArea.hPosition</code>	Defines the horizontal position of the text in the field.
<code>TextArea.hScrollPolicy</code>	Indicates whether the horizontal scroll bar is always on ("on"), is never on ("off"), or turns on when needed ("auto").
<code>TextArea.html</code>	A Boolean value that indicates whether the text field can be formatted with HTML.
<code>TextArea.length</code>	Read-only; the number of characters in the text field.

Property	Description
<code>TextArea.maxChars</code>	The maximum number of characters that the text field can contain.
<code>TextArea.maxHPosition</code>	Read-only; the maximum value of <code>TextArea.hPosition</code> .
<code>TextArea.maxVPosition</code>	Read-only; the maximum value of <code>TextArea.vPosition</code> .
<code>TextArea.password</code>	A Boolean value indicating whether the field is a password field (<code>true</code>) or not (<code>false</code>).
<code>TextArea.restrict</code>	The set of characters that a user can enter in the text field.
<code>TextArea.styleSheet</code>	Attaches a style sheet to the specified TextArea component.
<code>TextArea.text</code>	The text contents of a TextArea component.
<code>TextArea.vPosition</code>	A number indicating the vertical scrolling position.
<code>TextArea.vScrollPolicy</code>	Indicates whether the vertical scroll bar is always on (" <code>on</code> "), is never on (" <code>off</code> "), or turns on when needed (" <code>auto</code> ").
<code>TextArea.wordWrap</code>	A Boolean value indicating whether the text wraps (<code>true</code>) or not (<code>false</code>).

Properties inherited from the UIObject class

The following table lists the properties the TextArea class inherits from the UIObject class. When accessing these properties from the TextArea object, use the form `TextAreaInstance.propertyName`.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `TextArea` class inherits from the `UIComponent` class. When accessing these properties from the `TextArea` object, use the form

`TextAreaInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the `TextArea` class

The following table lists the event of the `TextArea` class.

Event	Description
<code>TextArea.change</code>	Notifies listeners that text has changed.

Events inherited from the `UIObject` class

The following table lists the events the `TextArea` class inherits from the `UIObject` class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the `UIComponent` class

The following table lists the events the `TextArea` class inherits from the `UIComponent` class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

TextArea.change

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(change){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.change = function(eventObject){  
    ...  
}  
textAreaInstance.addEventListener("change", listenerObject)
```

Description

Event; notifies listeners that text has changed. This event is broadcast after the text has changed. This event cannot be used to prevent certain characters from being added to the component's text field; for this purpose, use [TextArea.restrict](#).

The first usage example uses an `on()` handler and must be attached directly to a `TextArea` instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the instance `myTextArea`, sends “_level0.myTextArea” to the Output panel:

```
on(change){  
    trace(this);  
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*textAreaInstance*) dispatches an event (in this case, *change*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

This example uses the dispatcher/listener event model to track the total of number of times the text field has changed in a `TextArea` component named `myTextArea`:

```
// define a listener object
var myTextAreaListener:Object = new Object();

// create a Number variable to track the number of changes to the TextArea
var changeCount:Number = 0;

// define a function that is executed whenever the listener receives
// notification of a change in the TextArea component
myTextAreaListener.change = function(eventObj) {
    changeCount++;
    trace("Text has changed " + changeCount + " times now!");
    trace("It now contains: " + eventObj.target.text);
}

// register the listener object with the TextArea component instance
myTextArea.addEventListener("change", myTextAreaListener);
```

See also

`EventDispatcher.addEventListener()` in [Flash Help](#)

TextArea.editable

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.editable

Description

Property; a Boolean value that indicates whether the component is editable (`true`) or not (`false`). The default value is `true`.

TextArea.hPosition

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.hPosition

Description

Property; defines the horizontal position of the text in the field. The default value is 0.

Example

The following code displays the leftmost characters in the field:

```
myTextArea.hPosition = 0;
```

TextArea.hScrollPolicy**Availability**

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
textAreaInstance.hScrollPolicy
```

Description

Property; determines whether the horizontal scroll bar is always present ("on"), is never present ("off"), or appears automatically according to the size of the field ("auto"). The default value is "auto".

Example

The following code turns horizontal scroll bars on all the time:

```
text.hScrollPolicy = "on";
```

TextArea.html**Availability**

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
textAreaInstance.html
```

Description

Property; a Boolean value that indicates whether the text field is formatted with HTML (*true*) or not (*false*). If the `html` property is *true*, the text field is an HTML text field. If `html` is *false*, the text field is a non-HTML text field. The default value is *false*.

Example

The following example makes the `myTextArea` field an HTML text field and then formats the text with HTML tags:

```
myTextArea.html = true;
myTextArea.text = "The <b>Royal</b> Nonesuch"; // displays "The Royal
Nonesuch"
```

TextArea.length

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
textAreaInstance.length
```

Description

Property (read-only); indicates the number of characters in a text field. This property returns the same value as the ActionScript `text.length` property, but is faster. A character such as tab ("`\t`") counts as one character. The default value is 0.

Example

The following example gets the length of the text field and copies it to the `length` variable:

```
var length = myTextArea.length; // find out how long the text string is
```

TextArea.maxChars

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
textAreaInstance.maxChars
```

Description

Property; the maximum number of characters that the text field can contain. A script may insert more text than the `maxChars` property allows; the property indicates only how much text a user can enter. If the value of this property is `null`, there is no limit to the amount of text a user can enter. The default value is `null`.

Example

The following example limits the number of characters a user can enter to 255:

```
myTextArea.maxChars = 255;
```

TextArea.maxHPosition

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
textAreaInstance.maxHPosition
```

Description

Property (read-only); the maximum value of [TextArea.hPosition](#). The default value is 0.

Example

The following code causes the text to scroll to the far right:

```
myTextArea.hPosition = myTextArea.maxHPosition;
```

See also

[TextArea.vPosition](#)

TextArea.maxVPosition

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
textAreaInstance.maxVPosition
```

Description

Property (read-only); indicates the maximum value of [TextArea.vPosition](#). The default value is 0.

Example

The following code causes the text to scroll to the bottom of the component:

```
myTextArea.vPosition = myTextArea.maxVPosition;
```

See also

[TextArea.hPosition](#)

TextArea.password

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.password

Description

Property; a Boolean value indicating whether the text field is a password field (`true`) or not (`false`). If `password` is `true`, the text field is a password text field and hides the input characters. If `password` is `false`, the text field is not a password text field. The default value is `false`.

Example

The following code makes the text field a password field that displays all characters as asterisks (*):

```
myTextArea.password = true;
```

TextArea.restrict

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.restrict

Description

Property; indicates the set of characters that users can enter in the text field. The default value is `undefined`. If this property is `null`, users can enter any character. If this property is an empty string, no characters can be entered. If this property is a string of characters, users can enter only characters in the string; the string is scanned from left to right. You can specify a range by using a dash (-).

If the string begins with `^`, all characters that follow the `^` are considered unacceptable characters. If the string does not begin with `^`, the characters in the string are considered acceptable. The `^` can also be used as a toggle between acceptable and unacceptable characters.

For example, the following code allows A-Z except X and Q:

```
Ta.restrict = "A-Z^XQ";
```

The `restrict` property only restricts user interaction; a script may put any text into the text field. This property does not synchronize with the Embed Font Outlines check boxes in the Property inspector.

Example

In the following example, the first line of code limits the text field to uppercase letters, numbers, and spaces. The second line of code allows all characters except lowercase letters.

```
my_txt.restrict = "A-Z 0-9"; // limit control to uppercase letters, numbers,
    and spaces
my_txt.restrict = "^a-z"; // allow all characters, except lowercase letters
```

TextArea.styleSheet

Availability

Flash Player 7.

Usage

```
TextAreaInstance.styleSheet = TextFieldStyleSheetObject
```

Description

Property; attaches a style sheet to the `TextArea` component specified by *TextAreaInstance*.

The style sheet associated with a `TextArea` component may be changed at any time. If the style sheet in use is changed, the `TextArea` component is redrawn with the new style sheet. The style sheet may be set to `null` or `undefined` to remove the style sheet. If the style sheet in use is removed, the `TextArea` component is redrawn without a style sheet. The formatting done by a style sheet is not retained if the style sheet is removed.

Example

The following code creates a new `StyleSheet` object named `styles` with the new `TextField.StyleSheet` constructor. It then defines styles for `<html>`, `<body>` and `<td>` tags. It then uses `LoadVars.load` to load an HTML file named `myText.htm`. That file contains text within `<html>`, `<body>` and `<td>` tags. When the text is displayed in the `TextArea` instance `MyTextArea`, the text within those tags is styled according to the `StyleSheet` object `styles`.

```
// create the new StyleSheet object
var styles = new TextField.StyleSheet();

// define styles for <html>, <body>, and <td>...
styles.setStyle("html",
    {fontFamily: 'Arial,Helvetica,sans-serif',
    fontSize: '12px',
    color: '#0000FF'});

styles.setStyle("body",
    {color: '#00CCFF',
    textDecoration: 'underline'});

styles.setStyle("td",
    {fontFamily: 'Arial,Helvetica,sans-serif',
    fontSize: '24px',
    color: '#006600'});

// set the TextAreaInstance.styleSheet property to the newly defined
// styleSheet object named styles
```

```
myTextArea.styleSheet=styles;
myTextArea.html=true;

// Load text to display and define onLoad handler
var myVars:LoadVars = new LoadVars();
myVars.load("myText.htm");
myVars.onData = function(content) {
    _root.myTextArea.text = content;
};
```

TextArea.text

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.text

Description

Property; the text contents of a TextArea component. The default value is "" (an empty string).

Example

The following code places a string in the myTextArea instance, and then traces that string to the Output panel:

```
myTextArea.text = "The Royal Nonesuch";
trace(myTextArea.text); // traces "The Royal Nonesuch"
```

TextArea.vPosition

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

textAreaInstance.vPosition

Description

Property; defines the vertical scroll position of text in a text field. This property is useful for directing users to a specific paragraph in a long passage, or creating scrolling text fields. You can get and set this property. The default value is 0.

Example

The following code displays the topmost characters in a field:

```
myTextArea.vPosition = 0;
```

TextArea.vScrollPolicy

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
textAreaInstance.vScrollPolicy
```

Description

Property; determines whether the vertical scroll bar is always present ("on"), is never present ("off"), or appears automatically according to the size of the field ("auto"). The default value is "auto".

Example

The following code turns vertical scroll bars off all the time:

```
text.vScrollPolicy = "off";
```

TextArea.wordWrap

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
textAreaInstance.wordWrap
```

Description

Property; a Boolean value that indicates whether the text wraps (`true`) or not (`false`). The default value is `true`.

TossButton component

The TossButton component is a small round button with a curved arrow on it. It changes state when you move a mouse pointer over it. Use this button for sending data from your pod to its parent application and to start the parent application from the pod. For example, a user might have the Console open with your application's pod open to view headlines from a news feed. If the user wants more detail for a headline in the pod, the user can, with the headline selected, click the toss button to start the application. The application usually has more space to show detail and also offers more functionality than the pod.



Toss Button in a pod

Using the TossButton component

The TossButton component has the same methods, properties, and events as the Button object. For detailed information on the Button object, see the help system in your Flash authoring tool. However, for most developers, use of the TossButton component will simply require the `onRelease` event handler, which is documented here.

MTossButton.onRelease

Availability

Macromedia Central Player.

Edition

Macromedia Central SDK.

Usage

```
myTossButton.onRelease()
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the button is released. You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onRelease` method that sends a trace action to the Output panel:

```
myButton.onRelease = function () {  
    trace ("onRelease called");  
};
```

Tree component

The Tree component allows a user to view hierarchical data. The tree appears in a box like the List component, but each item in a tree is called a *node* and can be either a *leaf* or a *branch*. By default, a leaf is represented by a text label beside a file icon, and a branch is represented by a text label beside a folder icon with an expander arrow (disclosure triangle) that a user can open to expose children. The children of a branch can be leaves or branches.

The data of a tree component must be provided from an XML data source. For more information, see [“Using the Tree component” on page 478](#).

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

Key	Description
Down Arrow	Moves selection down one item.
Up Arrow	Moves selection up one item.
Right Arrow	Opens a selected branch node. If a branch is already open, moves to first child node.
Left Arrow	Closes a selected branch node. If on a leaf node of a closed branch node, moves to parent node.
Space	Opens or closes a selected branch node.
End	Moves selection to the bottom of the list.
Home	Moves selection to the top of the list.
Page Down	Moves selection down one page.
Page Up	Moves selection up one page.
Control	Allows multiple noncontiguous selections.
Shift	Allows multiple contiguous selections.

The Tree component cannot be made accessible to screen readers.

Using the Tree component

The Tree component can be used to represent hierarchical data such as e-mail client folders, file browser panes, or category browsing systems for inventory. Most often, the data for a tree is retrieved from a server in the form of XML, but it can also be well-formed XML that is created during authoring in Flash. The best way to create XML for the tree is to use the `TreeDataProvider` interface. You can also use the ActionScript XML class or build an XML string. After you create an XML data source (or load one from an external source), you assign it to `Tree.dataProvider`.

The Tree component comprises two sets of APIs: the `Tree` class and the `TreeDataProvider` interface. The `Tree` class contains the visual configuration methods and properties. The `TreeDataProvider` interface lets you construct XML and add it to multiple tree instances. A `TreeDataProvider` object broadcasts changes to any trees that use it. In addition, any XML or `XMLNode` object that exists on the same frame as a tree or a menu is automatically given the `TreeDataProvider` methods and properties. For more information, see “`TreeDataProvider` interface (Flash Professional only)” in Flash Help.

Formatting XML for the Tree component

The Tree component is designed to display hierarchical data structures using XML as the data model. It is important to understand the relationship of the XML data source to the Tree component.

Consider the following XML data source sample:

```
<node>
  <node label="Mail">
    <node label="INBOX"/>
    <node label="Personal Folder">
      <node label="Business" isBranch="true" />
      <node label="Demo" isBranch="true" />
      <node label="Personal" isBranch="true" />
      <node label="Saved Mail" isBranch="true" />
      <node label="bar" isBranch="true" />
    </node>
    <node label="Sent" isBranch="true" />
    <node label="Trash"/>
  </node>
</node>
```

Note: The `isBranch` attribute is read-only; you cannot set it directly. To set it, call `Tree.setIsBranch()`.

Nodes in the XML data source can have any name. Notice in the previous example that each node is named with the generic name `node`. The tree reads through the XML and builds the display hierarchy according to the nested relationship of the nodes.

Each XML node can be displayed as one of two types in the tree: branch or leaf. Branch nodes can contain multiple child nodes and appear as a folder icon with an expander arrow that allows users to open and close the folder. Leaf nodes appear as a file icon and cannot contain child nodes. Both leaves and branches can be roots; a root node appears at the top level of the tree and has no parent. The icons are customizable; for more information, see “[Using skins with the Tree component](#)” on page 488.

There are many ways to structure XML, but the Tree component cannot use all types of XML structures. Do not nest node attributes in a child node; each node should contain all its necessary attributes. Also, the attributes of each node should be consistent to be useful. For example, to describe a mailbox structure with a Tree component, use the same attributes on each node (message, data, time, attachments, and so on). This lets the tree know what it expects to render, and lets you loop through the hierarchy to compare data.

When a Tree displays a node, it uses the `label` attribute of the node by default as the text label. If any other attributes exist, they become additional properties of the node's attributes within the tree.

The actual root node is interpreted as the Tree component itself. This means that the first child (in the previous example, `<node label="Mail">`), is rendered as the root node in the tree view. This means that a tree can have multiple root folders. In the example, there is only one root folder displayed in the tree: Mail. However, if you were to add sibling nodes at that level in the XML, multiple root nodes would be displayed in the tree.

A data provider for a tree always wants a node that has children it can display. It displays the first child of the XMLNode object. When the XML is wrapped in an XML object, the structure looks like the following:

```
<XMLDocumentObject>
  <node>
    <node label="Mail">
      <node label="INBOX"/>
      <node label="Personal Folder">
        <node label="Business" isBranch="true" />
        <node label="Demo" isBranch="true" />
        <node label="Personal" isBranch="true" />
        <node label="Saved Mail" isBranch="true" />
        <node label="bar" isBranch="true" />
      </node>
      <node label="Sent" isBranch="true" />
      <node label="Trash"/>
    </node>
  </node>
</XMLDocumentObject>
```

Flash Player wraps the XML nodes in an extra document node, which is passed to the tree. When the tree tries to display the XML, it tries to display `<node>`, which doesn't have a label, so it doesn't display correctly.

To avoid this problem, the data provider for the Tree component should point at the XMLDocumentObject's first child, as shown here:

```
myTree.dataProvider = myXML.firstChild;
```

Tree parameters

You can set the following authoring parameters for each Tree component instance in the Property inspector or in the Component inspector:

multipleSelection is a Boolean value that indicates whether a user can select multiple items (true) or not (false). The default value is false.

rowHeight indicates the height of each row, in pixels. The default value is 20.

You can write ActionScript to control these and additional options for the Tree component using its properties, methods, and events. For more information, see [“Tree class” on page 488](#).

You cannot enter data parameters in the Property inspector or in the Component inspector for the Tree component as you can with other components. For more information, see [“Using the Tree component” on page 478](#) and [“Creating an application with the Tree component” on page 480](#).

Creating an application with the Tree component

The following procedures show how to use a Tree component to display mailboxes in an e-mail application.

The Tree component does not allow you to enter data parameters in the Property inspector or Component inspector. Because of the complexity of a Tree component’s data structure, you must either import an XML object at runtime or build one in Flash while authoring. To create XML in Flash, you can use the TreeDataProvider interface, use the ActionScript XML object, or build an XML string. Each of these options is explained in the following procedures.

To add a Tree component to an application and load XML:

1. In Flash, select File > New and select Flash Document.
2. Save the document as **treeMenu fla**.
3. In the Components panel, double-click the Tree component to add it to the Stage.
4. Select the Tree instance. In the Property inspector, enter the instance name **menuTree**.
5. Select the Tree instance and press F8. Select Movie Clip, and enter the name **TreeNavMenu**.
6. Click the Advanced button, and select Export for ActionScript.
7. Type **TreeNavMenu** in the AS 2.0 Class text box and click OK.
8. Select File > New and select ActionScript File.
9. Save the file as **TreeNavMenu.as** in the same directory as **treeMenu fla**.
10. In the Script window, enter the following code:

```
import mx.controls.Tree;

class TreeNavMenu extends MovieClip {
    var menuXML:XML;
    var menuTree:Tree;
    function TreeNavMenu() {
        // Set up the appearance of the tree and event handlers
        menuTree.setStyle("fontFamily", "_sans");
        menuTree.setStyle("fontSize", 12);
        // Load the menu XML
        var treeNavMenu = this;
        menuXML = new XML();
        menuXML.ignoreWhite = true;
        menuXML.load("TreeNavMenu.xml");
        menuXML.onLoad = function() {
```

```

        treeNavMenu.onMenuLoaded();
    };
}
function change(event:Object) {
    if (menuTree == event.target) {
        var node = menuTree.selectedItem;
        // If this is a branch, expand/collapse it
        if (menuTree.getIsBranch(node)) {
            menuTree.setIsOpen(node, !menuTree.getIsOpen(node), true);
        }
        // If this is a hyperlink, jump to it
        var url = node.attributes.url;
        if (url) {
            getURL(url, "_top");
        }
        // Clear any selection
        menuTree.selectedNode = null;
    }
}
function onMenuLoaded() {
    menuTree.dataProvider = menuXML.firstChild;
    menuTree.addEventListener("change", this);
}
}

```

This ActionScript sets up styles for the tree. An XML object is created to load the XML file that creates the tree's nodes. Then the onLoad event handler is defined to set the data provider to the contents of the XML file.

11. Create a new file called `TreeNavMenu.xml` in a text editor.
12. Enter the following code in the file:

```

<node>
  <node label="My Bookmarks">
    <node label="Macromedia Web site" url="http://www.macromedia.com" />
    <node label="MXNA blog aggregator" url="http://www.markme.com/mxna" />
  </node>
  <node label="Google" url="http://www.google.com" />
</node>

```

13. Save your documents and return to `treeMenu.fla`. Select **Control > Test Movie** to test the application.

To load XML from an external file:

1. In Flash, select **File > New** and select **Flash Document**.
2. Drag an instance of the **Tree** component onto the Stage.
3. Select the **Tree** instance. In the **Property inspector**, enter the instance name **myTree**.
4. In the **Actions panel** on **Frame 1**, enter the following code:

```

var myTreeDP:XML = new XML();
myTreeDP.ignoreWhite = true;
myTreeDP.load("treeXML.xml");
myTreeDP.onLoad = function() {
    myTree.dataProvider = this.firstChild;
}

```

```

};
var treeListener:Object = new Object();
treeListener.change = function(evt:Object) {
    trace("selected node is: "+evt.target.selectedNode);
    trace("");
};
myTree.addEventListener("change", treeListener);

```

This code creates an XML object called `myTreeDP` and calls the `XML.load()` method to load an XML data source. The code then defines an `onLoad` event handler that sets the `dataProvider` property of the `myTree` instance to the new XML object when the XML loads. For more information about the XML object, see its entry in *Flash ActionScript Language Reference*.

5. Create a new file called **treeXML.xml** in a text editor.

6. Enter the following code in the file:

```

<node>
  <node label="Mail">
    <node label="INBOX"/>
    <node label="Personal Folder">
      <node label="Business" isBranch="true" />
      <node label="Demo" isBranch="true" />
      <node label="Personal" isBranch="true" />
      <node label="Saved Mail" isBranch="true" />
      <node label="bar" isBranch="true" />
    </node>
    <node label="Sent" isBranch="true" />
    <node label="Trash"/>
  </node>
</node>

```

7. Select **Control > Test Movie**.

In the SWF file, you can see the XML structure displayed in the tree. Click items in the tree to see the `trace()` statements in the `change` event handler send the data values to the Output panel.

To use the `TreeDataProvider` class to create XML in Flash while authoring:

1. In Flash, select **File > New** and select **Flash Document**.
2. Drag an instance of the **Tree** component onto the Stage.
3. Select the **Tree** instance and in the **Property inspector**, enter the instance name **myTree**.
4. In the **Actions panel on Frame 1**, enter the following code:

```

var myTreeDP:XML = new XML();
myTreeDP.addTreeNode("Local Folders", 0);
// Use XML.firstChild to nest child nodes below Local Folders
var myTreeNode:XMLNode = myTreeDP.firstChild;
myTreeNode.addTreeNode("Inbox", 1);
myTreeNode.addTreeNode("Outbox", 2);
myTreeNode.addTreeNode("Sent Items", 3);
myTreeNode.addTreeNode("Deleted Items", 4);
// Assign the myTreeDP data source to the myTree component
myTree.dataProvider = myTreeDP;
// Set each of the four child nodes to be branches

```

```

for (var i = 0; i<myTreeNode.childNodes.length; i++) {
    var node:XMLNode = myTreeNode.getTreeNodeAt(i);
    myTree.setIsBranch(node, true);
}

```

This code creates an XML object called `myTreeDP`. Any XML object on the same frame as a Tree component automatically receives all the properties and methods of the `TreeDataProvider` interface. The second line of code creates a single root node called Local Folders. For detailed information about the rest of the code, see the comments (lines preceded with `//`) throughout the code.

5. Select Control > Test Movie.

In the SWF file, you can see the XML structure displayed in the tree. Click items in the tree to see the `trace()` statements in the change event handler send the data values to the Output panel.

To use the **ActionScript XML** class to create XML:

1. In Flash, select File > New and select Flash Document.
2. Drag an instance of the Tree component onto the Stage.
3. Select the Tree instance. In the Property inspector, enter the instance name **myTree**.
4. In the Actions panel on Frame 1, enter the following code:

```

// Create an XML object
var myTreeDP:XML = new XML();
// Create node values
var myNode0:XMLNode = myTreeDP.createElement("node");
myNode0.attributes.label = "Local Folders";
myNode0.attributes.data = 0;
var myNode1:XMLNode = myTreeDP.createElement("node");
myNode1.attributes.label = "Inbox";
myNode1.attributes.data = 1;
var myNode2:XMLNode = myTreeDP.createElement("node");
myNode2.attributes.label = "Outbox";
myNode2.attributes.data = 2;
var myNode3:XMLNode = myTreeDP.createElement("node");
myNode3.attributes.label = "Sent Items";
myNode3.attributes.data = 3;
var myNode4:XMLNode = myTreeDP.createElement("node");
myNode4.attributes.label = "Deleted Items";
myNode4.attributes.data = 4;
// Assign nodes to the hierarchy in the XML tree
myTreeDP.appendChild(myNode0);
myTreeDP.firstChild.appendChild(myNode1);
myTreeDP.firstChild.appendChild(myNode2);
myTreeDP.firstChild.appendChild(myNode3);
myTreeDP.firstChild.appendChild(myNode4);
// Assign the myTreeDP data source to the Tree component
myTree.dataProvider = myTreeDP;

```

For more information about the XML object, see its entry in *Flash ActionScript Language Reference*.

5. Select Control > Test Movie.

In the SWF file, you can see the XML structure displayed in the tree. Click items in the tree to see the `trace()` statements in the `change` event handler send the data values to the Output panel.

To use a well-formed string to create XML in Flash while authoring:

1. In Flash, select File > New and select Flash Document.
2. Drag an instance of the Tree component onto the Stage.
3. Select the Tree instance. In the Property inspector, enter the instance name **myTree**.
4. In the Actions panel on Frame 1, enter the following code:

```
var myTreeDP:XML = new XML("<node label='Local Folders'><node label='Inbox' data='0' /><node label='Outbox' data='1' /></node>");
myTree.dataProvider = myTreeDP;
```

This code creates the XML object `myTreeDP` and assigns it to the `dataProvider` property of `myTree`.

5. Select Control > Test Movie.

In the SWF file, you can see the XML structure displayed in the tree. Click items in the tree to see the `trace()` statements in the `change` event handler send the data values to the Output panel.

Customizing the Tree component

You can transform a Tree component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)). When a tree isn't wide enough to display the text of the nodes, the text is clipped.

Using styles with the Tree component

A Tree component uses the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>backgroundColor</code>	Both	The background color of the list. The default color is white and is defined on the class style declaration. This style is ignored if <code>alternatingRowColors</code> is specified.
<code>backgroundDisabledColor</code>	Both	The background color when the component's <code>enabled</code> property is set to "false". The default value is 0xDDDDDD (medium gray).

Style	Theme	Description
<code>depthColors</code>	Both	Sets the background colors for rows based on the depth of each node. The value is an array of colors where the first element is the background color for the root node, the second element is the background color for its children, and so on, continuing through the number of colors provided in the array. This style property is not set by default.
<code>border styles</code>	Both	The Tree component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See “ <code>RectBorder</code> class” in Flash Help. The default border style is <code>"inset"</code> .
<code>color</code>	Both	The text color.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is <code>0x848384</code> (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is <code>"_sans"</code> .
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either <code>"normal"</code> or <code>"italic"</code> . The default value is <code>"normal"</code> .
<code>fontWeight</code>	Both	The font weight: either <code>"none"</code> or <code>"bold"</code> . The default value is <code>"none"</code> . All components can also accept the value <code>"normal"</code> in place of <code>"none"</code> during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return <code>"none"</code> .
<code>textAlign</code>	Both	The text alignment: either <code>"left"</code> , <code>"right"</code> , or <code>"center"</code> . The default value is <code>"left"</code> .
<code>textDecoration</code>	Both	The text decoration: either <code>"none"</code> or <code>"underline"</code> . The default value is <code>"none"</code> .
<code>textIndent</code>	Both	A number indicating the text indent. The default value is 0.
<code>defaultLeafIcon</code>	Both	The icon displayed in a Tree control for leaf nodes when no icon is specified for a particular node. The default value is <code>"TreeNodeIcon"</code> , which is an image representing a piece of paper.
<code>disclosureClosedIcon</code>	Both	The icon displayed next to a closed folder node in a Tree component. The default value is <code>"TreeDisclosureClosed"</code> , which is a gray arrow pointing to the right.
<code>disclosureOpenIcon</code>	Both	The icon displayed next to an open folder node in a Tree component. The default value is <code>"TreeDisclosureOpen"</code> , which is a gray arrow pointing down.

Style	Theme	Description
<code>folderClosedIcon</code>	Both	The icon displayed for a closed folder node in a Tree component if no node-specific icon is set. The default value is "TreeFolderClosed", which is a yellow closed file folder image.
<code>folderOpenIcon</code>	Both	The icon displayed for an open folder node in a Tree component if no node-specific icon is set. The default value is "TreeFolderOpen", which is a yellow open file folder image.
<code>indentation</code>	Both	The number of pixels to indent each row of a Tree component. The default value is 17.
<code>openDuration</code>	Both	The duration, in milliseconds, of the expand and collapse animations. The default value is 250.
<code>openEasing</code>	Both	A reference to a tweening function that controls the expand and collapse animations. Defaults to sine in/out. For more information, see "Customizing component animations" in Flash Help.
<code>repeatDelay</code>	Both	The number of milliseconds of delay between when a user first presses a button in the scrollbar and when the action begins to repeat. The default value is 500 (half a second).
<code>repeatInterval</code>	Both	The number of milliseconds between automatic clicks when a user holds the mouse button down on a button in the scrollbar. The default value is 35.
<code>rolloverColor</code>	Both	The background color of a rolled-over row. The default value is <code>OxE3FFD6</code> (bright green) with the Halo theme and <code>OxAaaaaa</code> (light gray) with the Sample theme. When <code>themeColor</code> is changed through a <code>setStyle()</code> call, the framework sets <code>rolloverColor</code> to a value related to the <code>themeColor</code> chosen.
<code>selectionColor</code>	Both	The background color of a selected row. The default value is a <code>OxCDFFC1</code> (light green) with the Halo theme and <code>OxEeeeeee</code> (very light gray) with the Sample theme. When <code>themeColor</code> is changed through a <code>setStyle()</code> call, the framework sets <code>selectionColor</code> to a value related to the <code>themeColor</code> chosen.
<code>selectionDuration</code>	Both	The length of the transition from a normal state to a selected state or back from selected to normal, in milliseconds. The default value is 200.
<code>selectionDisabledColor</code>	Both	The background color of a selected row. The default value is a <code>OxDDDDDD</code> (medium gray). Because the default value for this property is the same as the default for <code>backgroundDisabledColor</code> , the selection is not visible when the component is disabled unless one of these style properties is changed.

Style	Theme	Description
<code>selectionEasing</code>	Both	A reference to the easing equation used to control the transition between selection states. Applies only for the transition from a normal to a selected state. The default equation uses a sine in/out formula. For more information, see “Customizing component animations” in Flash Help.
<code>textRollOverColor</code>	Both	The color of text when the pointer rolls over it. The default value is <code>0x2B333C</code> (dark gray). This style is important when you set <code>rollOverColor</code> , because the two must complement each other so that text is easily viewable during a rollover.
<code>textSelectedColor</code>	Both	The color of text in the selected row. The default value is <code>0x005F33</code> (dark gray). This style is important when you set <code>selectionColor</code> , because the two must complement each other so that text is easily viewable while selected.
<code>useRollOver</code>	Both	Determines whether rolling over a row activates highlighting. The default value is <code>true</code> .

Setting styles for all Tree components in a document

The `Tree` class inherits from the `List` class, which inherits from the `ScrollSelectList` class. The default class-level style properties are defined on the `ScrollSelectList` class, which the `Menu` component and all `List`-based components extend. You can set new default style values on this class directly, and the new settings will be reflected in all affected components.

```
_global.styles.ScrollSelectList.setStyle("backgroundColor", 0xFF00AA);
```

To set a style property on the `Tree` components only, you can create a new `CSSStyleDeclaration` instance and store it in `_global.styles.DataGrid`.

```
import mx.styles.CSSStyleDeclaration;
if (_global.styles.Tree == undefined) {
    _global.styles.Tree = new CSSStyleDeclaration();
}
_global.styles.Tree.setStyle("backgroundColor", 0xFF00AA);
```

When creating a new class-level style declaration, you will lose all default values provided by the `ScrollSelectList` declaration. This includes `backgroundColor`, which is required for supporting mouse events. To create a class-level style declaration and preserve defaults, use a `for..in` loop to copy the old settings to the new declaration.

```
var source = _global.styles.ScrollSelectList;
var target = _global.styles.Tree;
for (var style in source) {
    target.setStyle(style, source.getStyle(style));
}
```

For more information about class-level styles, see “Setting styles for a component class” in Flash Help.

Using skins with the Tree component

The Tree component uses an instance of `RectBorder` for its border and scroll bars for scrolling images. For more information about skinning these visual elements, see “`RectBorder` class” in Flash Help and “[Using skins with the Tree component](#)” on page 488.

Tree class

Inheritance `MovieClip` > `UIObject class` > `UIComponent class` > `View` > `ScrollView` > `ScrollSelectList` > [List component](#) > `Tree`

ActionScript Class Name `mx.controls.Tree`

The methods, properties, and events of the `Tree` class allow you to manage and manipulate `Tree` objects.

Method summary for the Tree class

The following table lists methods of the `Tree` class.

Method	Description
<code>Tree.addNode()</code>	Adds a node to a <code>Tree</code> instance.
<code>Tree.addNodeAt()</code>	Adds a node at a specific location in a <code>Tree</code> instance.
<code>Tree.getDisplayIndex()</code>	Returns the display index of a given node.
<code>Tree.getIsBranch()</code>	Specifies whether the folder is a branch (has a folder icon and an expander arrow).
<code>Tree.getIsOpen()</code>	Indicates whether a node is open or closed.
<code>Tree.getNodeDisplayedAt()</code>	Maps a display index of the tree onto the node that is displayed there.
<code>Tree.getNodeAt()</code>	Returns a node on the root of the tree.
<code>Tree.refresh()</code>	Updates the tree.
<code>Tree.removeAll()</code>	Removes all nodes from a <code>Tree</code> instance and refreshes the tree.
<code>Tree.removeTreeNodeAt()</code>	Removes a node at a specified position and refreshes the tree.
<code>Tree.setIcon()</code>	Specifies an icon for the specified node.
<code>Tree.setIsBranch()</code>	Specifies whether a node is a branch (has a folder icon and expander arrow).
<code>Tree.setIsOpen()</code>	Opens or closes a node.

Methods inherited from the UIObject class

The following table lists the methods the Tree class inherits from the UIObject class. When calling these methods from the Tree object, use the form *TreeInstance.methodName*.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it will be redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the Tree class inherits from the UIComponent class. When calling these methods from the Tree object, use the form *TreeInstance.methodName*.

Method	Description
UIComponent.getFocus()	Returns a reference to the object that has focus.
UIComponent.setFocus()	Sets focus to the component instance.

Methods inherited from the List class

The following table lists the methods the Tree class inherits from the List class. When calling these methods from the Tree object, use the form *TreeInstance.methodName*.

Method	Description
List.addItem()	Adds an item to the end of the list.
List.addItemAt()	Adds an item to the list at the specified index. With the Tree component, it is better to use Tree.addTreeNodeAt() .
List.getItemAt()	Returns the item at the specified index.
List.removeAll()	Removes all items from the list.
List.removeItemAt()	Removes the item at the specified index.
List.replaceItemAt()	Replaces the item at the specified index with another item.

Method	Description
<code>List.setPropertiesAt()</code>	Applies the specified properties to the specified item.
<code>List.sortItems()</code>	Sorts the items in the list according to the specified compare function.
<code>List.sortItemsBy()</code>	Sorts the items in the list according to a specified property.

Property summary for the Tree class

The following table lists properties of the Tree class.

Property	Description
<code>Tree.dataProvider</code>	Specifies an XML data source.
<code>Tree.firstVisibleNode</code>	Specifies the first node at the top of the display.
<code>Tree.selectedNode</code>	Specifies the selected node in a Tree instance.
<code>Tree.selectedNodes</code>	Specifies the selected nodes in a Tree instance.

Properties inherited from the UIObject class

The following table lists the properties the Tree class inherits from the UIObject class. When accessing these properties from the Tree object, use the form *TreeInstance.propertyName*.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `Tree` class inherits from the `UIComponent` class. When accessing these properties from the `Tree` object, use the form `TreeInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Properties inherited from the `List` class

The following table lists the properties the `Tree` class inherits from the `List` class. When accessing these properties from the `Tree` object, use the form `TreeInstance.propertyName`.

Property	Description
<code>List.cellRenderer</code>	Assigns the class or symbol to use to display each row of the list.
<code>List.dataProvider</code>	The source of the list items.
<code>List.hPosition</code>	The horizontal position of the list.
<code>List.hScrollPolicy</code>	Indicates whether the horizontal scroll bar is displayed ("on") or not ("off").
<code>List.iconField</code>	A field in each item to be used to specify icons.
<code>List.iconFunction</code>	A function that determines which icon to use.
<code>List.labelField</code>	Specifies a field of each item to be used as label text.
<code>List.labelFunction</code>	A function that determines which fields of each item to use for the label text.
<code>List.length</code>	The number of items in the list. This property is read-only.
<code>List.maxHPosition</code>	The number of pixels the list can scroll to the right, when <code>List.hScrollPolicy</code> is set to "on".
<code>List.multipleSelection</code>	Indicates whether multiple selection is allowed in the list (<code>true</code>) or not (<code>false</code>).
<code>List.rowCount</code>	The number of rows that are at least partially visible in the list.
<code>List.rowHeight</code>	The pixel height of every row in the list.
<code>List.selectable</code>	Indicates whether the list is selectable (<code>true</code>) or not (<code>false</code>).
<code>List.selectedIndex</code>	The index of a selection in a single-selection list.
<code>List.selectedIndices</code>	An array of the selected items in a multiple-selection list.
<code>List.selectedItem</code>	The selected item in a single-selection list. This property is read-only.
<code>List.selectedItems</code>	The selected item objects in a multiple-selection list. This property is read-only.

Property	Description
<code>List.vPosition</code>	Scrolls the list so the topmost visible item is the number assigned.
<code>List.vScrollPolicy</code>	Indicates whether the vertical scroll bar is displayed ("on"), not displayed ("off"), or displayed when needed ("auto").

Event summary for the Tree class

The following table lists events of the Tree class.

Event	Description
<code>Tree.nodeClose</code>	Broadcast when a node is closed by a user.
<code>Tree.nodeOpen</code>	Broadcast when a node is opened by a user.

Events inherited from the UIObject class

The following table lists the events the Tree class inherits from the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the Tree class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Events inherited from the List class

The following table lists the events the Tree class inherits from the List class.

Event	Description
List.change	Broadcast whenever user interaction causes the selection to change.
List.itemRollOut	Broadcast when the pointer rolls over and then off of list items.
List.itemRollOver	Broadcast when the pointer rolls over list items.
List.scroll	Broadcast when a list is scrolled.

Tree.addNode()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
myTree.addNode(label [, data])
```

Usage 2:

```
myTree.addNode(child)
```

Parameters

label A string that displays the node, or an object with a label field (or whatever label field name is specified by the `labelField` property).

data An object of any type that is associated with the node. This parameter is optional.

child Any XMLNode object.

Returns

The added XML node.

Description

Method; adds a child node to the tree. The node is constructed either from the information supplied in the *label* and *data* parameters (Usage 1), or from the prebuilt child node, which is an XMLNode object (Usage 2). Adding a preexisting node removes the node from its previous location.

Calling this method refreshes the view.

Example

The following code adds a new node to the root of `myTree`. The second line of code moves a node from the root of `mySecondTree` to the root of `myTree`.

```
myTree.addTreeNode("Inbox", 3);  
myTree.addTreeNode(mySecondTree.getTreeNodeAt(3));
```

Tree.addTreeNodeAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

Usage 1:

```
myTree.addTreeNodeAt(index, label [, data])
```

Usage 2:

```
myTree.addTreeNodeAt(index, child)
```

Parameters

index The zero-based index position (among the child nodes) at which the node should be added.

label A string that displays the node.

data An object of any type that is associated with the node. This parameter is optional.

child Any `XMLNode` object.

Returns

The added XML node.

Description

Method; adds a node at the specified location in the tree. The node is constructed either from the information supplied in the *label* and *data* parameters (Usage 1), or from the prebuilt `XMLNode` object (Usage 2). Adding a preexisting node removes the node from its previous location.

Calling this method refreshes the view.

Example

The following example adds a new node as the second child of the root of `myTree`. The second line moves a node from `mySecondTree` to become the fourth child of the root of `myTree`:

```
myTree.addTreeNodeAt(1, "Inbox", 3);  
myTree.addTreeNodeAt(3, mySecondTree.getTreeNodeAt(3));
```

Tree.dataProvider

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myTree.dataProvider
```

Description

Property; either XML or a string. If `dataProvider` is an XML object, it is added directly to the tree. If `dataProvider` is a string, it must contain valid XML that is read by the tree and converted to an XML object.

You can either load XML from an external source at runtime or create it in Flash while authoring. To create XML, you can use either the `TreeDataProvider` methods, or the built-in `ActionScript XML` class methods and properties. You can also create a string that contains XML.

XML objects that are on the same frame as a `Tree` component automatically contain the `TreeDataProvider` methods and properties. You can use the `ActionScript XML` or `XMLNode` object.

Example

The following example imports an XML file and assigns it to the `myTree` instance of the `Tree` component:

```
myTreeDP = new XML();
myTreeDP.ignoreWhite = true;
myTreeDP.load("http://myServer.myDomain.com/source.xml");
myTreeDP.onLoad = function(){
    myTree.dataProvider = myTreeDP;
}
```

Note: Most XML files contain white space. To make Flash ignore white space, you must set the `XML.ignoreWhite` property to `true`.

Tree.firstVisibleNode

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myTree.firstVisibleNode = someNode
```

Description

Property; indicates the first node that is visible at the top of the tree display. Use this property to scroll the tree display to a desired position. If the specified node *someNode* is within a node that hasn't been expanded, setting `firstVisibleNode` has no effect. The default value is the first visible node or `undefined` if there is no visible node. The value of this property is an `XMLNode` object.

This property is an analogue to the `List.vPosition` property.

Example

The following example sets the scroll position to the top of the display:

```
myTree.firstVisibleNode = myTree.getTreeNodeAt(0);
```

Tree.getDisplayIndex()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myTree.getDisplayIndex(node)
```

Parameters

node An `XMLNode` object.

Returns

The index of the specified node, or `undefined` if the node is not currently displayed.

Description

Method; returns the display index of the node specified in the *node* parameter.

The display index indicates the item's position in the list of items that are visible in the tree window. For example, any children of a closed node are not in the display index. The list of display indices starts with 0 and proceeds through the visible items regardless of parent. In other words, the index is the row number, starting with 0, of the displayed rows.

Example

The following code gets the display index of `myNode`:

```
var x = myTree.getDisplayIndex(myNode);
```

Tree.getIsBranch()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myTree.getIsBranch(node)
```

Parameters

node An XMLNode object.

Returns

A Boolean value that indicates whether the node is a branch (`true`) or not (`false`).

Description

Method; indicates whether the specified node has a folder icon and expander arrow (and is therefore a branch). This is set automatically when children are added to the node. You only need to call [Tree.setIsBranch\(\)](#) to create empty folders.

Example

The following code assigns the node state to a variable:

```
var open = myTree.getIsBranch(myTree.getTreeNodeAt(1));
```

See also

[Tree.setIsBranch\(\)](#)

Tree.getIsOpen()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myTree.getIsOpen(node)
```

Parameters

node An XMLNode object.

Returns

A Boolean value that indicates whether the tree is open (`true`) or closed (`false`).

Description

Method; indicates whether the specified node is open or closed.

Note: Display indices change every time nodes open and close.

Example

The following code assigns the state of the node to a variable:

```
var open = myTree.getIsOpen(myTree.getTreeNodeAt(1));
```

Tree.getNodeDisplayedAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myTree.getNodeDisplayedAt(index)
```

Parameters

index An integer representing the display position in the viewable area of the tree. This number is zero-based; the node at the first position is 0, second position is 1, and so on.

Returns

The specified XMLNode object.

Description

Method; maps a display index of the tree onto the node that is displayed there. For example, if the fifth row of the tree showed a node that is eight levels deep into the hierarchy, that node would be returned by a call to `getNodeDisplayedAt(4)`.

The display index is an array of items that can be viewed in the tree window. For example, any children of a closed node are not in the display index. The display index starts with 0 and proceeds through the visible items regardless of parent. In other words, the display index is the row number, starting with 0, of the displayed rows.

Note: Display indices change every time nodes open and close.

Example

The following code gets a reference to the XML node that is the second row displayed in `myTree`:

```
myTree.getNodeDisplayedAt(1);
```

Tree.getTreeNodeAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myTree.getTreeNodeAt(index)
```

Parameters

index The index number of a node.

Returns

An XMLNode object.

Description

Method; returns the specified node on the root of myTree.

Example

The following code gets the second node on the first level in the tree myTree:

```
myTree.getTreeNodeAt(1);
```

Tree.nodeClose

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();  
listenerObject.nodeClose = function(eventObject){  
    // insert your code here  
}  
myTreeInstance.addEventListener("nodeClose", listenerObject)
```

Description

Event; broadcast to all registered listeners when the nodes of a Tree component are closed by a user.

Version 2 components use a dispatcher/listener event model. The Tree component broadcasts a `nodeClose` event when one of its nodes is clicked closed; the event is handled by a function, also called a *handler*, that is attached to a listener object (*listenerObject*) that you create.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Tree.nodeClose` event's event object has one additional property: `node` (the XML node that closed).

For more information, see “EventDispatcher class” in Flash Help.

Example

In the following example, a handler called `myTreeListener` is defined and passed to the `myTree.addEventListener()` method as the second parameter. The event object is captured by the `nodeClose` handler in the `evtObject` parameter. When the `nodeClose` event is broadcast, a trace statement is sent to the Output panel.

```
myTreeListener = new Object();
myTreeListener.nodeClose = function(evtObject){
    trace(evtObject.node + " node was closed");
}
myTree.addEventListener("nodeClose", myTreeListener);
```

Tree.nodeOpen

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.nodeOpen = function(eventObject){
    // insert your code here
}
myTreeInstance.addEventListener("nodeOpen", listenerObject)
```

Description

Event; broadcast to all registered listeners when a user opens a node on a Tree component.

Version 2 components use a dispatcher/listener event model. The Tree component dispatches a `nodeOpen` event when a node is clicked open by a user; the event is handled by a function, also called a *handler*, that is attached to a listener object (*listenerObject*) that you create. You call the `addEventListener()` method and pass it the name of the handler as a parameter.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. The `Tree.nodeOpen` event's event object has one additional property: `node` (the XML node that was opened).

For more information, see “EventDispatcher class” in Flash Help.

Example

In the following example, a handler called `myTreeListener` is defined and passed to `myTree.addEventListener()` as the second parameter. The event object is captured by the `nodeOpen` handler in the `evtObject` parameter. When the `nodeOpen` event is broadcast, a trace statement is sent to the Output panel.

```
myTreeListener = new Object();
myTreeListener.nodeOpen = function(evtObject){
    trace(evtObject.node + " node was opened");
}
myTree.addEventListener("nodeOpen", myTreeListener);
```

Tree.refresh()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myTree.refresh()
```

Parameters

None.

Returns

Nothing.

Description

Method; updates the tree.

Tree.removeAll()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myTree.removeAll()
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all nodes and refreshes the tree.

Example

The following code empties `myTree`:

```
myTree.removeAll();
```

Tree.removeTreeNodeAt()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myTree.removeTreeNodeAt(index)
```

Parameters

index The index number of a tree child. Each child of a tree is assigned a zero-based index in the order in which it was created.

Returns

An `XMLNode` object, or `undefined` if an error occurs.

Description

Method; removes a node (specified by its index position) on the root of the tree and refreshes the tree.

Example

The following code removes the fourth child of the root of the tree `myTree`:

```
myTree.removeTreeNodeAt(3);
```

Tree.selectedNode

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myTree.selectedNode
```

Description

Property; specifies the selected node in a tree instance.

Example

The following example specifies the first child node in `myTree`:

```
myTree.selectedNode = myTree.getTreeNodeAt(0);
```

See also

[Tree.selectedNodes](#)

Tree.selectedNodes

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myTree.selectedNodes
```

Description

Property; specifies the selected nodes in a tree instance.

Example

The following example selects the first and third child nodes in `myTree`:

```
myTree.selectedNodes = [myTree.getTreeNodeAt(0), myTree.getTreeNodeAt(2)];
```

See also

[Tree.selectedNode](#)

Tree.setIcon()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myTree.setIcon(node, linkID [, linkID2])
```

Parameters

node An XML node.

linkID The linkage identifier of a symbol to be used as an icon beside the node. This parameter is used for leaf nodes and for the closed state of branch nodes.

linkID2 For a branch node, the linkage identifier of a symbol to be used as an icon that represents the open state of the node. This parameter is optional.

Returns

Nothing.

Description

Method; specifies an icon for the specified node. This method takes one ID parameter (*linkID*) for leaf nodes and two ID parameters (*linkID* and *linkID2*) for branch nodes (the closed and open icons). For leaf nodes, the second parameter is ignored. For branch nodes, if you omit *linkID2*, the icon is used for both the closed and open states.

Example

The following code specifies that a symbol with the linkage identifier `imageIcon` be used beside the second node of `myTree`:

```
myTree.setIcon(myTree.getTreeNodeAt(1), "imageIcon");
```

Tree.setIsBranch()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
myTree.setIsBranch(node, isBranch)
```

Parameters

node An XML node.

isBranch A Boolean value indicating whether the node is (`true`) or is not (`false`) a branch.

Returns

Nothing.

Description

Method; specifies whether the node has a folder icon and expander arrow and either has children or can have children. A node is automatically set as a branch when it has children; you only need to call `setIsBranch()` when you want to create an empty folder. You may want to create branches that don't yet have children if, for example, you only want child nodes to load when a user opens a folder.

Calling `setIsBranch()` refreshes any views.

Example

The following code makes a node of `myTree` a branch:

```
myTree.setIsBranch(myTree.getTreeNodeAt(1), true);
```

Tree.setIsOpen()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX Professional 2004.

Usage

```
myTree.setIsOpen(node, open [, animate])
```

Parameters

node An XML node.

open A Boolean value that opens a node (`true`) or closes it (`false`).

animate A Boolean value that determines whether the opening transition is animated (`true`) or not (`false`). This parameter is optional.

Returns

Nothing.

Description

Method; opens or closes a node.

Example

The following code opens a node of `myTree`:

```
myTree.setIsOpen(myTree.getTreeNodeAt(1), true);
```

Window component

A Window component displays the contents of a movie clip inside a window with a title bar, a border, and an optional close button.

A Window component can be modal or nonmodal. A modal window prevents mouse and keyboard input from going to other components outside the window. The Window component also supports dragging; a user can click the title bar and drag the window and its contents to another location. Dragging the borders doesn't resize the window.

A live preview of each Window instance reflects changes made to all parameters except `contentPath` in the Property inspector or Component inspector during authoring.

When you add the Window component to an application, you can use the Accessibility panel to make it accessible to screen readers. First, you must add the following line of code to enable accessibility:

```
mx.accessibility.WindowAccImpl.enableAccessibility();
```

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

Using the Window component

You can use a window in an application whenever you need to present a user with information or a choice that takes precedence over anything else in the application. For example, you might need a user to fill out a login window, or a window that changes and confirms a new password.

There are several ways to add a window to an application. You can drag a window from the Components panel to the Stage. You can also call `createClassObject()` (see [UIObject.createClassObject\(\)](#)) to add a window to an application. The third way of adding a window to an application is to use the `PopUpManager` class. Use the PopUp Manager to create modal windows that overlap other objects on the Stage. For more information, see [“Window class” on page 510](#).

If you use the PopUp Manager to add a Window component to a document, the Window instance will have its own Focus Manager, distinct from the rest of the document. If you don't use the PopUp Manager, the window's contents participate in focus ordering with the other components in the document. For more information about controlling focus, see [“Creating custom focus navigation” in Flash Help](#) or [“FocusManager class” on page 231](#).

Window parameters

You can set the following authoring parameters for each Window component instance in the Property inspector or in the Component inspector:

contentPath specifies the contents of the window. This can be the linkage identifier of the movie clip or the symbol name of a screen, form, or slide that contains the contents of the window. This can also be an absolute or relative URL for a SWF or JPEG file to load into the window. The default value is `""`. Loaded content is clipped to fit the window.

title indicates the title of the window.

closeButton indicates whether a close button is displayed (`true`) or not (`false`). Clicking the close button broadcasts a `click` event, but doesn't close the window. You must write a handler that calls `Window.deletePopUp()` to explicitly close the window. For more information about the `click` event, see [Window.click](#).

Note: If a window was created by means other than the PopUp Manager, you can't close it.

You can write ActionScript to control these and additional options for the Window component using its properties, methods, and events. For more information, see [“Window class” on page 510](#).

Creating an application with the Window component

The following procedure explains how to add a Window component to an application. In this example, the window asks a user to change her password and confirm the new password.

To create an application with the Window component:

1. Create a new movie clip that contains password and password confirmation fields, and OK and Cancel buttons. Name the movie clip **PasswordForm**.

This is the content that will fill the window. The content should be aligned at 0,0 because it is positioned in the upper left corner of the window.

2. In the library, select the PasswordForm movie clip and select Linkage from the Library options menu.
3. Select Export for ActionScript.

The linkage identifier **PasswordForm** is automatically entered in the Identifier box.

4. Enter **mx.core.View** in the class field and click OK.
5. Drag a Window component from the Components panel to the Stage and delete the component from the Stage. This adds the component to the library.
6. In the library, select the Window SWC file and select Linkage from the Library options menu.
7. Select Export for ActionScript.
8. Drag a button component from the Components panel to the Stage; in the Property inspector, give it the instance name **button**.
9. Open the Actions panel, and enter the following click handler on Frame 1:

```
buttonListener = new Object();
buttonListener.click = function(){
    myWindow = mx.managers.PopUpManager.createPopUp(_root,
        mx.containers.Window, true, {title:"Change Password",
        contentPath:"PasswordForm"});
    myWindow.setSize(240,110);
}
button.addEventListener("click", buttonListener);
```

This handler calls `PopUpManager.createPopUp()` to instantiate a Window component with the title bar “Change Password”; the window displays the contents of the PasswordForm movie clip when the button is clicked. To close the window when the OK or Cancel button is clicked, you must write another handler.

Customizing the Window component

You can transform a Window component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use `UIObject.setSize()`.

Resizing the window does not change the size of the close button or title caption. The title caption is aligned to the left and the close bar to the right.

Using styles with the Window component

A Window component supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>backgroundColor</code>	Both	The background color. The default value is white for the Halo theme and OxEFEBEF (light gray) for the Sample theme.
<i>border styles</i>	Both	The Window component uses a <code>RectBorder</code> instance as its border and responds to the styles defined on that class. See "RectBorder class" in Flash Help. The Window component has a component-specific border style of "default" with the Halo theme and "outset" with the Sample theme.
<code>color</code>	Both	The text color. The default value is OxB3333C for the Halo theme and blank for the Sample theme.
<code>disabledColor</code>	Both	The color for text when the component is disabled. The default color is OxB48384 (dark gray).
<code>embedFonts</code>	Both	A Boolean value that indicates whether the font specified in <code>fontFamily</code> is an embedded font. This style must be set to <code>true</code> if <code>fontFamily</code> refers to an embedded font. Otherwise, the embedded font will not be used. If this style is set to <code>true</code> and <code>fontFamily</code> does not refer to an embedded font, no text will be displayed. The default value is <code>false</code> .
<code>fontFamily</code>	Both	The font name for text. The default value is "_sans".
<code>fontSize</code>	Both	The point size for the font. The default value is 10.
<code>fontStyle</code>	Both	The font style: either "normal" or "italic". The default value is "normal".
<code>fontWeight</code>	Both	The font weight: either "none" or "bold". The default value is "none". All components can also accept the value "normal" in place of "none" during a <code>setStyle()</code> call, but subsequent calls to <code>getStyle()</code> will return "none".
<code>textAlign</code>	Both	The text alignment: either "left", "right", or "center". The default value is "left".
<code>textDecoration</code>	Both	The text decoration: either "none" or "underline". The default value is "none".
<code>textIndent</code>	Both	A number indicating the text indent. The default value is 0.

Text styles can be set on the Window component itself, or they can be set on the `_global.styles.windowStyles` class style declaration. This has the advantage of not causing style settings to propagate to child components through style inheritance.

The following example demonstrates how to italicize the title of a Window component without having this setting propagate to child components.

```
import mx.containers.Window;
_global.styles.windowStyles.setStyle("fontStyle", "italic");
createClassObject(Window, "window", 1, {title: "A Window"});
```

Notice that this example sets the property before instantiating the window through `createClassObject()`. For the styles to take effect, they must be set before the window is created.

Using skins with the Window component

The Window component uses skins for its title background and close button, and a `RectBorder` instance for the border. The Window skins are found in the Flash UI Components 2/Themes/MMDefault/Window Assets folder in each of the theme files. For more information about skinning, see “About skinning components” in Flash Help. For more information about the `RectBorder` class and using it to customize borders, see “RectBorder class” in Flash Help.

The title background skin is always displayed. The height of the background is determined by the skin graphics. The width of the skin is set by the Window component according to the Window instance’s size. The close skins are displayed when the `closeButton` property is set to `true` and when a change state results from user interaction.

A Window component uses the following skin properties:

Property	Description
<code>skinTitleBackground</code>	The title bar. The default value is <code>TitleBackground</code> .
<code>skinCloseUp</code>	The close button. The default value is <code>CloseButtonUp</code> .
<code>skinCloseDown</code>	The close button in its down state. The default value is <code>CloseButtonDown</code> .
<code>skinCloseDisabled</code>	The close button in its disabled state. The default value is <code>CloseButtonDisabled</code> .
<code>skinCloseOver</code>	The close button in its over state. The default value is <code>CloseButtonOver</code> .

The following example demonstrates how to create a new movie clip symbol to use as the title background.

To set the title of an Window component to a custom movie clip symbol:

1. Create a new FLA file.
2. Create a new symbol by selecting **Insert > New Symbol**.
3. Set the name to `TitleBackground`.
4. If the advanced view is not displayed, click the **Advanced** button.
5. Select **Export for ActionScript**.
6. The identifier will be automatically filled out with `TitleBackground`.

7. Set the AS 2.0 class to `mx.skins.SkinElement`.

`SkinElement` is a simple class that can be used for all skin elements that don't provide their own ActionScript implementation. It provides movement and sizing functionality required by the version 2 component framework.

8. Ensure that Export in First Frame is already selected, and click OK.

9. Open the new symbol for editing.

10. Use the drawing tools to create a box with a red fill and black line.

11. Set the border style to hairline.

12. Set the box, including the border, so that it is positioned at (0,0) and has a width of 100 and height of 22.

The Window component will set the proper width of the skin as needed but it will use the existing height as the height of the title.

13. Click the Back button to return to the main Timeline.

14. Drag the Window component to the Stage.

15. Select Control > Test Movie.

Window class

Inheritance `MovieClip` > `UIObject class` > `UIComponent class` > View > `ScrollView` > Window

ActionScript Class Name `mx.containers.Window`

The properties of the Window class let you do the following at runtime: set the title caption, add a close button, and set the display content. Setting a property of the Window class with ActionScript overrides the parameter of the same name set in the Property inspector or Component Inspector panel.

The best way to instantiate a window is to call `PopUpManager.createPopUp()`. This method creates a window that can be modal (overlapping and disabling existing objects in an application) or nonmodal. For example, the following code creates a modal Window instance (the last parameter indicates modality):

```
var newWindow = PopUpManager.createPopUp(this, Window, true);
```

Flash simulates modality by creating a large transparent window underneath the Window component. Because of the way transparent windows are rendered, you may notice a slight dimming of the objects under the transparent window. You can set the effective transparency by changing the `_global.style.modalTransparency` value from 0 (fully transparent) to 100 (opaque). If you make the window partially transparent, you can also set the color of the window by changing the Modal skin in the default theme.

If you use `PopUpManager.createPopUp()` to create a modal window, you must call `Window.deletePopUp()` to remove it so that the transparent window is also removed. For example, if you use the close button in the window, you would write the following code:

```
obj.click = function(evt){
    this.deletePopUp();
}
window.addEventListener("click", obj);
```

Note: Code does not stop executing when a modal window is created. In other environments (for example, Microsoft Windows), if you create a modal window, the lines of code that follow the creation of the window do not run until the window is closed. In Flash, the lines of code are run after the window is created and before it is closed.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.containers.Window.version);
```

Note: The code `trace(myWindowInstance.version);` returns `undefined`.

Method summary for the Window class

The following table lists the method of the Window class.

Method	Description
<code>Window.deletePopUp()</code>	Removes a window instance created by <code>PopUpManager.createPopUp()</code> .

Methods inherited from the UIObject class

The following table lists the methods the Window class inherits from the UIObject class. When calling these methods from the Window object, use the form `WindowInstance.methodName`.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it will be redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the `UIComponent` class

The following table lists the methods the `Window` class inherits from the `UIComponent` class. When calling these methods from the `Window` object, use the form *WindowInstance.methodName*.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the `Window` class

The following table lists properties of the `Window` class.

Property	Description
<code>Window.closeButton</code>	Indicates whether a close button is (<code>true</code>) or is not (<code>false</code>) included on the title bar.
<code>Window.content</code>	A reference to the content (root movie clip) of the window.
<code>Window.contentPath</code>	Sets the name of the content to display in the window.
<code>Window.title</code>	The text that appears in the title bar.
<code>Window.titleStyleDeclaration</code>	The style declaration that formats the text in the title bar.

Properties inherited from the `UIObject` class

The following table lists the properties the `Window` class inherits from the `UIObject` class. When accessing these properties from the `Window` object, use the form *WindowInstance.propertyName*.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).

Property	Description
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `Window` class inherits from the `UIComponent` class. When accessing these properties from the `Window` object, use the form `WindowInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the `Window` class

The following table lists the events of the `Window` class.

Event	Description
<code>Window.click</code>	Broadcast when the close button is clicked (released).
<code>Window.complete</code>	Broadcast when a window is created.
<code>Window.mouseDownOutside</code>	Broadcast when the mouse is clicked (released) outside the modal window.

Events inherited from the `UIObject` class

The following table lists the events the `Window` class inherits from the `UIObject` class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the UIComponent class

The following table lists the events the Window class inherits from the UIComponent class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

Window.click

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(click){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.click = function(eventObject){  
    ...  
}  
windowInstance.addEventListener("click", listenerObject)
```

Description

Event; broadcast to all registered listeners when the mouse is clicked (released) over the close button.

The first usage example uses an `on()` handler and must be attached directly to a Window instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the Window instance `myWindow`, sends “_level0.myWindow” to the Output panel:

```
on(click){  
    trace(this);  
}
```

The second usage example uses a dispatcher/listener event model. A component instance (*windowInstance*) dispatches an event (in this case, `click`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following example creates a modal window and then defines a click handler that deletes the window. You must add a `Window` component to the Stage and then delete it to add the component to the document library; then add the following code to Frame 1:

```
import mx.managers.PopUpManager
import mx.containers.Window
var myTW = PopUpManager.createPopUp(_root, Window, true, {closeButton: true,
    title:"My Window"});
windowListener = new Object();
windowListener.click = function(evt){
    _root.myTW.deletePopUp();
}
myTW.addEventListener("click", windowListener);
```

See also

`EventDispatcher.addEventListener()` in Flash Help, [Window.closeButton](#)

Window.closeButton

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

windowInstance.closeButton

Description

Property; a Boolean value that indicates whether the title bar should have a close button (`true`) or not (`false`). This property must be set in the *initObject* parameter of the `PopUpManager.createPopUp()` method. The default value is `false`.

Clicking the close button broadcasts a `click` event, but doesn't close the window. You must write a handler that calls `Window.deletePopUp()` to explicitly close the window. For more information about the `click` event, see [Window.click](#).

Example

The following code creates a window that displays the content in the movie clip “LoginForm” and has a close button on the title bar:

```
var myTW = PopUpManager.createPopUp(_root, Window, true,
    {contentPath:"LoginForm", closeButton:true});
```

See also

`PopUpManager.createPopUp()` in [Flash Help](#), [Window.click](#)

Window.complete

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
listenerObject = new Object();
listenerObject.complete = function(eventObject){
    ...
}
windowInstance.addEventListener("complete", listenerObject)
```

Description

Event; broadcast to all registered listeners when a window is created. Use this event to size a window to fit its contents.

A component instance (*windowInstance*) dispatches an event (in this case, *complete*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following example creates a window and then defines a *complete* handler that resizes the window to fit its contents. (This code would be placed on Frame 1 of the component in the library.)

```
import mx.managers.PopUpManager
import mx.containers.Window
var myTW = PopUpManager.createPopUp(_root, Window, true, {title:"Password
    Change", contentPath: "PasswordForm"});
```

```
lo = new Object();
lo.handleEvent = function(evtObj){
    if(evtObj.type == "complete"){
        _root.myTW.setSize(myTW.content._width, myTW.content._height + 25);
    }
}
myTW.addEventListener("complete", lo);
```

See also

[EventDispatcher.addEventListener\(\)](#) in Flash Help

Window.content

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

windowInstance.content

Description

Property; a reference to the content (root movie clip) of the window. This property returns a MovieClip object. When you attach a symbol from the library, the default value is an instance of the attached symbol. When you load content from a URL, the default value is *undefined* until the load operation has started.

Example

The following code sets the value of the text property within the content inside the Window component:

```
myLoginWindow.content.password.text = "secret";
```

Window.contentPath

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

windowInstance.contentPath

Description

Property; sets the name of the content to display in the window. This value can be the linkage identifier of a movie clip in the library, or the absolute or relative URL of a SWF or JPEG file to load. The default value is "" (an empty string).

Example

The following code creates a Window instance that displays the movie clip with the linkage identifier “LoginForm”:

```
var myTW = PopUpManager.createPopUp(_root, Window, true,
    {contentPath:"LoginForm"});
```

Window.deletePopUp()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
windowInstance.deletePopUp()
```

Parameters

None.

Returns

Nothing.

Description

Method; deletes the window instance and removes the modal state. This method can be called only on Window instances that were created by `PopUpManager.createPopUp()`.

Example

The following code creates a modal window, then creates a listener that deletes the window when the close button is clicked:

```
import mx.managers.PopUpManager;
import mx.containers.Window;

var myTW:MovieClip = PopUpManager.createPopUp(_root, Window, true,
    {closeButton:true, title:"Test"});
var twListener:Object = new Object();
twListener.click = function(evt:Object){
    evt.target.deletePopUp();
}
myTW.addEventListener("click", twListener);
```

Note: Remember to add a Window component to the Stage then delete it to add it to the Library before running the above code.

Window.mouseDownOutside

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(mouseDownOutside){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.mouseDownOutside = function(eventObject){  
    ...  
}  
windowInstance.addEventListener("mouseDownOutside", listenerObject)
```

Description

Event; broadcast to all registered listeners when the mouse is clicked (released) outside the modal window. This event is rarely used, but you can use it to dismiss a window if the user tries to interact with something outside of it.

The first usage example uses an `on()` handler and must be attached directly to a Window instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the Window instance `myWindowComponent`, sends “_level0.myWindowComponent” to the Output panel:

```
on(mouseDownOutside){  
    trace(this);  
}
```

The second usage example uses a dispatcher/listener event model. A component instance (`windowInstance`) dispatches an event (in this case, `mouseDownOutside`) and the event is handled by a function, also called a *handler*, on a listener object (`listenerObject`) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (`eventObject`) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following example creates a window instance and defines a `mouseDownOutside` handler that calls a `beep()` method if the user clicks outside the window:

```
var myTW = PopUpManager.createPopUp(_root, Window, true, undefined, true);
// create a listener
twListener = new Object();
twListener.mouseDownOutside = function()
{
    beep(); // make a noise if user clicks outside
}
myTW.addEventListener("mouseDownOutside", twListener);
```

See also

`EventDispatcher.addEventListener()` in [Flash Help](#)

Window.title

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

windowInstance.title

Description

Property; a string indicating the text of the title bar. The default value is "" (an empty string).

Example

The following code sets the title of the window to “Hello World”:

```
myTW.title = "Hello World";
```

Window.titleStyleDeclaration

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

windowInstance.titleStyleDeclaration

Description

Property; a string indicating the style declaration that formats the title bar of a window. The default value is `undefined`, which indicates bold, white text.

Example

With a Window component in the library, use the following ActionScript to format the style of the window's title bar.

```
// create a new CSSStyleDeclaration named TitleStyles
// and list it with the global styles list
_global.styles.TitleStyles = new mx.styles.CSSStyleDeclaration();
// customize styles
_global.styles.TitleStyles.fontStyle = "italic";
_global.styles.TitleStyles.textDecoration = "underline";
_global.styles.TitleStyles.color = 0xff0000;
_global.styles.TitleStyles.fontSize = 14;

tw = mx.managers.PopUpManager.createPopUp(this, mx.containers.Window, true,
    {closeButton:true, titleStyleDeclaration:"TitleStyles"});
tw.title = "Testing Styles";
tw.setSize(200, 100);
tw.move(20, 20);
var handleCloseObject:Object = new Object();
handleCloseObject.click = function(evt:Object) {
    evt.target.deletePopUp();
};
tw.addEventListener("click", handleCloseObject);
```

For more information about styles, see “Using styles to customize component color and text” in Flash Help.

UIComponent class

The UIComponent class does not represent a visual component; it contains methods, properties, and events that allow Macromedia components to share some common behavior. All version 2 components extend UIComponent. The UIComponent class lets you do the following:

- Receive focus and keyboard input
- Enable and disable components
- Resize by layout

To use the methods and properties of UIComponent, you call them directly from whichever component you are using. For example, to call `UIComponent.setFocus()` from the `RadioButton` component, you would write the following code:

```
myRadioButton.setFocus();
```

You only need to create an instance of UIComponent if you are using version 2 of the Macromedia Component Architecture to create a new component. Even in that case, UIComponent is often created implicitly by other subclasses such as `Button`. If you do need to create an instance of UIComponent, use the following code:

```
class MyComponent extends mx.core.UIComponent;
```

UIComponent class (API)

Inheritance MovieClip > [UIObject class](#) > UIComponent

ActionScript Class Name mx.core.UIComponent

The methods, properties, and events of the UIComponent class allow you to control the common behavior of Flash visual components.

Method summary for the UIComponent class

The following table lists methods of the UIComponent class.

Method	Description
UIComponent.setFocus()	Returns a reference to the object that has focus.
UIComponent.setFocus()	Sets focus to the component instance.

Methods inherited from the UIObject class

The following table lists the methods the UIComponent class inherits from the UIObject class. When calling these methods from the UIComponent object, use the form *UIComponentInstance.methodName*.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it will be redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.
UIObject.setSkin()	Sets a skin in the object.
UIObject.setStyle()	Sets the style property on the style declaration or object.

Property summary for the UIComponent class

The following table lists properties of the UIComponent class.

Property	Description
UIComponent.enabled	Indicates whether the component can receive focus and input.
UIComponent.tabIndex	A number indicating the tab order for a component in a document.

Properties inherited from the UIObject class

The following table lists the properties the UIComponent class inherits from the UIObject class. When accessing these properties from the UIComponent object, use the form *UIComponentInstance.propertyName*.

Property	Description
UIObject.bottom	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
UIObject.height	The height of the object, in pixels. Read-only.
UIObject.left	The left edge of the object, in pixels. Read-only.
UIObject.right	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
UIObject.scaleX	A number indicating the scaling factor in the x direction of the object, relative to its parent.
UIObject.scaleY	A number indicating the scaling factor in the y direction of the object, relative to its parent.
UIObject.top	The position of the top edge of the object, relative to its parent. Read-only.
UIObject.visible	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
UIObject.width	The width of the object, in pixels. Read-only.
UIObject.x	The left edge of the object, in pixels. Read-only.
UIObject.y	The top edge of the object, in pixels. Read-only.

Event summary for the UIComponent class

The following table lists events of the UIComponent class.

Event	Description
UIComponent.focusIn	Broadcast when an object receives focus.
UIComponent.focusOut	Broadcast when an object loses focus.
UIComponent.keyDown	Broadcast when a key is pressed.
UIComponent.keyUp	Broadcast when a key is released.

Events inherited from the UIObject class

The following table lists the events the UIComponent class inherits from the UIObject class.

Event	Description
UIObject.draw	Broadcast when an object is about to draw its graphics.
UIObject.hide	Broadcast when an object's state changes from visible to invisible.
UIObject.load	Broadcast when subobjects are being created.

Event	Description
UIObject.move	Broadcast when the object has moved.
UIObject.resize	Broadcast when an object has been resized.
UIObject.reveal	Broadcast when an object's state changes from invisible to visible.
UIObject.unload	Broadcast when the subobjects are being unloaded.

UIComponent.enabled

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.enabled
```

Description

Property; indicates whether the component can (`true`) or cannot (`false`) accept focus and mouse input. The default value is `true`.

Example

The following example sets the `enabled` property of a `CheckBox` component to `false`:

```
checkBoxInstance.enabled = false;
```

UIComponent.focusIn

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(focusIn){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.focusIn = function(eventObject){  
    ...  
}  
componentInstance.addEventListener("focusIn", listenerObject)
```

Description

Event; notifies listeners that the object has received keyboard focus.

The first usage example uses an `on()` handler and must be attached directly to a component instance.

The second usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, *focusIn*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following code disables a Button component, `btn`, while a user types in the TextInput component, `txt`, and enables the button when the user click on it:

```
var txt:mx.controls.TextInput;
var btn:mx.controls.Button;

var txtListener:Object = new Object();
txtListener.focusOut = function() {
    _root.btn.enabled = true;
}
txt.addEventListener("focusOut", txtListener);

var txtListener2:Object = new Object();
txtListener2.focusIn = function() {
    _root.btn.enabled = false;
}
txt.addEventListener("focusIn", txtListener2);
```

See also

`EventDispatcher.addEventListener()` in Flash Help, [UIComponent.focusOut](#)

UIComponent.focusOut

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
on(focusOut){
```

```

    ...
}
listenerObject = new Object();
listenerObject.focusOut = function(eventObject){
    ...
}
componentInstance.addEventListener("focusOut", listenerObject)

```

Description

Event; notifies listeners that the object has lost keyboard focus.

The first usage example uses an `on()` handler and must be attached directly to a component instance.

The second usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, `focusOut`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following code disables a Button component, `btn`, while a user types in the TextInput component, `txt`, and enables the button when the user click on it:

```

var txt:mx.controls.TextInput;
var btn:mx.controls.Button;

var txtListener:Object = new Object();
txtListener.focusOut = function() {
    _root.btn.enabled = true;
}
txt.addEventListener("focusOut", txtListener);

var txtListener2:Object = new Object();
txtListener2.focusIn = function() {
    _root.btn.enabled = false;
}
txt.addEventListener("focusIn", txtListener2);

```

See also

`EventDispatcher.addEventListener()` in Flash Help, [UIComponent.focusIn](#)

UIComponent.getFocus()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.getFocus();
```

Parameters

None.

Returns

A reference to the object that currently has focus.

Description

Method; returns a reference to the object that has keyboard focus.

Example

The following code returns a reference to the object that has focus and assigns it to the tmp variable:

```
var tmp = checkbox.getFocus();
```

UIComponent.keyDown

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
on(keyDown){  
    ...  
}  
listenerObject = new Object();  
listenerObject.keyDown = function(eventObject){  
    ...  
}  
componentInstance.addEventListener("keyDown", listenerObject)
```

Description

Event; notifies listeners when a key is pressed. This is a very low-level event that you should not use unless necessary, because it can affect system performance.

The first usage example uses an `on()` handler and must be attached directly to a component instance.

The second usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, *keyDown*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following code makes an icon blink when a key is pressed:

```
formListener.handleEvent = function(eventObj)
{
    form.icon.visible = !form.icon.visible;
}
form.addEventListener("keyDown", formListener);
```

UIComponent.keyUp

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
on(keyUp){
    ...
}
listenerObject = new Object();
listenerObject.keyUp = function(eventObject){
    ...
}
componentInstance.addEventListener("keyUp", listenerObject)
```

Description

Event; notifies listeners when a key is released. This is a low-level event that you should not use unless necessary, because it can affect system performance.

The first usage example uses an `on()` handler and must be attached directly to a component instance.

The second usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, *keyUp*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following code makes an icon blink when a key is released:

```
formListener.handleEvent = function(eventObj)
{
    form.icon.visible = !form.icon.visible;
}
form.addEventListener("keyUp", formListener);
```

UIComponent.setFocus()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.setFocus();
```

Parameters

None.

Returns

Nothing.

Description

Method; sets the focus to this component instance. The instance with focus receives all keyboard input.

Example

The following code gives focus to the checkbox instance:

```
checkbox.setFocus();
```

UIComponent.tabIndex

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

instance.tabIndex

Description

Property; a number indicating the tabbing order for a component in a document.

Example

The following code sets the value of `tmp` to the `tabIndex` property of the checkbox instance:

```
var tmp = checkbox.tabIndex;
```

UIEventDispatcher class

ActionScript Class Name mx.events.UIEventDispatcher

Inheritance EventDispatcher class > UIEventDispatcher

The UIEventDispatcher class is mixed in to the UIComponent class and allows components to emit certain events.

If you want an object that doesn't inherit from UIComponent to dispatch certain events, you can use UIEventDispatcher.

Method summary for the UIEventDispatcher class

The following table lists the method of the UIEventDispatcher class.

Method	Description
<code>UIEventDispatcher.removeEventListener()</code>	Removes a registered listener from a component instance. This method overrides the <code>eventDispatcher.removeEventListener()</code> method.

Methods inherited from the EventDispatcher class

The following table lists the methods the UIEventDispatcher class inherits from the EventDispatcher class. When calling these methods from the UIEventDispatcher object, use the form *UIEventDispatcherInstance.methodName*.

Method	Description
<code>EventDispatcher.addEventListener()</code>	Registers a listener to a component instance.
<code>EventDispatcher.dispatchEvent()</code>	Dispatches an event to all registered listeners.

Event summary for the UIEventDispatcher class

The following table lists events of the UIEventDispatcher class.

Method	Description
UIEventDispatcher.keyDown	Broadcast when a key is pressed.
UIEventDispatcher.keyUp	Broadcast when a pressed key is released.
UIEventDispatcher.load	Broadcast when a component loads into Flash Player.
UIEventDispatcher.mouseDown	Broadcast when the mouse is pressed.
UIEventDispatcher.mouseOut	Broadcast when the mouse is moved off a component instance.
UIEventDispatcher.mouseOver	Broadcast when the mouse is moved over a component instance.
UIEventDispatcher.mouseUp	Broadcast when the mouse is pressed and released.
UIEventDispatcher.unload	Broadcast when a component is unloaded from Flash Player.

UIEventDispatcher.keyDown

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.keyDown = function(eventObject){
    // insert your code here
}
componentInstance.addEventListener("keyDown", listenerObject)
```

Description

Event; broadcast to all registered listeners when a key is pressed and the Flash application has focus.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event.

UIEventDispatcher.keyUp

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.keyUp = function(eventObject){
    // insert your code here
}
componentInstance.addEventListener("keyUp", listenerObject)
```

Description

Event; broadcast to all registered listeners when a key that was pressed is released and the Flash application has focus.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event.

UIEventDispatcher.load

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.load = function(eventObject){
    // insert your code here
}
componentInstance.addEventListener("load", listenerObject)
```

Description

Event; broadcast to all registered listeners when a component is loaded into Flash Player.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event.

UIEventDispatcher.mouseDown

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.mouseDown = function(eventObject){
    // insert your code here
}
componentInstance.addEventListener("mouseDown", listenerObject)
```

Description

Event; broadcast to all registered listeners when a Flash application has focus and the mouse is pressed.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event.

UIEventDispatcher.mouseOut

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.mouseOut = function(eventObject){
    // insert your code here
}
componentInstance.addEventListener("mouseOut", listenerObject)
```

Description

Event; broadcast to all registered listeners when a Flash application has focus and the mouse is moved off a component instance.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event.

UIEventDispatcher.mouseOver

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.mouseOver = function(eventObject){
    // insert your code here
}
componentInstance.addEventListener("mouseOver", listenerObject)
```

Description

Event; broadcast to all registered listeners when a Flash application has focus and the mouse is moved over a component instance.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event.

UIEventDispatcher.mouseUp

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();
listenerObject.mouseUp = function(eventObject){
    // insert your code here
}
componentInstance.addEventListener("mouseUp", listenerObject)
```

Description

Event; broadcast to all registered listeners when a Flash application has focus and the mouse is pressed and released.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. For more information, see “EventDispatcher class” in Flash Help.

UIEventDispatcher.removeListener()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004 and Flash MX Professional 2004.

Usage

```
componentInstance.removeEventListener(event, listener)
```

Parameters

event A string that is the name of the event.

listener A reference to a listener object or function.

Returns

Nothing.

Description

Method; unregisters a listener object from a component instance that is broadcasting an event. This method overrides the `EventDispatcher.removeListener()` event found in the `EventDispatcher` base class.

UIEventDispatcher.unload

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX Professional 2004.

Usage

```
listenerObject = new Object();  
listenerObject.unload = function(eventObject){  
    // insert your code here  
}  
componentInstance.addEventListener("unload", listenerObject)
```

Description

Event; broadcast to all registered listeners when a component is unloaded from Flash Player.

When the event is triggered, it automatically passes an event object (*eventObject*) to the handler. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event.

UIObject class

Inheritance MovieClip > UIObject

ActionScript Class Name mx.core.UIObject

UIObject is the base class for all version 2 components; it is not a visual component. The UIObject class wraps the ActionScript MovieClip object and contains functions and properties that allow version 2 components to share some common behavior. Wrapping the MovieClip class allows Macromedia to add new events and extend functionality in the future without breaking content. Wrapping the MovieClip class also allows users who aren't familiar with the traditional Flash concepts of "movie" and "frame" to use properties, methods, and events to create component-based applications without learning those concepts.

The UIObject class implements the following:

- Styles
- Events
- Resize by scaling

To use the methods and properties of the UIObject class, you call them directly from whichever component you are using. For example, to call the `UIObject.setSize()` method from the `RadioButton` component, you would write the following code:

```
myRadioButton.setSize(30, 30);
```

You only need to create an instance of UIObject if you are using version 2 of the Macromedia Component Architecture to create a new component. Even in that case, UIObject is often created implicitly by other subclasses like `Button`. If you do need to create an instance of UIObject, use the following code:

```
class MyComponent extends UIObject;
```

Method summary for the UIObject class

The following table lists methods of the UIObject class.

Method	Description
<code>UIObject.createClassObject()</code>	Creates an object on the specified class.
<code>UIObject.createObject()</code>	Creates a subobject on an object.
<code>UIObject.destroyObject()</code>	Destroys a component instance.
<code>UIObject.doLater()</code>	Calls a function when parameters have been set in the Property and Component inspectors.
<code>UIObject.getStyle()</code>	Gets the style property from the style declaration or object.
<code>UIObject.invalidate()</code>	Marks the object so it will be redrawn on the next frame interval.
<code>UIObject.move()</code>	Moves the object to the requested position.
<code>UIObject.redraw()</code>	Forces validation of the object so it is drawn in the current frame.
<code>UIObject.setSize()</code>	Resizes the object to the requested size.

Method	Description
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Property summary for the UIObject class

The following table lists properties of the UIObject class.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Event summary for the UIObject class

The following table lists events of the UIObject class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

UIObject.bottom

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.bottom
```

Description

Property (read-only); a number indicating the bottom position of the object, in pixels, relative to its parent's bottom. To set this property, call `UIObject.move()`.

Example

This example moves the check box so it aligns under the bottom edge of the list box:

```
myCheckbox.move(myCheckbox.x, form.height - listbox.bottom);
```

UIObject.createClassObject()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.createClassObject(className, instanceName, depth,  
initObject)
```

Parameters

className An object indicating the class of the new instance.

instanceName A string indicating the instance name of the new instance.

depth A number indicating the depth of the new instance.

initObject An object containing initialization properties for the new instance.

Returns

A UIObject object that is an instance of the specified class.

Description

Method; creates an instance of a component at runtime. You need to use the `import` statement and specify the class package name before calling this method. In addition, the component must be in the FLA file's library.

Example

The following code imports the assets of the Button component and then makes a subobject of the Button component.

```
import mx.controls.Button;
createClassObject(Button,"button2",5,{label:"Test Button"});
```

The following example creates a CheckBox object:

```
import mx.controls.CheckBox;
form.createClassObject(CheckBox, "cb", 0, {label:"Check this"});
```

You can also specify the class package name using the following syntax:

```
createClassObject(mx.controls.Button,"button2",5,{label:"Test Button"});
```

UIObject.createObject()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.createObject(linkageName, instanceName, depth, initObject)
```

Parameters

linkageName A string indicating the linkage identifier of a symbol in the library.

instanceName A string indicating the instance name of the new instance.

depth A number indicating the depth of the new instance.

initObject An object containing initialization properties for the new instance.

Returns

A UIObject object that is an instance of the symbol.

Description

Method; creates a subobject on an object. This method is generally used only by component developers or advanced developers.

Example

The following example creates a CheckBox instance on the form object:

```
form.createObject("CheckBox", "sym1", 0);
```

UIObject.destroyObject()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.destroyObject(instanceName)
```

Parameters

instanceName A string indicating the instance name of the object to be destroyed.

Returns

Nothing.

Description

Method; destroys a component instance.

UIObject.doLater()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.doLater(target, "function")
```

Parameters

target A reference to a Timeline that contains the specified function.

function A string indicating a function name to be called after a frame has passed.

Returns

Nothing.

Description

Method; calls a user-defined function only after the component has finished setting all of its properties from the Property inspector or Component inspector. All version 2 components that inherit from UIObject have the `doLater()` method.

Component properties set in the Property inspector or Component inspector may not be immediately available to ActionScript in the Timeline. For example, attempting to trace the `label` property from a CheckBox component using ActionScript on the first frame of your SWF fails without notification, even though the component appears on the Stage as expected.

Although properties that are set in a class or a frame script are available immediately, most properties assigned in the Property inspector or Component inspector are not set until the next frame within the component itself.

Although any approach that delays access of the property will resolve this problem, the simplest and most direct solution is to use the `doLater()` method.

Example

The following example shows how the `doLater()` method is used:

```
// doLater() is called from the component instance
myCheckBox.doLater (this, "delay");

// the function or method called from doLater()

function delay() {
    trace(myCheckBox.label); // the property can now be traced
    // any additional statements go here
}
```

UIObject.draw

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
on(draw){
    ...
}
listenerObject = new Object();
listenerObject.draw = function(eventObject){
    ...
}
componentInstance.addEventListener("draw", listenerObject)
```

Description

Event; notifies listeners that the object is about to draw its graphics. This is a low-level event that you should not use unless necessary, because it can affect system performance.

The first usage example uses an `on()` handler and must be attached directly to a component instance.

The second usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, *draw*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following code redraws the object `form2` when the `form` object is drawn:

```
formListener.draw = function(eventObj){
    form2.redraw(true);
}
form.addEventListener("draw", formListener);
```

See also

`EventDispatcher.addEventListener()` in Flash Help

UIObject.getStyle()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.getStyle(propertyName)
```

Parameters

propertyName A string indicating the name of the style property (for example, "fontWeight", "borderStyle", and so on).

Returns

The value of the style property. The value can be of any data type.

Description

Method; gets the style property from the style declaration or object. If the style property is an inheriting style, the ancestors of the object may be the source of the style value.

For a list of the styles supported by each component, see the individual component entries. See also “Using global, custom, and class styles in the same document” in Flash Help.

Example

The following code sets the `ib` instance's `fontWeight` style property to bold if the `cb` instance's `fontWeight` style property is bold:

```
if (cb.getStyle("fontWeight") == "bold")
{
    ib.setStyle("fontWeight", "bold");
};
```

UIObject.height

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

componentInstance.height

Description

Property (read-only); a number indicating the height of the object, in pixels. To change the height property, call `UIObject.setSize()`.

Example

The following example makes the check box taller:

```
myCheckbox.setSize(myCheckbox.width, myCheckbox.height + 10);
```

UIObject.hide

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
on(hide){
    ...
}
listenerObject = new Object();
listenerObject.hide = function(eventObject){
    ...
}
componentInstance.addEventListener("hide", listenerObject)
```

Description

Event; broadcast when the object's `visible` property is changed from `true` to `false`.

Example

The following handler displays a message in the Output panel when the object it's attached to becomes invisible.

```
on(hide) {  
    trace("I've become invisible.");  
}
```

See also

[UIObject.reveal](#)

UIObject.invalidate()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.invalidate()
```

Parameters

None.

Returns

Nothing.

Description

Method; marks the object so it will be redrawn on the next frame interval.

This method is primarily useful to developers of new custom components. A custom component is likely to support a number of operations that change the component's appearance.

Often, the best way to build a component is to centralize the logic for updating the component's appearance in the `draw()` method. If the component has a `draw()` method, you can call `invalidate()` on the component to redraw it. (For information on defining a `draw()` method, see "Defining core functions" in Flash Help.)

All operations that change the component's appearance can call `invalidate()` instead of redrawing the component themselves. This has some advantages: code isn't duplicated unnecessarily, and multiple changes can easily be batched up into one redraw, instead of causing multiple, redundant redraws.

Example

The following example marks the `ProgressBar` instance `pBar` for redrawing:

```
pBar.invalidate();
```

UIObject.left

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.left
```

Description

Property (read-only); a number indicating the left edge of the object, in pixels, relative to its parent. To set this property, call `UIObject.move()`.

UIObject.load

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(load){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.load = function(eventObject){  
    ...  
}  
componentInstance.addEventListener("load", listenerObject)
```

Description

Event; notifies listeners that the subobject for this object is being created.

The first usage example uses an `on()` handler and must be attached directly to a component instance.

The second usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, `load`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following example creates an instance of `MySymbol` once the `form` instance is loaded:

```
formListener.handleEvent = function(eventObj)
{
    form.createObject("MySymbol", "sym1", 0);
}
form.addEventListener("load", formListener);
```

UIObject.move

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(move){
    ...
}
```

Usage 2:

```
listenerObject = new Object();
listenerObject.move = function(eventObject){
    ...
}
componentInstance.addEventListener("move", listenerObject)
```

Description

Event; notifies listeners that the object has moved.

The first usage example uses an `on()` handler and must be attached directly to a component instance.

The second usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, *move*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following example calls the `move()` method to keep `form2` 100 pixels down and to the right of `form1`:

```
formListener.handleEvent = function(){
    form2.move(form1.x + 100, form1.y + 100);
}
form1.addEventListener("move", formListener);
```

UIObject.move()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

componentInstance.move(*x*, *y*)

Parameters

- x* A number that indicates the position of the object's upper left corner, relative to its parent.
- y* A number that indicates the position of the object's upper left corner, relative to its parent.

Returns

Nothing.

Description

Method; moves the object to the requested position. You should pass only integral values to `UIObject.move()`, or the component may appear fuzzy.

Example

This example move the check box 10 pixels to the right:

```
myCheckbox.move(myCheckbox.x + 10, myCheckbox.y);
```

UIObject.redraw()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.redraw(always)
```

Parameters

always A Boolean value. If *true*, the method draws the object, even if *invalidate()* wasn't called. If *false*, the method draws the object only if *invalidate()* was called.

Returns

Nothing.

Description

Method; forces validation of the object so that it is drawn in the current frame.

Example

The following example creates a check box and a button and draws them because other scripts are not expected to modify the form:

```
form.createClassObject(mx.controls.CheckBox, "cb", 0);  
form.createClassObject(mx.controls.Button, "b", 1);  
form.redraw(true)
```

UIObject.resize

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(resize){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.resize = function(eventObject){  
    ...  
}  
componentInstance.addEventListener("resize", listenerObject)
```

Description

Event; notifies listeners that an object has been resized.

The first usage example uses an `on()` handler and must be attached directly to a component instance.

The second usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, *resize*) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following example calls the `setSize()` method to make `sym1` half the width and a fourth of the height when `form` is moved:

```
formListener.handleEvent = function(eventObj){
    form.sym1.setSize(sym1.width / 2, sym1.height / 4);
}
form.addEventListener("resize", formListener);
```

UIObject.reveal

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
on(reveal){
    ...
}
listenerObject = new Object();
listenerObject.reveal = function(eventObject){
    ...
}
componentInstance.addEventListener("reveal", listenerObject)
```

Description

Event; broadcast when the object's `visible` property changes from `false` to `true`.

Example

The following handler displays a message in the Output panel when the object it's attached to becomes visible.

```
on(reveal) {  
    trace("I've become visible.");  
}
```

See also

[UIObject.hide](#)

UIObject.right

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

componentInstance.right

Description

Property (read-only); a number indicating the right edge of the object, in pixels, relative to its parent's right edge. To set this property, call [UIObject.move\(\)](#).

Example

The following example moves the check box so it aligns under the right edge of the list box:

```
myCheckbox.move(form.width - listbox.right, myCheckbox.y);
```

UIObject.scaleX

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

componentInstance.scaleX

Description

Property; a number indicating the scaling factor in the *x* direction of the object, relative to its parent.

Example

The following example makes the check box twice as wide and sets the `tmp` variable to the horizontal scale factor:

```
checkbox.scaleX = 200;  
var tmp = checkbox.scaleX;
```

UIObject.scaleY

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.scaleY
```

Description

Property; a number indicating the scaling factor in the *y* direction of the object, relative to its parent.

Example

The following example makes the check box twice as high and sets the `tmp` variable to the vertical scale factor:

```
checkbox.scaleY = 200;  
var tmp = checkbox.scaleY;
```

UIObject.setSize()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.setSize(width, height)
```

Parameters

width A number that indicates the width of the object, in pixels.

height A number that indicates the height of the object, in pixels.

Returns

Nothing.

Description

Method; resizes the object to the requested size. You should pass only integral values to `UIObject.setSize()`, or the component may appear fuzzy. This method (like all methods and properties of `UIObject`) is available from any component instance.

When you call this method on a `ComboBox` instance, the combo box is resized and the `rowHeight` property of the contained list is also changed.

Example

This example resizes the `pBar` component instance to 100 pixels wide and 100 pixels high:

```
pBar.setSize(100, 100);
```

UIObject.setSkin()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
componentInstance.setSkin(id, linkageName)
```

Parameters

id A number indicating the depth of the skin within the component.

linkageName A string indicating an asset in the library.

Returns

A reference to the movie clip (skin) that was attached.

Description

Method; sets a skin in the component instance. Use this method in a component's class file when you are creating a component. For more information, see "About assigning skins" in Flash Help.

You cannot use this method to set a component's skins at runtime (for example, the way you set a component's styles at runtime).

Example

This example is a code snippet from the class file of a new component, called `Shape`. It creates a variable, `themeShape` and sets it to the `Linkage` identifier of the skin. In the `createChildren()` method, the `setSkin()` method is called and passed the `id` 1 and the variable that holds the linkage identifier of the skin:

```
class Shape extends UIComponent{

    static var symbolName:String = "Shape";
    static var symbolOwner:Object = Shape;
    var className:String = "Shape";
```

```

var themeShape:String = "circle_skin"

function Shape(){
}

function init(Void):Void{
    super.init();
}

function createChildren():Void{
    setSkin(1, themeShape);
    super.createChildren();
}
}

```

UIObject.setStyle()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

componentInstance.setStyle(propertyName, value)

Parameters

propertyName A string indicating the name of the style property (for example, "fontWeight", "borderStyle", and so on).

value The value of the property. If the value is a string, it must be enclosed in quotation marks.

Returns

A UIObject object that is an instance of the specified class.

Description

Method; sets the style property on the style declaration or object. If the style property is an inheriting style, the children of the object are notified of the new value.

For a list of the styles supported by each component, see individual component entries. For example, Button component styles are listed in “Using styles with the Button component” under “Button component.”

Example

The following code sets the `fontWeight` style property of the check box instance `cb` to bold:

```
cb.setStyle("fontWeight", "bold");
```

UIObject.top

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

componentInstance.top

Description

Property (read-only); a number indicating the top edge of the object, in pixels, relative to its parent. To set this property, call [UIObject.move\(\)](#).

UIObject.unload

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(unload){  
    ...  
}
```

Usage 2:

```
listenerObject = new Object();  
listenerObject.unload = function(eventObject){  
    ...  
}  
componentInstance.addEventListener("unload", listenerObject)
```

Description

Event; notifies listeners that the subobjects of this object are being unloaded.

The first usage example uses an `on()` handler and must be attached directly to a component instance.

The second usage example uses a dispatcher/listener event model. A component instance (*componentInstance*) dispatches an event (in this case, `unload`) and the event is handled by a function, also called a *handler*, on a listener object (*listenerObject*) that you create. You define a method with the same name as the event on the listener object; the method is called when the event is triggered. When the event is triggered, it automatically passes an event object (*eventObject*) to the listener object method. Each event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call the `EventDispatcher.addEventListener()` method on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

For more information, see “EventDispatcher class” in Flash Help.

Example

The following example deletes `sym1` when the `unload` event is triggered:

```
function unload(){
    form.destroyObject(sym1);
}
form.addEventListener("unload", this);
```

UIObject.visible

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

componentInstance.visible

Description

Property; a Boolean value indicating whether the object is visible (`true`) or not (`false`).

Example

The following example makes the `myLoader` loader instance visible:

```
myLoader.visible = true;
```

UIObject.width

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

componentInstance.width

Description

Property (read-only); a number indicating the width of the object, in pixels. To change the width, call `UIObject.setSize()`.

Example

The following example makes the check box wider:

```
myCheckbox.setSize(myCheckbox.width + 10, myCheckbox.height);
```

UIObject.x**Availability**

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

componentInstance.x

Description

Property (read-only); a number indicating the left edge of the object, in pixels. To set this property, call `UIObject.move()`.

Example

The following example moves the check box 10 pixels to the right:

```
myCheckbox.move(myCheckbox.x + 10, myCheckbox.y);
```

UIObject.y**Availability**

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

componentInstance.y

Description

Property (read-only); a number indicating the top edge of the object, in pixels. To set this property, call `UIObject.move()`.

Example

The following example moves the check box down 10 pixels:

```
myCheckbox.move(myCheckbox.x, myCheckbox.y + 10);
```

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 runtime with ActionScript.

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.

Using the UIScrollBar component

To use the UIScrollBar component, verify that object snapping is turned on (View > Snapping > Snap to Objects). Then create a text input 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 will not be displayed correctly. One of the arrow buttons will become 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.

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 parameters

You can set the following authoring parameters for each UIScrollBar instance in the Property inspector or in the Component inspector:

_targetInstanceName indicates the name of the text field instance that the UIScrollBar component is attached to.

horizontal indicates whether the scroll bar is oriented horizontally (`true`) or vertically (`false`). The default value is `false`.

You can write ActionScript to control these and additional options for a UIScrollBar component using its properties, methods, and events. For more information, see [“UIScrollBar class” on page 561](#).

Creating an application with the UIScrollBar component

The following procedure explains how to add a UIScrollBar component to an application while authoring.

To create an application with the UIScrollBar component:

1. Create a text input field and give it an instance name in the Property inspector. Add enough text to the field so that users will have to scroll to view it all.
2. In the Property inspector, set the Line Type of the text input field to Multiline, or Multiline No Wrap if you plan to use the scroll bar horizontally.

3. Verify that object snapping is turned on (View > Snapping > Snap to Objects).
4. 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.

The `_targetInstanceName` property of the component is automatically populated with the text field instance name in the Property and Component inspectors.

5. Select Control > Test Movie.

The application runs, and the scroll bar scrolls the contents of the text field.

You can also create a `UIScrollBar` component instance and associate it with a text field at runtime with `ActionScript`.

The following code creates a vertically oriented `UIScrollBar` instance and attaches it to the right side of a text field instance named `MyTextField`:

```
// createClassObject like any other component. Name it vSB.
createClassObject(mx.controls.UIScrollBar, "vSB", 10);

// set the target text field
vSB.setScrollTarget(MyTextField);

// size it to match the text field
vSB.setSize(16, MyTextField._height);

// move it next to the text field
vSB.move(MyTextField._x + MyTextField._width, MyTextField._y);
```

The following code creates a horizontally oriented `UIScrollBar` instance and attaches it to the bottom of a text field instance named `MyTextField`:

```
// createClassObject like any other component. Name it hSB.
_root.createClassObject(mx.controls.UIScrollBar, "hSB", 20);
hSB.horizontal = true

// set the target text field
hSB.setScrollTarget(MyTextField);

// size it to match the text field
hSB.setSize(MyTextField._width, 16);

// move it to the bottom of the text field
hSB.move(MyTextField._x, MyTextField._y + MyTextField._height);
```

Customizing the `UIScrollBar` component

You can transform a `UIScrollBar` component horizontally and vertically while authoring and at runtime. While authoring, select the component on the Stage and use the Free Transform tool or any of the Modify > Transform commands. At runtime, use the `setSize()` method (see [UIObject.setSize\(\)](#)) or any applicable properties and methods of the `UIScrollBar` class.

Note, however, that with the Halo theme, the width of a vertically oriented scroll bar must be 16 pixels, and the height of a horizontally oriented scroll bar must also be 16 pixels. These dimensions are determined strictly by the current theme used with the scroll bar. Only the dimension of a scroll bar that corresponds to its length can be changed.

You can customize the appearance of a `UIScrollBar` instance by using styles and skins.

Using styles with the `UIScrollBar` component

The `UIScrollBar` component supports the following styles:

Style	Theme	Description
<code>themeColor</code>	Halo	The base color scheme of a component. Possible values are "haloGreen", "haloBlue", and "haloOrange". The default value is "haloGreen".
<code>scrollTrackColor</code>	Sample	The background color for the scroll track. The default value is <code>0xCCCCCC</code> (light gray).
<code>symbolColor</code>	Sample	The color of the up and down scroll arrows. The default value is <code>0x000000</code> (black).
<code>symbolDisabledColor</code>	Sample	The color of the up and down scroll arrows in a disabled scroll bar. The default value is <code>0x848384</code> (dark gray).

Using skins with the `UIScrollBar` component

The `UIScrollBar` component uses 13 skins for the track, scroll box (thumb), and buttons. To customize these skin elements, edit the symbols in the `Flash UI Components 2/Themes/MMDefault/ScrollBar Assets/States` folder. For more information, see “About skinning components” in Flash Help.

Both horizontal and vertical scroll bars use the same vertical skins, and when displaying a horizontal scroll bar the `UIScrollBar` component rotates the skins as appropriate.

The `UIScrollBar` component supports the following skin properties.

Property	Description
<code>upArrowUpName</code>	The up (normal) state of the up and left buttons. The default value is <code>ScrollUpArrowUp</code> .
<code>upArrowOverName</code>	The rollover state of the up and left buttons. The default value is <code>ScrollUpArrowOver</code> .
<code>upArrowDownName</code>	The pressed state of the up and left buttons. The default value is <code>ScrollUpArrowDown</code> .
<code>downArrowUpName</code>	The up (normal) state of the down and right buttons. The default value is <code>ScrollDownArrowUp</code> .
<code>downArrowOverName</code>	The rollover state of the down and right buttons. The default value is <code>ScrollDownArrowOver</code> .
<code>downArrowDownName</code>	The pressed state of the down and right buttons. The default value is <code>ScrollDownArrowDown</code> .

Property	Description
<code>scrollTrackName</code>	The symbol used for the scroll bar's track (background). The default value is <code>ScrollTrack</code> .
<code>scrollTrackOverName</code>	The symbol used for the scroll track (background) when rolled over. The default value is <code>undefined</code> .
<code>scrollTrackDownName</code>	The symbol used for the scroll track (background) when pressed. The default value is <code>undefined</code> .
<code>thumbTopName</code>	The top and left caps of the scroll box (thumb). The default value is <code>ScrollThumbTopUp</code> .
<code>thumbMiddleName</code>	The middle (expandable) part of the thumb. The default value is <code>ScrollThumbMiddleUp</code> .
<code>thumbBottomName</code>	The bottom and right caps of the thumb. The default value is <code>ScrollThumbBottomUp</code> .
<code>thumbGripName</code>	The grip displayed in front of the thumb. The default value is <code>ScrollThumbGripUp</code> .

The following example demonstrates how to put a thin blank line in the middle of the scroll track.

To create movie clip symbols for `UIScrollBar` skins:

1. Create a new FLA file.
2. Select `File > Import > Open External Library`, and select the `HaloTheme.fla` file.
This file is located in the application-level configuration folder. For the exact location on your operating system, see “About themes” in Flash Help.
3. In the theme's Library panel, expand the `Flash UI Components 2/Themes/MMDefault` folder and drag the `ScrollBar Assets` folder to the library for your document.
4. Expand the `ScrollBar Assets/States` folder in the library of your document.
5. Open the symbols you want to customize for editing.
For example, open the `ScrollTrack` symbol.
6. Customize the symbol as desired.
For example, draw a black rectangle in the middle of the track using a 1 x 4 rectangle at (8,0).
7. Repeat steps 5-6 for all symbols you want to customize.
For example, draw the same line on the `ScrollTrackDisabled` symbol.
8. Click the Back button to return to the main Timeline.
9. Create an input type `TextField` instance on the Stage.
10. Drag a `UIScrollBar` component to the `TextField` instance.
11. Select `Control > Test Movie`.

UIScrollBar class

Inheritance MovieClip > [UIObject class](#) > [UIComponent class](#) > ScrollBar > UIScrollBar

ActionScript Class Name mx.controls.UIScrollBar

The properties of the UIScrollBar class let you adjust the scroll position and the amount of scrolling that occurs when the user clicks the scroll arrows or the scroll track.

Unlike most other components, events are broadcast when the mouse button is pressed and continue broadcasting until the button is released.

Each component class has a `version` property, which is a class property. Class properties are available only on the class itself. The `version` property returns a string that indicates the version of the component. To access this property, use the following code:

```
trace(mx.controls.UIScrollBar.version);
```

Note: The code `trace(myUIScrollBarInstance.version);` returns `undefined`.

Method summary for the UIScrollBar class

The following table lists the method of the UIScrollBar class.

Method	Description
UIScrollBar.setScrollProperties()	Sets the scroll range of the scroll bar and the size of the text field that the scroll bar is attached to.
UIScrollBar.setScrollTarget()	Assigns the scroll bar to a text field.

Methods inherited from the UIObject class

The following table lists the methods the UIScrollBar class inherits from the UIObject class. When calling these methods from the UIScrollBar object, use the form *UIScrollBarInstance.methodName*.

Method	Description
UIObject.createClassObject()	Creates an object on the specified class.
UIObject.createObject()	Creates a subobject on an object.
UIObject.destroyObject()	Destroys a component instance.
UIObject.doLater()	Calls a function when parameters have been set in the Property and Component inspectors.
UIObject.getStyle()	Gets the style property from the style declaration or object.
UIObject.invalidate()	Marks the object so it will be redrawn on the next frame interval.
UIObject.move()	Moves the object to the requested position.
UIObject.redraw()	Forces validation of the object so it is drawn in the current frame.
UIObject.setSize()	Resizes the object to the requested size.

Method	Description
<code>UIObject.setSkin()</code>	Sets a skin in the object.
<code>UIObject.setStyle()</code>	Sets the style property on the style declaration or object.

Methods inherited from the UIComponent class

The following table lists the methods the UIScrollBar class inherits from the UIComponent class. When calling these methods from the UIScrollBar object, use the form

UIScrollBarInstance.methodName.

Method	Description
<code>UIComponent.getFocus()</code>	Returns a reference to the object that has focus.
<code>UIComponent.setFocus()</code>	Sets focus to the component instance.

Property summary for the UIScrollBar class

The following table lists properties of the UIScrollBar class.

Property	Description
<code>UIScrollBar.lineScrollSize</code>	The number of lines or pixels scrolled when the user clicks the arrow buttons of the scroll bar.
<code>UIScrollBar.pageScrollSize</code>	The number of lines or pixels scrolled when the user clicks the track of the scroll bar.
<code>UIScrollBar.scrollPosition</code>	The current scroll position of the scroll bar.
<code>UIScrollBar._targetInstanceName</code>	The instance name of the text field associated with the UIScrollBar instance.
<code>UIScrollBar.horizontal</code>	A Boolean value indicating whether the scroll bar is oriented vertically (<code>false</code>), the default, or horizontally (<code>true</code>).

Properties inherited from the UIObject class

The following table lists the properties the UIScrollBar class inherits from the UIObject class.

When accessing these properties from the UIScrollBar object, use the form

UIScrollBarInstance.propertyName.

Property	Description
<code>UIObject.bottom</code>	The position of the bottom edge of the object, relative to the bottom edge of its parent. Read-only.
<code>UIObject.height</code>	The height of the object, in pixels. Read-only.
<code>UIObject.left</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.right</code>	The position of the right edge of the object, relative to the right edge of its parent. Read-only.
<code>UIObject.scaleX</code>	A number indicating the scaling factor in the x direction of the object, relative to its parent.

Property	Description
<code>UIObject.scaleY</code>	A number indicating the scaling factor in the y direction of the object, relative to its parent.
<code>UIObject.top</code>	The position of the top edge of the object, relative to its parent. Read-only.
<code>UIObject.visible</code>	A Boolean value indicating whether the object is visible (<code>true</code>) or not (<code>false</code>).
<code>UIObject.width</code>	The width of the object, in pixels. Read-only.
<code>UIObject.x</code>	The left edge of the object, in pixels. Read-only.
<code>UIObject.y</code>	The top edge of the object, in pixels. Read-only.

Properties inherited from the `UIComponent` class

The following table lists the properties the `UIScrollBar` class inherits from the `UIComponent` class. When accessing these properties from the `UIScrollBar` object, use the form `UIScrollBarInstance.propertyName`.

Property	Description
<code>UIComponent.enabled</code>	Indicates whether the component can receive focus and input.
<code>UIComponent.tabIndex</code>	A number indicating the tab order for a component in a document.

Event summary for the `UIScrollBar` class

The following table lists the event of the `UIScrollBar` class.

Event	Description
<code>UIScrollBar.scroll</code>	Broadcast when any part of the scroll bar is clicked.

Events inherited from the `UIObject` class

The following table lists the events the `UIScrollBar` class inherits from the `UIObject` class.

Event	Description
<code>UIObject.draw</code>	Broadcast when an object is about to draw its graphics.
<code>UIObject.hide</code>	Broadcast when an object's state changes from visible to invisible.
<code>UIObject.load</code>	Broadcast when subobjects are being created.
<code>UIObject.move</code>	Broadcast when the object has moved.
<code>UIObject.resize</code>	Broadcast when an object has been resized.

Event	Description
<code>UIObject.reveal</code>	Broadcast when an object's state changes from invisible to visible.
<code>UIObject.unload</code>	Broadcast when the subobjects are being unloaded.

Events inherited from the `UIComponent` class

The following table lists the events the `UIScrollBar` class inherits from the `UIComponent` class.

Event	Description
<code>UIComponent.focusIn</code>	Broadcast when an object receives focus.
<code>UIComponent.focusOut</code>	Broadcast when an object loses focus.
<code>UIComponent.keyDown</code>	Broadcast when a key is pressed.
<code>UIComponent.keyUp</code>	Broadcast when a key is released.

`UIScrollBar.horizontal`

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
scrollBarInstance.horizontal
```

Description

Property; indicates whether the scroll bar is oriented vertically (`false`) or horizontally (`true`).

This property can be tested and set. The default value is `false`.

Example

The following example sets the scroll bar named `MyScrollBar` to a horizontal orientation:

```
myScrollBar.horizontal = true;
```

`UIScrollBar.lineScrollSize`

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
scrollBarInstance.lineScrollSize
```

Description

Property; gets or sets the number of lines or pixels scrolled when the user clicks the arrow buttons of the `UIScrollBar` component. If the scroll bar is oriented vertically, the value is a number of lines. If the scroll bar is oriented horizontally, the value is a number of pixels.

The default value is 1.

Example

The following example sets the scroll bar to scroll two lines of text each time the user clicks one of the scroll arrows:

```
myScrollBar.lineScrollSize = 2;
```

UIScrollBar.pageScrollSize

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
scrollBarInstance.pageScrollSize
```

Description

Property; gets or sets the number of lines or pixels scrolled when the user clicks the track of the `UIScrollBar` component. If the scroll bar is oriented vertically, the value is a number of lines. If the scroll bar is oriented horizontally, the value is a number of pixels.

You can also set this value by passing a *pageSize* parameter with the `UIScrollBar.setScrollTarget()` method.

Example

The following example sets the scroll bar to scroll 10 lines of text each time the user clicks the scroll track:

```
myScrollBar.pageScrollSize = 10;
```

UIScrollBar.scroll

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

Usage 1:

```
on(scroll){
    ...
}
```

Usage 2:

```
listenerObject = new Object();
listenerObject.scroll = function(eventObject){
    ...
}
UIScrollBarInstance.addEventListener("scroll", listenerObject)
```

Description

Event; broadcast to all registered listeners when the mouse is clicked (released) over the scroll bar. The `UIScrollBar.scrollPosition` property and the scroll bar's onscreen image are updated before this event is broadcast.

The first usage example uses an `on()` handler and must be attached directly to a `UIScrollBar` component instance. The keyword `this`, used inside an `on()` handler attached to a component, refers to the component instance. For example, the following code, attached to the `UIScrollBar` component instance `myUIScrollBarComponent`, sends “_level0.myUIScrollBarComponent” to the Output panel:

```
on(scroll){
    trace(this);
}
```

The second usage example uses a dispatcher/listener event model, in which the script is placed on a frame in the Timeline that contains the component instance. A component instance (`UIScrollBarInstance`) dispatches an event (in this case, `scroll`) and the event is handled by a function, also called a *handler*, on a listener object (`listenerObject`) that you create. You define a method with the same name as the event on the listener object; the method is called when the event occurs. When the event occurs, it automatically passes an event object (`eventObject`) to the listener object method. The event object has properties that contain information about the event. You can use these properties to write code that handles the event. Finally, you call `addEventListener()` (see `EventDispatcher.addEventListener()` in Flash Help) on the component instance that broadcasts the event to register the listener with the instance. When the instance dispatches the event, the listener is called.

In addition to the normal properties of the event object (`type` and `target`), the event object for the `scroll` event includes a third property named `direction`. The `direction` property contains a string describing which way the scroll bar is oriented. The possible values for the `direction` property are `vertical` (the default) and `horizontal`.

For more information about the `type` and `target` event object properties, see “Event objects” in Flash Help.

Example

The following code implements Usage 1. The code is attached to the `UIScrollBar` component instance and sends a message to the Output panel when the user clicks the scroll bar. The `on()` handler must be attached directly to the `UIScrollBar` instance.

```
on(scroll){
    trace("UIScrollBar component was clicked");
}
```

The following example implements Usage 2 and creates a listener object called `myListener` with a `scroll` event handler for the `verticalScroll` instance of the `UIScrollBar` component:

```
myListener = new Object();
myListener.scroll = function(eventObj){
    // insert code to handle the "scroll" event
}
verticalScroll.addEventListener("scroll", myListener);
```

UIScrollBar.scrollPosition

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

`scrollBarInstance.scrollPosition`

Description

Property; gets or sets the current scroll position of the scrollable text field. The position of the scroll box (thumb) also updates when a new `scrollPosition` value is set. The value of `scrollPosition` depends on whether the `UIScrollBar` instance is being used for vertical or horizontal scrolling.

If the `UIScrollBar` instance is being used for vertical scrolling (the most common use), the value of `scrollPosition` is an integer with a range that begins with 0 and ends with a number that is equal to the total number of lines in the text field divided by the number of lines that can be displayed in the text field simultaneously. If `scrollPosition` is set to a number greater than this range, the text field simply scrolls to the end of the text.

To scroll the text to the first line, set `scrollPosition` to 0.

To scroll the text to the end, set `scrollPosition` to the number of lines of text in the text field minus 1. You can determine the number of lines by retrieving the value of the `maxscroll` property of the text field.

If the `UIScrollBar` instance is being used for horizontal scrolling, the value of `scrollPosition` is an integer value ranging from 0 to the width of the text field, in pixels. You can determine the width of the text field in pixels by getting the value of the `maxhscroll` property of the text field.

The default value of `scrollPosition` is 0.

Example

The following example scrolls the text field to the beginning of the text it contains:

```
myScrollBar.scrollPosition = 0;
```

The following example scrolls the text field to the end of the text it contains:

```
myScrollBar.scrollPosition = myTextField.maxscroll - 1;
```

UIScrollBar.setScrollProperties()

Availability

Flash Player 6 (6.0.79.0).

Edition

Flash MX 2004.

Usage

```
scrollBarInstance.setScrollProperties(pageSize, minPos, maxPos)
```

Parameters

pageSize The number of items that can be viewed in the display area. This parameter sets the size of the text field's bounding box. If the scroll bar is vertical, this value is a number of lines of text; if the scroll bar is horizontal, this value is a number of pixels.

minPos This parameter refers to the lowest numbered line of text when the scroll bar is used vertically, or the lowest numbered pixel in the text field's bounding box when the scroll bar is used horizontally. The value is usually 0.

maxPos This value refers to the highest numbered line of text when the scroll bar is used vertically, or the highest numbered pixel in the text field's bounding box when the scroll bar is used horizontally.

Description

Method; sets the scroll range of the scroll bar and the size of the text field that the scroll bar is attached to. This method is primarily useful when you attach a UIScrollBar component to a text field at runtime (using `UIScrollBar.setScrollTarget()`) rather than while authoring.

The `minPos` and `maxPos` values are used together by the UIScrollBar component to determine the scroll range for the scroll bar and the associated text field.

If you use the `replaceText` method to set the text of the text field, you must use `setScrollProperties()` to cause an update of the scroll bars.

Example

The following example sets up a UIScrollBar component to display 10 lines of text at a time in the text field out of a range of 0 to 99 lines:

```
myScrollBar.setScrollProperties(10, 0, 99);
```

UIScrollBar.setScrollTarget()

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
scrollBarInstance.setScrollTarget(textInstance)
```

Parameters

textInstance The text field to assign to the scroll bar.

Description

Method; assigns a UIScrollBar component to a text field instance. If you need to associate a text field and a UIScrollBar component at runtime, use this method.

Example

The following example assigns the UIScrollBar instance named `myScrollBar` to the text field instance named `txt`. The scroll bar is oriented vertically.

```
myScrollBar.setScrollTarget(txt);
```

The following example assigns the UIScrollBar instance named `myScrollBar` to the text field instance named `task_list`. The scroll bar is oriented vertically.

```
myScrollBar.setScrollTarget(task_list, true);
```

UIScrollBar._targetInstanceName

Availability

Flash Player 6 (6.0 79.0).

Edition

Flash MX 2004.

Usage

```
scrollBarInstance._targetInstanceName
```

Description

Property; indicates the instance name of the text field associated with a UIScrollBar component. This property can be tested and set. However, it should not be used to create an association between a text field and a scroll bar. Use `UIScrollBar.setScrollTarget()` instead.

Example

The following example gets the name of the text field instance attached to the scroll bar `MyScrollBar` and sets its border to `true`:

```
var theName = myScrollBar._targetInstanceName;  
theName.border = true;
```