

ActionScript 3.0 Cookbook

Text

By Joey Lott, Darron Schall and Keith Peters

Publisher: O'Reilly Media, Inc.

Pub Date: 2006-10-11

ISBN: 9780596526955

Table Of Contents

Text.....	3
Introduction	3
Creating an Outline Around a Text Field	4
Creating a Background for a Text Field	4
Making a User Input Field	5
Making a Password Input Field	5
Filtering Text Input	6
Setting a Field's Maximum Length	7
Displaying Text	8
Displaying HTML-Formatted Text	8
Condensing Whitespace	9
Sizing Text Fields to Fit Contents	10
Scrolling Text Programmatically	11
Responding to Scroll Events	14
Formatting Text	14
Formatting User-Input Text	21
Formatting a Portion of Existing Text	21
Setting a Text Field's Font	22
Embedding Fonts	23
Creating Text that Can Be Rotated	24
Displaying Unicode Text	25
Assigning Focus to a Text Field	26
Selecting Text with ActionScript	27
Setting the Insertion Point in a Text Field	27
Responding When Text Is Selected or Deselected	28
Responding to User Text Entry	29
Adding a Hyperlink to Text	30
Calling ActionScript from Hyperlinks	32
Working with Advanced Text Layout	33
Applying Advanced Anti-Aliasing	35
Replacing Text	36
Retrieving a List of System Fonts	37
Index.....	38

Copyright (©) 2004, 2005, 2006, 2007 O'Reilly Media.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472

The copyrights in individual elements of this work are owned by their respective publishers, authors or others, as the case may be, and the prior written permission of the copyright owner is required for reuse in any form or medium of any individual element.

O'Reilly books may be purchased for educational, business or sales promotional use. Online editions are also available for most titles (<http://safari.oreilly.com/>). For more information, contact our corporate/institutional sales department: (800)998.9938 or corporate@oreilly.com.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. Microsoft, the .NET logo, Virtual C#, Visual Basic, Visual Studio, and Windows are registered trademarks or trademarks of Microsoft Corporation. Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps. The association between the image of the African crowned crane and the topic of C# is a trademark of O'Reilly Media, Inc.

While reasonable care has been exercised in the preparation of this book, the publisher and the authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Introduction

The *flash.text.TextField* class is the way in which all text is displayed in Flash Player. Even the text components such as *TextArea* and *TextInput* use the *TextField* class to display text. Flash Player enables a great deal of functionality for text fields from enabling user input to embedding fonts to using Cascading Style Sheets (CSS) to format text. In this chapter, we'll discuss all the many things you can accomplish with text.

As implied in the preceding paragraph, the *TextField* class is packaged in the *flash.display* package. Therefore, you need to either import the class or refer to the class with the fully qualified class name. All examples in this chapter assume you've imported the class with the following line of code:

```
import flash.text.TextField;
```

ActionScript 3.0 uses a display list that is quite different from previous versions of ActionScript. With earlier versions of ActionScript, you construct a text field using the *TextField* constructor as follows:

```
var field:TextField = new TextField( );
```

However, with ActionScript 3.0, the new text field object isn't automatically added to the display list. That means that if you want to make the text field visible, you have to use the *addChild()* method. As discussed in *Chapter 6*, the *addChild()* method is defined for all container display objects, such as *Sprite*, and it adds the object specified as a parameter to the display list of the object from which it is called. For example, the following line of code adds the field *TextField* object to the display list of the instance of the *TextExample* class:

```
package {  
  
    import flash.display.Sprite;  
    import flash.text.TextField;  
    public class TextExample extends Sprite {  
        public function TextExample() {  
            var field:TextField = new TextField( );  
            addChild(field);  
        }  
    }  
}
```

When the examples in this chapter reference an object called `field`, it's frequently assumed that the object is a *TextField* object that was instantiated via the *TextField* constructor and added to the display list with the *addChild()* method.

Creating an Outline Around a Text Field

Problem

You want to place a border around a text field.

Solution

Set the text field's border property to true. Additionally, you can change the color of the border by setting the object's borderColor property.

Discussion

By default, a text field does not have a visible border, which is assumed to be the most common preferred behavior. For example, you may not want a border around an item label. However, there are many cases in which you will want to apply a border to a text field, such as a field that requires some user input. The border shows the user where to click to input a value. Simply setting a text field's border property to true turns on the border around the object:

```
field.border = true;
```

To turn off the border, simply set the border property to false.

The default border color is black, but that can be changed with the borderColor property, which accepts a hex RGB value corresponding to the desired color:

```
field.borderColor = 0xFF00FF; // Make the border violet.
```

Creating a Background for a Text Field

Problem

You want to make a visible background behind the text in a text field.

Solution

Set the text field's background property to true. Additionally, you can change the color of the background by setting the object's backgroundColor property.

Discussion

By default, text fields don't have a visible background. However, you can create a background for a text field by setting the background property for that object to true:

```
field.background = true;
```

By default, the background for a text field is white (if made visible). You can, however, assign the background color by setting the value of the object's backgroundColor property, which accepts a hex RGB value corresponding to the desired color, as shown here:

```
field.backgroundColor = 0x00FFFF; // Set the background to light blue
```

Making a User Input Field

Problem

You want to create a user input field to allow the user to enter text.

Solution

Set the text field's `type` property to `TextFieldType.INPUT`.

Discussion

There are two types of text fields: dynamic and input. The default text field type is dynamic. This means that it can be controlled with `ActionScript`, but the user cannot input text into it. To enable the field for user input, set the `type` property to the `INPUT` constant of the `flash.display.TextFieldType` class:

```
field.type = TextFieldType.INPUT;
```

Though it isn't a requirement, input fields generally also have a border and a background. Otherwise, the user might find it difficult to locate and select the field:

```
field.border = true;  
field.background = true;
```

For a user to be able to input text, the field's `selectable` property must be `true`, which is the default. You don't need to set the `selectable` property to `true`, unless you previously set it to `false`.

If you have previously defined an input text field that you want to make a dynamic text field (so that it does not accept user input) you can set the `type` property to the `DYNAMIC` constant of the `flash.display.TextFieldType` class:

```
field.type = TextFieldType.DYNAMIC;
```

Making a Password Input Field

Problem

You want to create a password-style text field that hides the characters as asterisks. You also want to disable copying from the text field.

Solution

Set the text field's `password` property to `true`.

Discussion

When a user enters a password into a field, you generally want to make it so others aren't able to read the password. This is a basic security precaution. The common convention is to display only asterisks in the field as the user types. This way, the user can see that she is successfully entering a value without anyone else being able to easily read what she's just typed.

To create an input field that is automatically masked with asterisks, you only need to set the *TextField*.password property to true:

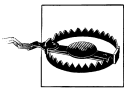
```
field.password = true;
```

When you set the password property to true, all text entered into the text field, either programmatically or by user input, displays as asterisks:

```
field.password = true;  
field.text = "example text"; // Text field displays: *****
```

Password text fields also disable copying from the text field. That prevents a user from being able to copy passwords or similar data and then paste it into a plain text document.

Password text fields are most commonly also input text fields; however, you can set the password property to true for a dynamic text field as well.



Although password text fields hide the data from the display, they do not encrypt the data or make it more secure for sending over a network. If data security is important to your application, you must use technologies that make data secure as it is transferred over a network. If you send data from Flash using standard HTTP, then the data is sent in an unsecured fashion. Use a technology such as SSL if you must send data in a secure fashion.

Filtering Text Input

Problem

You want to restrict the characters that a user can type into an input field.

Solution

Set the `restrict` property of the text field.

Discussion

By default, a user can type any character into an input field. However, in many scenarios, you might want to restrict the allowable characters. For example, you might restrict characters to numbers and dashes in the case of an input field for telephone numbers.

The *TextField*.`restrict` property lets you specify the allowed characters for user input into a field. Specify a string containing the allowable characters, such as:

```
field.restrict = "abcdefg";
```

This example lets the user enter any of the allowable characters: *a*, *b*, *c*, *d*, *e*, *f*, or *g*. Other characters are disallowed. If the user tries to enter *grabs*, only *gab* appears, since the letters *r* and *s* are not in the allowable character set.



If the `restrict` string is set to the empty string then *all* characters are allowed. To prevent input entirely, set the `type` to `DYNAMIC`.

Also, note that ActionScript distinguishes between upper- and lowercase characters. In other words, there is a difference between *a* and *A*. If the `restrict` property is set to `abcdefg`, the uppercase variants of the allowable characters (such as *A*, *B*, *C*) will be entered as the lowercase (allowable) equivalents (*a*, *b*, *c*). The same is true in reverse, such that if a lowercase character is entered when only the uppercase counterpart is allowed, the character is converted to uppercase.

The `restrict` property supports certain regular expression-like patterns. Therefore, you can also enter ranges by indicating the first character in the range and the last character in the range separated by a dash (-):

```
field.restrict = "a-zA-Z"; // Allow only upper- and lowercase letters
field.restrict = "a-zA-Z "; // Allow only letters and spaces
field.restrict = "0-9"; // Allow only numbers
```

In addition to specifying allowable characters, you can also disallow characters with a `restrict` string by using the caret character (^).

All characters and ranges in a `restrict` string following the caret are disallowed; for example:

```
field.restrict = "^abcdefg"; // Allows all except lowercase a through g
field.restrict = "^a-z"; // Disallows all lowercase letters (but allows other
// characters, including uppercase.)
field.restrict = "0-9^5"; // Allows numbers only, with the exception of 5
```

You can also specify allowable characters by using Unicode escape sequences. For example, if you want to disallow users from entering the `␣` character (Control-Z) into a field, you can specify its Unicode code point in the `restrict` property, as follows:

```
field.restrict = "\\u001A";
```

To allow a literal character that has a special meaning when used in a `restrict` string (such as a dash or caret), you must *escape* the character in the `restrict` string by preceding it with two backslashes (not just one), as shown here:

```
field.restrict = "0-9\\\\-"; // Allow numbers and dashes
field.restrict = "0-9\\\\^"; // Allow numbers and caret marks
```

If you want to escape the backslash character, you must precede it with three backslashes, for a total of four backslashes:

```
field.restrict = "0-9\\\\\\\\"; // Allow numbers and backslashes
```

The `restrict` property only affects the characters that the user can input. It does not have any effect on which characters can be displayed programmatically.

Setting a Field's Maximum Length

Problem

You want to limit the length of the string input into a text field.

Solution

Set the text field's `maxChars` property.

Discussion

By default, an input text field allows a user to type in as many characters as he desires. However, you may have good reason to want to set a maximum. For example, if an input field prompts a user for his two-character country code, you might want to prevent the user from entering more than two characters. Setting the `maxChars` property to a number limits the user input to that many characters:

```
field.maxChars = 6; // maximum of 6 characters can be input
```

Set `maxChars` to `null` to allow an entry of unlimited length, if you've previously assigned a non-null value to `maxChars`.

See Also

Recipe 9.5

Displaying Text

Problem

You want to display text within a text field.

Solution

Set the `text` property of a text field.

Discussion

Aside from being used as input fields, text fields are often used to display text to the user. Setting a text field's `text` property causes the corresponding text to display in the field:

```
field.text = "this will display in the field";
```

Special characters, such as `\t` for tab and `\n` for newline, can be used within a text string.

You can append text by using the `+=` operator or the `appendText()` method:

```
field.appendText("new text");
```

See Also

Recipe 9.8 for information on support for HTML-formatted text.

Displaying HTML-Formatted Text

Problem

You want to display HTML content in a text field.

Solution

Set the text field's `htmlText` property to the value of the HTML content to display.

Discussion

Text fields can interpret and display basic HTML tags, if properly configured. Using HTML in a text field is a convenient way to add hyperlinks and simple formatting, such as font color and bolded text.

The value of the text field object's `htmlText` property is interpreted as HTML:

```
field.htmlText = "<u>This displays as underlined text.</u>";
```

No matter what, the `text` property of a text field is rendered as plain text. This means that even if the `text` property is set to `<u>test</u>`, the object displays `<u>test</u>` instead of `test`. That means that if you want to display HTML code in its unrendered format assign the HTML value to the `text` property of the text field, as follows:

```
field.text = "<u>underlined text</u>";
```

```
/* text field displays:  
<u>underlined text</u>  
*/
```

This can be a useful technique if, for example, you want to show both the rendered HTML and the HTML source code in side-by-side text fields:

```
htmlCode = "<i>italicized text</i>";  
sourceHTML.text = htmlCode;  
renderedHTML.htmlText = htmlCode;
```

You cannot display both rendered and unrendered HTML in the same text field. If you try, you will end up with unreliable results.

The set of HTML tags supported by text fields includes: ``, `<i>`, `<u>`, `` (with `face`, `size`, and `color` attributes), `<p>`, `
`, `<a >`, ``, ``, and `<textformat>` (with `leftmargin`, `rightmargin`, `blockindent`, `indent`, `leading`, and `tabstops` attributes corresponding to the `TextFormat` class's properties of the same names).

Condensing Whitespace

Problem

You want to condense whitespace in an HTML text field display.

Solution

Set the object's `condenseWhite` property to `true`.

Discussion

When you use HTML in a text field, the optional `condenseWhite` setting condenses whitespace, as is done in most HTML browsers. For example, the following text would be rendered in a web

browser with only a single space between “hello” and “friend” in spite of the fact that the original source has multiple spaces between the two words.

```
hello      friend
```

In ActionScript text fields, however, all of the spaces are displayed, unless you set the `condenseWhite` property to true:

```
field.condenseWhite = true;  
field.htmlText = "hello      friend"; // Displays: "hello friend"
```

The `condenseWhite` property works only when the `html` property is true.

See Also

Recipe 9.8

Sizing Text Fields to Fit Contents

Problem

You want to size a text field’s viewable area to fit the text it contains.

Solution

Use the `autoSize` property.

Discussion

You can set the `autoSize` property of a text field so it automatically resizes itself in order to fit its contents. The possible values for `autoSize` are the `RIGHT`, `LEFT`, `CENTER`, and `NONE` constants of the *flash.text.TextFieldAutoSize* class. By default, `autoSize` is set to `NONE`, meaning that the text field does not automatically resize.

Set the property to `LEFT` if you want the text field to resize while fixing the upper-left corner’s position. In other words, the text field’s lower-right corner is the point that moves when it expands and contracts:

```
// These two lines do the same thing  
field.autoSize = TextFieldAutoSize.LEFT;  
field.autoSize = true;
```

Set the property to `CENTER` if you want the text field to be anchored at its center point. While the top of the object remains fixed, it expands and contracts downward and equally to the right and left:

```
field.autoSize = TextFieldAutoSize.CENTER;
```

Set the property to `RIGHT` if you want the upper-right corner of the text field to remain steady while the object expands and contracts in the direction of the lower-left corner:

```
field.autoSize = TextFieldAutoSize.RIGHT;
```

When *wordWrap* is set to `false` (the default), then the text field expands horizontally to accommodate the text. In such a case, the text field expands vertically only if there are newlines

within the text assigned to the text field. The following example illustrates a text field that auto sizes to accommodate all the text on one line:

```
var field:TextField = new TextField();
field.autoSize = TextFieldAutoSize.LEFT;
field.text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi tortor purus, aliquet a, ornare a
addChild(field);
```

The following adds a newline character to the text assigned to the text field so that it auto sizes to display all the text on two lines:

```
var field:TextField = new TextField();
field.autoSize = TextFieldAutoSize.LEFT;
field.text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi tortor purus, aliquet a, ornare a
field.text += "\\n";
field.text += "Nullam hendrerit molestie erat. Nunc nulla tortor, ullamcorper et, elementum vel, fringilla sed
addChild(field);
```

When *wordWrap* is set to true, then the text field never expands beyond the value of the *width* property (100 by default). If necessary, the text automatically wraps to a new line if *autoSize* is set to RIGHT, LEFT, or CENTER:

```
var field:TextField = new TextField();
field.autoSize = TextFieldAutoSize.LEFT;
field.wordWrap = true;
field.text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi tortor purus, aliquet a, ornare a
addChild(field);
```

See Also

Recipe 9.6

Scrolling Text Programmatically

Problem

You want to scroll text in a text field via `ActionScript`.

Solution

Use the `scrollV`, `maxScrollV`, `bottomScrollV`, `scrollH`, and `maxScrollH` properties of the text field. Use the `mouseWheelEnabled` property to enable scrolling of text by way of the mouse wheel.

Discussion

You can control the scrolling of a text field with `ActionScript` and without the aid of a scrollbar. For example, you may want to scroll the contents of a text field automatically to display a word or selection within the text. You can programmatically control a text field's scrolling in both the vertical and horizontal directions using some built-in properties. You should use the `scrollV`, `maxScrollV`, and `bottomScrollV` properties to control vertical scrolling, and use the `scrollH` and `maxScrollH` properties to control horizontal scrolling.

Every text field has a number of lines, whether it is 1 or 100. Each of these lines is identified by a number starting at 1. Some of these lines may be visible, and some may be beyond the border of the text field. Therefore, to view the lines that extend beyond the visible portion of the text field you must scroll to them. *Figure 9-1* illustrates this point. It depicts a text field's display where the solid line indicates the object's border (the visible area), and the dotted line surrounds the rest of the text contained within the object but lying outside the its visible area. To the left of the text field are line numbers for each line of text. The three labels—`scrollV`, `bottomScrollV`, and `maxScrollV`—indicate the meaning of the text field properties of the same names.

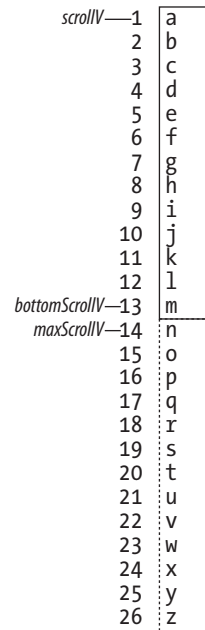


Figure 9-1.
Understanding `scrollV`, `bottomScrollV`, and `maxScrollV`

The `scrollV` property is a read-write property that indicates the top line of the text field's visible area. In *Figure 9-1*, the `scrollV` property's value is 1. To scroll the contents of a text field, assign a newline number to the `scrollV` property. Setting `scrollV` to 6, for example, scrolls the contents of the text field up until line 6 is the top-most line displayed. The value of `scrollV` should always be an integer; Flash cannot scroll to non-integer values:

```
field.scrollV = 1; // Scroll to the top
field.scrollV += 1; // Scroll to the next line
field.scrollV = 6; // Scroll to line 6
```

You can scroll to the next page of a text field's contents by using the `bottomScrollV` property, which indicates the bottom-most visible line in the text field. While you cannot set `bottomScrollV`, you can use it to determine the new value to assign to `scrollV`. In *Figure 9-1*, `bottomScrollV` is 13. If `scrollV` is set to 6, then `bottomScrollV` is automatically updated to 18:

```
// Scroll to the next page with the previous page's bottom line
// at the top
field.scrollV = field.bottomScrollV;
// Scroll to the next complete page without the bottom line from
// previous page
field.scrollV = field.bottomScrollV + 1;
```

You should use the `maxScrollV` property to scroll to the last page of contents within a text field. The `maxScrollV` property is also a read-only property. This property contains the value of the maximum line number that can be assigned to `scrollV`. Therefore, the `maxScrollV` property changes only when the number of lines in the text field changes (either through user input or `ActionScript` assignment). In *Figure 9-1*, `maxScrollV` is 14. This is because with 26 total lines in the text field and 13 visible lines, when `scrollV` is set to 14, the last visible line is 26 (the last line in the text field).

Don't try to set `scrollV` to a value less than 1 or greater than the value of `maxScrollV`. Although this won't cause an error, it won't scroll the text beyond the contents. Add blank lines to the beginning or end of the text field's contents to artificially extend its scrolling range:

```
field.scrollV = field.maxScrollV; // Scroll to the bottom
```

The vertical scrolling properties are in units of lines, but the horizontal scrolling properties (`scrollH` and `maxScrollH`) are in units of pixels. Other than that, `scrollH` and `maxScrollH` work more or less in the same fashion as `scrollV` and `maxScrollV` (there is no property for horizontal scrolling that corresponds to `bottomScrollV`). The `scrollH` property is a read-write property that allows you to control the value of the leftmost visible pixel starting with 0. The `maxScrollH` property is a read-only property that indicates the pixel value of the maximum value that can be assigned to `scrollH`:

```
field.scrollH = 0; // Scroll to the far left
field.scrollH += 1; // Scroll to the right 1 pixel
field.scrollH = field.maxScrollH; // Scroll to the far right
```

Figure 9-2 depicts `maxScrollH` for a dynamic text field (a field whose type is set to `DYNAMIC`). Text shown in gray is outside the visible area of the text field.

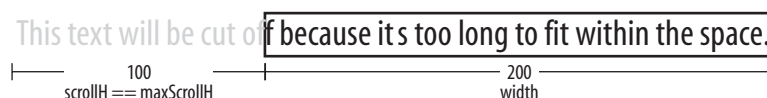


Figure 9-2.
The `maxScrollH` property for a dynamic text field

Figure 9-3 depicts `maxScrollH` for an input text field (a field whose type is set to `input`).



Flash automatically adds a buffer space to allow room for user input. Again, text shown in gray is outside the visible area of the text field.

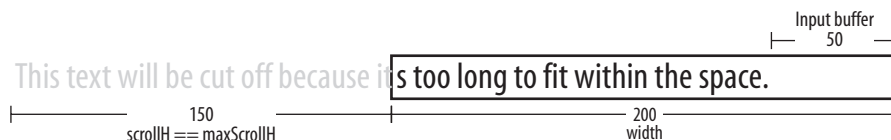


Figure 9-3.
The `maxScrollH` property for an input text field

Use the `mouseWheelEnabled` property to enable text scrolling using a scrollwheel mouse. The property is set to `false` by default; setting the property to `true` enables mouse wheel scrolling, as shown here:

```
field.mouseWheelEnabled = true;
```

Responding to Scroll Events

Problem

You want to have some actions performed when a text field's contents are scrolled.

Solution

Listen for the scroll event.

Discussion

When a text field is scrolled vertically or horizontally (meaning that the `scrollY` or `scrollX` property has been changed either by your custom ActionScript code or a scrollbar), the text field dispatches a scroll event. The scroll event name is defined by the `SCROLL` constant of the `flash.events.Event` class, and the event that is dispatched is a `flash.events.Event` object. The following code registers a listener for the scroll event for a text field called `field`:

```
field.addEventListener(Event.SCROLL, onTextScroll);
```

The following defines an example `onTextScroll()` method that listens for the scroll event:

```
private function onTextScroll(event:Event):void {  
    trace("scroll");  
}
```

See Also

Recipes [9.23](#) and [9.24](#)

Formatting Text

Problem

You want to format the text in a text field.

Solution

Use HTML tags, pass a `TextFormat` object to the `TextField.setTextFormat()` method, or use a `StyleSheet` object, and apply it to the text field's `styleSheet` property.

Discussion

Although you can set the color of the entire contents of the text field using the `textColor` property, for example, the `TextField` class doesn't offer precise control over character formatting. However, there are several ways in which you can apply more exacting formatting to text fields:

- Use HTML tags to apply formatting. For example, you can use ``, ``, and `<u>` tags.
- Use a `TextFormat` object.
- Use CSS.

Each of the three ways in which you can apply formatting to a text field has its own advantages and disadvantages. Applying HTML formatting is relatively simple, but it is the most difficult to manage of the different techniques. Using a `TextFormat` object is more complex than applying formatting via HTML. However, when you want to apply complex formatting, it is a better option than HTML formatting. Using CSS with a `StyleSheet` object allows the greatest flexibility, and it allows you to load a CSS document, which makes it simple to maintain because you can edit the CSS document without having to re-export the `.swf` file.

For fast and simple formatting, HTML tags are simplest. For example, the following code displays bolded and underlined text:

```
field.html = true;
field.htmlText = "<b>Bold text</b> <u>Underlined text</u>";
```

You can use `TextFormat` objects to apply more complex formatting to the text displayed in text fields. The first step in formatting text with a `TextFormat` object is to instantiate the object by using the constructor method:

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

Next, assign values to the `TextFormat` object's properties as you want:

```
formatter.bold = true;           // Bold the text
formatter.color = 0xFFFF00;     // Make the text yellow
formatter.blockIndent = 5;      // Adjust the margin by 5 points
```

You can apply text formatting to the existing text for an entire text field by passing a `TextFormat` object to the text field's `setTextFormat()` method:

```
field.setTextFormat(formatter);
```

When you invoke the `setTextFormat()` method this way, the formatting from the `TextFormat` object is applied to the text already assigned to the text field. The formatting does not apply to any text assigned to the text field after the `setTextFormat()` method is invoked. If additional text is entered by the user, the original text retains its applied formatting, but the inserted text does not have any special formatting applied to it. All formatting is removed if the text value is modified by appending a value by way of `ActionScript`:

```
field.text = "this is sample text";
field.setTextFormat(formatter); // Formatting applied
field.text = "this is new text"; // No formatting applied
field.setTextFormat(formatter); // Formatting reapplied
field.text += "appended text"; // Formatting removed
```

If you make changes to the `TextFormat` object, you should reapply the formatting to the text field by passing the modified object to the `setTextFormat()` method. Otherwise, the changes are not automatically displayed.

CSS support is available using the *flash.text.StyleSheet* class. The *StyleSheet* constructor does not require any parameters, as shown here:

```
var css:StyleSheet = new StyleSheet( );
```

Table 9-1 lists the supported CSS properties and the equivalent ActionScript properties. Use the CSS property names when defining a CSS document or a string to parse as CSS, and use the ActionScript properties when defining style objects (as discussed next).

Table .
CSS properties supported by Flash Player

CSS property	ActionScript property	Description
	<i>color</i>	The hexadecimal value as a string in the format of #RRGGBB.
		The way in which the text is displayed in the text field. The possible options are <i>inline</i> (no line breaks before or after the text), <i>block</i> (default style with line breaks before and after), or <i>none</i> (hidden).
<i>display</i>		<i>display</i>
	<i>font-family</i>	A comma-delimited list of font names. In addition to embedded fonts or named device fonts, Flash Player supports the following device font groups: <i>_sans</i> , <i>_serif</i> , <i>_typewriter</i> . You can also use <i>sans-serif</i> ,
		<i>fontFamily</i>

<i>CSS property</i>	<i>ActionScript property</i>	<i>Description</i>
		<i>serif, and mono, and they will be interpreted as _sans, _serif, and _typewriter, respectively.</i>
<i>font-size</i>	<i>fontSize</i>	<i>The numeric font size.</i>
<i>font-style</i>	<i>fontStyle</i>	<i>normal or italic.</i>
<i>font-weight</i>	<i>fontWeight</i>	<i>normal or bold.</i>
		<i>true or false.</i>
		<i>Kerning only works with embedded fonts that support kerning.</i>
<i>kerning</i>	<i>kerning</i>	<i>Additionally, the .swf must be compiled on Windows for kerning to work.</i>
		<i>The number of pixels to add between letters.</i>
<i>letter-spacing</i>	<i>letterSpacing</i>	
		<i>Number of pixels to apply to the left margin.</i>
<i>margin-left</i>	<i>marginLeft</i>	
		<i>Number of pixels to apply to the right margin.</i>
<i>margin-right</i>	<i>marginRight</i>	
		<i>left, center, right, or justify.</i>
<i>text-align</i>	<i>textAlign</i>	
		<i>none or underline.</i>
<i>text-decoration</i>	<i>textDecoration</i>	
		<i>Number of pixels to apply as the indent.</i>
<i>text-indent</i>	<i>textIndent</i>	

You can construct a new *StyleSheet* object and then populate it in several ways. One way to populate a *StyleSheet* object is to define style objects and assign them by using the *setStyle()* method. A style

object is an associative array with properties from the ActionScript properties list of *Table 9-1*. The following is an example of a style object:

```
var sampleStyle:Object = new Object();
sampleStyle.color = "#FFFFFF";
sampleStyle.textAlign = "center";
```

Note that you can define a style object with object literal notation as well, as shown here:

```
var sampleStyle:Object = {color: "#FFFFFF", textAlign: "center"};
```

Once you've defined one or more style objects, you can add them to the *StyleSheet* object with the *setStyle()* method. The *setStyle()* method requires two parameters: the name of the style and the style object. You can define styles for tags as well as for classes. The following defines a CSS class called *sample*:

```
css.setStyle(".sample", sampleStyle);
```

Although you can define a stylesheet with style objects and the *setStyle()* method, the most common and practical use of the class is to load and parse an external CSS document. Loading CSS at runtime has the advantage that you can change the styles by editing the CSS document, and you don't have to recompile the *.swf*.

To load a CSS document, use the *flash.net.URLLoader* class. Once data from the document has loaded, you can use that data to populate a *StyleSheet* object by passing the data to the *parseCSS()* method of the object. The next example uses *styles.css*, a CSS document defined as follows and saved in the same directory as the *.swf*.

```
p {
  font-family: _sans;
  color: #FFFFFF;
}
.emphasis {
  font-weight: bold;
  font-style: italic;
}
```

The following example illustrates how to load a CSS document and use that data to populate a *StyleSheet* object:

```
package {

  import flash.display.Sprite;
  import flash.text.TextField;
  import flash.events.Event;
  import flash.text.TextFieldAutoSize;
  import flash.text.StyleSheet;
  import flash.net.URLLoader;
  import flash.net.URLRequest;

  public class CSSText extends Sprite {

    public function CSSText() {
      var loader:URLLoader = new URLLoader();
      loader.addEventListener(Event.COMPLETE, onLoadCSS);
      var request:URLRequest = new URLRequest("styles.css");
      loader.load(request);
    }
  }
}
```

```

    }

    private function onLoadCSS(event:Event):void {
        var css:StyleSheet = new StyleSheet( );
        css.parseCSS(URLLoader(event.target).data);
        var field:TextField = new TextField( );
        field.autoSize = TextFieldAutoSize.LEFT;
        field.wordWrap = true;
        field.width = 200;
        addChild(field);
        field.styleSheet = css;
        field.htmlText = "<p><span class='emphasis'>Lorem ipsum</span> dolor sit amet, consectetur adipiscing
    }
}
}
}

```

There are a few things to watch for when working with CSS in Flash:

- CSS can be applied only to text fields rendering HTML.
- The HTML and CSS must correspond. For example, if the CSS defines a class called *someCSSClass*, then it has an effect only if the HTML applies that class to some portion of the text (as in the preceding example).
- The HTML text must be applied *after* the CSS is applied.

If you want to allow the user to select from different CSS documents, it may be helpful to store the HTML text in a variable or an associative array. Then you can re-apply the HTML each time the new CSS is loaded, as in the following example:

```

package {

    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import flash.text.TextFieldAutoSize;
    import flash.text.StyleSheet;
    import flash.net.URLLoader;
    import flash.net.URLRequest;

    public class CSSText extends Sprite {

        private var _field:TextField;
        private var _html:String;

        public function CSSText( ) {
            var css1:TextField = new TextField( );
            css1.text = "stylesheet 1";
            css1.selectable = false;
            var css1Container:Sprite = new Sprite( );
            css1Container.addEventListener(MouseEvent.CLICK, onCSS1);
            css1Container.addChild(css1);
            addChild(css1Container);

            var css2:TextField = new TextField( );
            css2.text = "stylesheet 2";

```

```

css2.selectable = false;
var css2Container:Sprite = new Sprite( );
css2Container.addEventListener(MouseEvent.CLICK, onCSS2);
css2Container.addChild(css2);
addChild(css2Container);

css2Container.y = 25;

    _field = new TextField( );
    _field.autoSize = TextFieldAutoSize.LEFT;
    _field.wordWrap = true;
    _field.width = 200;
    addChild(_field);
    _html = "<p><span class='emphasis'>Lorem ipsum</span> dolor sit amet, consectetur adipiscing elit. M";
    _field.y = 50;
}

private function loadCSS(url:String):void {
    var loader:URLLoader = new URLLoader( );
    loader.addEventListener(Event.COMPLETE, onLoadCSS);
    var request:URLRequest = new URLRequest(url);
    loader.load(request);
}

private function onCSS1(event:MouseEvent):void {
    loadCSS("styles.css");
}

private function onCSS2(event:MouseEvent):void {
    loadCSS("styles2.css");
}

private function onLoadCSS(event:Event):void {
    var css:StyleSheet = new StyleSheet( );
    css.parseCSS(URLLoader(event.target).data);
    _field.styleSheet = css;
    _field.htmlText = _html;
}
}
}

```

See Also

Recipe 9.8 explains how to use HTML-formatted text, *Recipe 9.14* explains how to apply formatting to new text rather than existing text, and *Recipe 9.15* applies formatting to individual characters rather than an entire field.

Formatting User-Input Text

Problem

You want to apply formatting to text as the user enters it into a text field.

Solution

Apply a *TextFormat* object by using the *defaultTextFormat* property of the text field.

Discussion

You should use the *defaultTextFormat* property of a text field object to apply text formatting to text as it is entered by user input. Create a *TextFormat* object as in *Recipe 9.13*, and then assign that object to the text field's *defaultTextFormat* property:

```
var formatter:TextFormat = new TextFormat();
formatter.color = 0x0000FF; // Make the text blue
field.defaultTextFormat = formatter;
```

When you use *defaultTextFormat*, the formatting is applied to text that the user types into the field.

See Also

Recipes *9.13* and *9.15*

Formatting a Portion of Existing Text

Problem

You want to add formatting to some, but not all, text in a text field, or you want to apply different formatting to various parts of a text field.

Solution

Create a *TextFormat* object and use it to format a substring of the text field by using one of the *setTextFormat()* method variations.

Discussion

You can format an entire text field as shown in *Recipe 9.13*, or you can use one of the versions of the *setTextFormat()* method to format just a portion of a text field. These variations allow you to apply formatting to the specified character range only.

You can set the formatting for a single character within a text field by invoking the *setTextFormat()* method and passing it two parameters:

index

The zero-relative index of the character to which the formatting should be applied.

textFormat

A reference to a *TextFormat* object.

This example applies the formatting to the first character only:

```
field.setTextFormat(0, formatter);
```

Alternatively, if you want to apply the formatting to a range of characters, you can invoke *setTextFormat()* with three parameters:

startIndex

The beginning, zero-relative character index.

endIndex

The index of the character after the last character in the desired range.

textFormat

A reference to a *TextFormat* object.

This example applies the formatting to the first 10 characters:

```
field.setTextFormat(0, 10, formatter);
```

You may notice that when you try to format portions of a text field, certain formatting options do not get applied under certain circumstances. For example, text alignment is applied only if the formatting is applied to the first character in the line.

See Also

Recipe 9.13

Setting a Text Field's Font

Problem

You want to use ActionScript to change the font used for some displayed text.

Solution

Use a `` tag in HTML, set the `font` property of a *TextFormat* object, or use the `font-family` property in CSS.

Discussion

You can programmatically specify the font that is used to display text by using one of several different options. You can use a `` tag if you are applying the formatting using HTML. For example:

```
field.htmlText = "<font face='Arial'>Formatted text</font>";
```

You can also use the `font` property of a *TextFormat* object. You can assign to this property the string name of the font (or fonts) that should be used:

```
formatter.font = "Arial";
```

And you can also define a font-family property in CSS:

```
p {
  font-family: Arial;
}
```

The font specified must be available on the computer on which the movie is running. Because some computers may not have the preferred font installed, you can specify multiple font names separated by commas, as follows:

```
formatter.font = "Arial, Verdana, Helvetica";
```



The preceding example uses a *TextFormat* object. However, you can specify a comma-delimited list of fonts by using any of the techniques described.

The first font is used unless it cannot be found, in which case the Flash Player attempts to use the next font in the list.

If none of the specified fonts are found, the default system font is used. You can optionally specify a *font group*. Font groups use the default device font within a category. A font group is often preferable to allowing Flash Player to use the default system font, since at least you'll know what general characteristics the font will have. There are three font groups: *_sans*, *_serif*, and *_typewriter*. The *_sans* group is generally a font such as Arial or Helvetica, the *_serif* group is generally a font such as Times or Times New Roman, and the *_typewriter* group is generally a font such as Courier or Courier New.

See Also

Recipes *9.13*, *9.15*, and *9.17*

Embedding Fonts

Problem

You want to ensure that text displays properly, even if the intended font isn't installed on the user's computer.

Solution

Embed the font by using the `[Embed]` metatag. Then set the text field's `embedFonts` property to true, and apply the font to the text field using a `` tag, a *TextFormat* object, or CSS.

Discussion

You should embed fonts if you want to ensure that text displays using the intended font, even if the user's computer does not have that font installed. To embed a font, use the `[Embed]` metatag. The `[Embed]` metatag should appear in an ActionScript file outside the class declaration. You can embed either TrueType fonts or system fonts. To embed a TrueType font with the `[Embed]` metatag use the following syntax:

```
[Embed(source="pathToTtfFile", fontName="FontName", mimeType="application/x-font-truetype")]
```

The path to the TrueType font file can be relative or absolute, such as in the following example:

```
[Embed(source="C:\\Windows\\Fonts\\Example.ttf", fontName="ExampleFont", mimeType="application/x-font-true")]
```

The *fontName* attribute value is how you refer to the font from CSS or ActionScript.

The syntax for embedding system fonts is similar, except that it uses a *systemFont* attribute rather than a source attribute. The *systemFont* attribute value is the name of the system font you want to embed. The following embeds Times New Roman:

```
[Embed(systemFont="Times New Roman", fontName="Times New Roman", mimeType="application/x-font-truetype")]
```



The preceding example uses the same name for *fontName* as the actual system font name.

Once you've embedded the font, the next step is to tell the text field to use the embedded font. To do that, simply set the `embedFonts` property of the text field to `true`. By default, the property is `false`, which means that Flash uses device fonts. By setting the `embedFonts` property to `true`, the text field can use embedded fonts only. If you try to assign a device font to a text field with `embedFonts` set to `true`, nothing is displayed:

```
field.embedFonts = true;
```

Once you've enabled embedded fonts, your next step is to tell the text field which embedded font to use; you do this with a `` tag, a *TextFormat* object, or with CSS. For example, if the *fontName* value is Times New Roman, your code might look like the following:

```
formatter.font = "Times New Roman";
```

The following example sets the font by using a font tag:

```
field.htmlText = "<font family='Times New Roman'>Example</font>";
```

And the following illustrates how to set the font by using CSS:

```
var css:StyleSheet = new StyleSheet( );  
css.setStyle("p", {fontFamily: "Times New Roman"});  
field.htmlText = "<p>Example</p>";
```

You cannot specify a comma-delimited list of fonts if `embedFonts` is set to `true`.

See Also

Recipes [9.13](#) and [9.16](#)

Creating Text that Can Be Rotated

Problem

You want to make sure that some text continues to display, even when it's rotated.

Solution

Use embedded fonts.

Discussion

Unless you specifically use an embedded font, text fields use device fonts. For most situations, this is perfectly workable. However, in situations in which you want to rotate a text field or its parent container (e.g., you want to place a label next to an object vertically), you must use embedded fonts. Device fonts disappear when rotating a text field.

See Also

Recipe 9.17 explains how to embed fonts.

Displaying Unicode Text

Problem

You want to display Unicode text in your application, possibly including non-English characters.

Solution

Load the text from an external source. Optionally, use Unicode escape sequences for the characters within the assignment to the text field's text property.

Discussion

If you want to display Unicode text in a text field, there are several ways in which you can accomplish that:

- Load the Unicode text from an external source (e.g., a text file, XML document, database).
- Use the character directly within the ActionScript code as a string value.
- Use a Unicode escape sequence.

If you load the text from an external source, you can load Unicode text and assign it to a text field. For more information regarding loading content from external sources, see *Chapter 20*.

Assuming you're using an editor that supports Unicode (such as Flex Builder), you can add the character directly within the code. Optionally, if you know the escape sequence for the character, you can assign it to a text field's text property. All Unicode escape sequences in ActionScript begin with `\\u` and are followed by a four-digit hexadecimal number. The escape sequences must be enclosed in quotes. The following example displays a registered mark (®) in both ways:

```
field.text = "Add a registered mark directly (®) or with a Unicode  
escape sequence (\\u00AE)";
```



If you want a Unicode character reference online, under Windows, you can open up the Character Map utility using Start Programs Accessories System Tools Character Map. On Mac OS X, first enable the Input Menu in System Preferences International Input Menu, and turn on the checkbox next to Character Palette. To open the Character Palette, go to the menu bar, select the flag for your country (e.g., the Stars and Stripes for the U.S.), and select Show Character Palette.

See Also

Table A-1 in the Appendix lists the Unicode escape sequences.

Assigning Focus to a Text Field

Problem

You want to use ActionScript to bring focus to a text field.

Solution

Use the *Stage.focus* property.

Discussion

Use the *Stage.focus* property to programmatically assign focus to a specific text field. Every display object has a *stage* property that references the *Stage* instance. Therefore, from a class that extends a display object class (*Sprite*, *MovieClip*, etc.) the following code assigns focus to a text field called *field*:

```
stage.focus = field;
```

When an *.swf* first loads in a web browser, it does not have focus. Therefore, you must move focus to the Flash Player before you can programmatically assign focus to an element of the Flash application. The following is a working example that uses a sprite button to assign focus to a text field:

```
package {

    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.text.TextFieldType;
    import flash.events.MouseEvent;

    public class TextExample extends Sprite {

        public function TextExample() {
            var field:TextField = new TextField();
            field.border = true;
            field.background = true;
            field.type = TextFieldType.INPUT;
            addChild(field);
            var button:Sprite = new Sprite();
            button.graphics.lineStyle();
            button.graphics.beginFill(0xFFFFFF);
            button.graphics.drawRect(0, 0, 100, 50);
            button.graphics.endFill();
            button.addEventListener(MouseEvent.CLICK, onClick);
            button.y = 100;
            addChild(button);
        }
    }
}
```

```

        private function onClick(event:MouseEvent):void {
            stage.focus = TextField(getChildAt(0));
        }
    }
}

```

To remove focus from a text field you should assign *Stage.focus* the null value:

```
stage.focus = null;
```

Selecting Text with ActionScript

Problem

You want to highlight a portion of the text within a text field.

Solution

Use the *TextField.setSelection()* method.

Discussion

The *TextField.setSelection()* method highlights a portion of the text in the text field. The *setSelection()* method takes two parameters:

startIndex

The beginning, zero-relative index of the text to highlight.

endIndex

The index of the character after the text to highlight.

For the selection to work, the text field must have focus, which you can set by using *Stage.focus*, as discussed in *Recipe 9.20*:

```

stage.focus = field;           // Set the focus to the text field
field.text = "this is example text"; // Set the text value
field.setSelection(0, 4);      // Highlight the word "this"

```

Use the read-only *selectionBeginIndex* and *selectionEndIndex* properties to retrieve the indices of the selected character range.

See Also

Recipes *9.20* and *9.23*

Setting the Insertion Point in a Text Field

Problem

You want ActionScript to set the insertion point for a text field.

Solution

Use the `TextField.setSelection()` method.

Discussion

You can use `TextField.setSelection()` to set the cursor position in a text field by setting the beginning and ending index parameters to the same value. This example sets the cursor position in the text field, assuming it has focus:

```
// Positions the insertion point before the first character
field.setSelection(0, 0);
```

You can retrieve the index of the cursor position with the read-only `caretIndex` property:

```
trace(field.caretIndex);
```

See Also

Recipe 9.21

Responding When Text Is Selected or Deselected

Problem

You want to perform a task when a text field is selected or deselected.

Solution

Listen for the `focusIn` and `focusOut` events.

Discussion

Text fields dispatch `focusIn` events when focus is shifted to the field and they dispatch `focusOut` events when focus is shifted away from the field. The events dispatched in both cases are `flash.events.FocusEvent` objects. The `FocusEvent` class defines a `relatedObject` property. In the case of `focusIn` events, the `relatedObject` property is a reference to the object that just had focus. In the case of `focusOut` events, the `relatedObject` property is a reference to the object that just received focus. Use the `flash.events.FocusEvent` constants of `FOCUS_IN` and `FOCUS_OUT` when registering listeners:

```
field.addEventListener(FocusEvent.FOCUS_IN, onFocus);
```

The `focusIn` and `focusOut` events both occur after the focus has already changed. They are non-cancelable events. If you want to be able to cancel the events, you must listen for events that occur before `focusIn` and `focusOut` are dispatched. The `keyFocusChange` and `mouseFocusChange` events are cancelable events that occur when the user attempts to move focus from a text field by way of the keyboard or mouse, respectively. You can register listeners by using the `FocusEvent` constants of `KEY_FOCUS_CHANGE` and `MOUSE_FOCUS_CHANGE`. Use the `FocusEvent.preventDefault()` method to cancel the default behavior. The following example disallows using the Tab key to move from `field1` to `field2` if `field1` doesn't have any text:

```

package {

import flash.display.Sprite;
import flash.text.TextField;
import flash.text.TextFieldType;
import flash.events.FocusEvent;

public class Text extends Sprite {

private var _field1:TextField;
private var _field2:TextField;

public function Text() {
    _field1 = new TextField();
    _field1.border = true;
    _field1.background = true;
    _field1.type = TextFieldType.INPUT;
    addChild(_field1);
    _field1.addEventListener(FocusEvent.KEY_FOCUS_CHANGE, onKeyFocus);
    _field2 = new TextField();
    _field2.border = true;
    _field2.background = true;
    _field2.type = TextFieldType.INPUT;
    addChild(_field2);
    _field2.y = 100;
}

private function onKeyFocus(event:FocusEvent):void {
    if(_field1.text == "") {
        event.preventDefault();
    }
}
}
}

```

Responding to User Text Entry

Problem

You want to perform a task when the content of a text field is modified by user input.

Solution

Listen for the *textInput* event.

Discussion

You can specify actions to be performed each time the content of a text field is changed by user input, whether that change is deleting or cutting characters, typing in a character, or pasting characters. When a user makes any change to the value of an input text field, the text field dispatches

a *textInput* event. You can register a listener to listen for the *textInput* event using the *flash.events.TextEvent.TEXT_INPUT* constant:

```
field.addEventListener(TextEvent.TEXT_INPUT, onTextInput);
```

The *textInput* event is a *TextEvent* object, and it is cancelable. The *TextEvent* class defines a *text* property, which contains the value of the text entered by the user. The following example ensures that the first character the user types in a text field is not an “a”:

```
package {

    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.text.TextFieldType;
    import flash.events.TextEvent;
    import flash.events.TextEvent;

    public class Text extends Sprite {

        private var _field:TextField;

        public function Text() {
            _field = new TextField();
            _field.border = true;
            _field.background = true;
            _field.type = TextFieldType.INPUT;
            addChild(_field);
            _field.addEventListener(TextEvent.TEXT_INPUT, onTextInput);
        }

        private function onTextInput(event:TextEvent):void {
            if(event.text == "a" && _field.length == 0) {
                event.preventDefault();
            }
        }
    }
}
```

See Also

Recipe 9.12

Adding a Hyperlink to Text

Problem

You want to hyperlink some of the text displayed in a text field.

Solution

Use HTML `<a href>` tags within the object's `htmlText` property. Alternatively, use a *TextFormat* object with a value assigned to the `url` property.

Discussion

Both solutions to this problem require that you set the text field's `html` property to `true`:

```
field.html = true;
```

If you want to use HTML to add a hyperlink, add an `<a href>` tag to the text field's `htmlText` property, as follows:

```
field.htmlText = "<a href='http://www.rightactionscript.com'>Website</a>";
```

You can add a target window into which to open the link by adding a `target` attribute to the `<a href>` HTML tag. For example:

```
field.htmlText = "<a href='http://www.rightactionscript.com' target='blank'>Website</a>";
```

When text is hyperlinked in Flash, the mouse cursor changes to a hand when it is over the linked text. Flash does not inherently provide any indication that the text is linked, unlike most HTML browsers (which use an underline and color change). For this reason, it is helpful to add HTML markup that underlines and colors the linked text:

```
var htmlLink:String = "<font color='#0000FF'><u>";  
htmlLink += "<a href='http://www.rightactionscript.com'>Website</a>";  
htmlLink += "</u></font>";  
field.htmlText = htmlLink;
```

You can accomplish the same tasks without HTML by using a *TextFormat* object. The *TextFormat* class includes a `url` property for just this purpose. Assigning the URL to the `url` property links the formatted text; for example:

```
field.text = "Website";  
var formatter:TextFormat = new TextFormat();  
formatter.url = "http://www.rightactionscript.com/";  
field.setTextFormat(formatter);
```

If you want to specify a target window into which the link opens, you can set the value of the *TextFormat* object's `target` property, as follows:

```
field.text = "Website";  
var formatter:TextFormat = new TextFormat();  
formatter.url = "http://www.rightactionscript.com/";  
formatter.target = "_blank";  
field.setTextFormat(formatter);
```

As with the HTML technique, when using a *TextFormat* object to create a hyperlink, Flash does not offer any indication as to the link's presence other than the hand cursor when it is moused over. You can add color and/or an underline to the linked text to provide the user with the indication that it is a link. You should use the *TextFormat* object's `color` and `underline` properties for this purpose:

```
field.text = "Website";  
var formatter:TextFormat = new TextFormat();
```

```
formatter.color = 0x0000FF;
formatter.underline = true;
formatter.url = "http://www.rightactionsript.com/";
field.setTextFormat(formatter);
```

You can use either of the techniques in this recipe to add links that point not only to *http* and *https* protocols, as shown in the examples, but also to link to other protocols. For example, you can use the same techniques to open a new email message:

```
field.text = "email";
var formatter:TextFormat = new TextFormat();
formatter.color = 0x0000FF;
formatter.underline = true;
formatter.url = "mailto:joey@person13.com";
field.setTextFormat(formatter);
```

Be aware, however, that many other types of links (such as *mailto* links) only work when the movie is played in a web browser in which a default client for the protocol has been defined.

Using CSS, you can apply advanced formatting to `<a href>` tags by using the `a:link`, `a:active`, and `a:hover` styles, as shown in the following example:

```
var css:StyleSheet = new StyleSheet();
css.parseCSS("a {color: #0000FF;} a:hover {text-decoration: underline;}");
field.styleSheet = css;
field.html = true;
field.htmlText = "<a href='http://www.rightactionsript.com'>Website</a>";
```

See Also

Recipes *9.8* and *9.15*

Calling ActionScript from Hyperlinks

Problem

You want to call an ActionScript method when the user clicks a hyperlink.

Solution

Use the *event* protocol and listen for link events.

Discussion

Many applications require calling ActionScript when the user clicks a hyperlink. With ActionScript 3.0, however, it is a very simple task. First, you must define the hyperlink to use the event protocol, as follows:

```
field.htmlText = "<a href='event:http://www.rightactionsript.com'>Website</a>";
```

When you use the *event* protocol, the default behavior does not occur. When the user clicks on a hyperlink, it normally opens the URL in a web browser. However, when you use the *event* protocol, an event is dispatched, which means you have to register a listener to listen for that event. The

event type is *link*, and you can use the *flash.events.TextEvent.LINK* constant when registering a listener:

```
field.addEventListener(TextEvent.LINK, onClickHyperlink);
```

The *event* that is dispatched is a *flash.events.TextEvent* type. The *text* property of the *event* object contains the value of the href attribute minus the *event* protocol. That means that in the preceding example, the value of the *event* object's *text* property is *http://www.rightactionscript.com*. Since the hyperlink does not attempt to open a URL in a browser window when using the *event* protocol, you don't have to use a valid URL for the href value. You can use any string that would be useful in determining which hyperlink the user clicked.

Working with Advanced Text Layout

Problem

You want to work with advanced text layout.

Solution

Use the *numLines* property and the *getCharBoundaries()*, *getCharIndexAtPoint()*, *getFirstCharInParagraph()*, *getLineIndexAtPoint()*, *getLineIndexOfChar()*, *getLineLength()*, *getLineMetrics()*, *getLineOffset()*, *getLineText()*, and *getParagraphLength()* methods.

Discussion

In versions of Flash Player up to and including Flash Player 8, it was difficult to control and read text layout with much precision. Starting with Flash Player 8.5, though, the *TextField* class defines an API for more precise reading of text layout.

The *TextField* class defines two methods for retrieving information about characters from text. The *getCharBoundaries()* method returns a *flash.geom.Rectangle* object that defines the boundaries of the character at the index specified by the parameter. The *getCharIndexAtPoint()* method returns the index of a character at the *x* and *y* coordinates specified by the parameters. The following example uses *getCharIndexAtPoint()* and *getCharBoundaries()* to highlight a character when the user clicks on it:

```
package {

    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.events.MouseEvent;
    import flash.geom.Rectangle;

    public class Text extends Sprite {

        private var _field:TextField;
        private var _highlight:Sprite;

        public function Text() {
            _field = new TextField();
            _field.border = true;
```

```

        _field.background = true;
        _field.multiline = true;
        _field.wordWrap = true;
        _field.selectable = false;
        _field.width = 400;
        _field.height = 400;
        addChild(_field);
        _field.text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi tortor purus, aliquet a,
        _field.addEventListener(MouseEvent.CLICK, onClick);
        _highlight = new Sprite();
        addChild(_highlight);
    }

    private function onClick(event:MouseEvent):void {
        var index:int = _field.getCharIndexAtPoint(mouseX, mouseY);
        var rectangle:Rectangle = _field.getCharBoundaries(index);
        _highlight.graphics.clear();
        _highlight.graphics.lineStyle(0, 0, 0);
        _highlight.graphics.beginFill(0x00FFFF, .25);
        _highlight.graphics.drawRect(rectangle.x, rectangle.y, rectangle.width, rectangle.height);
        _highlight.graphics.endFill();
    }

}
}
}

```

The *TextField* class also defines a property and methods for retrieving information about lines of text. The *numLines* property tells you how many lines of text a text field contains. The *getLineIndexAtPoint()* method returns the index of a line at the coordinates specified as the parameters passed to the method. The *getLine-IndexOfChar()* returns the line index of the line that contains the character with the index specified by the parameter passed to the method. The *getLineLength()* method returns the number of characters in a line specified by its line index. The *getLineText()* method returns the text contained within a line at a specified line index. The *getLineOffset()* method returns the character index the first character of a line at a specified line index. The *getLineMetrics()* method returns a *flash.text.TextLineMetrics* object for a line with a specified line index. The *TextLineMetrics* class defines ascent, descent, height, width, leading, and *x* properties that describe the line of text.

There are two methods for retrieving information about paragraphs. The *getFirstCharInParagraph()* method returns the character index of the first character in a paragraph that also contains the character at the index specified in the parameter. The *getParagraphLength()* method returns the number of characters in a paragraph that contain the character at the index specified by the parameter.

The following example uses most of the methods discussed in this recipe to highlight a paragraph when the user clicks a character:

```

package {

    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.events.MouseEvent;
    import flash.geom.Rectangle;
    import flash.text.TextLineMetrics;

```

```

public class Text extends Sprite {

    private var _field:TextField;
    private var _highlight:Sprite;

    public function Text() {
        _field = new TextField();
        _field.border = true;
        _field.background = true;
        _field.multiline = true;
        _field.wordWrap = true;
        _field.selectable = false;
        _field.width = 400;
        _field.height = 400;
        addChild(_field);
        _field.text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi tortor purus, aliquet a, or
        _field.addEventListener(MouseEvent.CLICK, onDoubleClick);
        _highlight = new Sprite();
        addChild(_highlight);
    }

    private function onDoubleClick(event:MouseEvent):void {
        var index:int = _field.getCharIndexAtPoint(mouseX, mouseY);
        var startIndex:int = _field.getFirstCharInParagraph(index);
        var stopIndex:int = startIndex + _field.getParagraphLength(index);
        var startLine:int = _field.getLineIndexofChar(startIndex);
        var stopLine:int = _field.getLineIndexofChar(stopIndex - 1);
        var metrics:TextLineMetrics;
        var lineCharacter:int;
        var rectangle:Rectangle;
        _highlight.graphics.clear();
        _highlight.graphics.lineStyle(0, 0, 0);
        for(var i:int = startLine; i <= stopLine; i++) {
            lineCharacter = _field.getLineOffset(i);
            rectangle = _field.getCharBoundaries(lineCharacter);
            metrics = _field.getLineMetrics(i);
            _highlight.graphics.beginFill(0x00FFFF, .25);
            _highlight.graphics.drawRect(rectangle.x, rectangle.y, metrics.width, metrics.height);
            _highlight.graphics.endFill();
        }
    }
}

```

Applying Advanced Anti-Aliasing

Problem

You want to have more precise control over anti-aliasing of text.

Solution

Embed the font, set the `antiAliasType` property of the text field to `flash.text.AntiAliasType.ADVANCED`, and set the `gridTypeFit` and sharpness properties.

Discussion

By default, text displays with normal anti-aliasing settings. For most fonts at sizes 10 and higher the normal anti-aliasing settings make for legible text. However, for smaller font sizes and for certain fonts, the normal anti-aliasing settings make the text less than legible. In those cases, you can set the anti-alias type for the text field to advanced and use the `gridFitType` and sharpness properties to more precisely control how the text is rendered.



To enable advanced anti-alias settings for a text field, you must use embedded fonts. See [Recipe 9.17](#) for more details on how to embed fonts.

The `TextField.antiAliasType` property accepts one of the `flash.text.AntiAliasType` constants of `NORMAL` (default) or `ADVANCED`. Set the value of the property to `ADVANCED` for a text field if you want to enable more precise anti-alias settings:

```
field.antiAliasType = AntiAliasType.ADVANCED;
```

The `gridFitType` property determines how the font outlines snap to whole pixels on the screen. The possible values are the `NONE`, `PIXEL`, and `SUBPIXEL` constants of the `flash.text.GridFitType` class. The default value of `NONE` means that the text does not snap to whole pixels. That can cause text to appear blurry at smaller font sizes. The `PIXEL` setting snaps horizontal and vertical lines of the font outlines to whole pixels on the screen. The `PIXEL` setting works only when text is left-aligned. If you want center- or right-aligned text to snap to pixels, use the `SUBPIXEL` setting.

```
field.gridFitType = GridFitType.PIXEL;
```

The sharpness property ranges from -400 to 400, with a default value of 0; it determines how crisply the edges of the font outlines are rendered. The lower the number, the less sharply the fonts are rendered. The greater the number, the more sharply the fonts are rendered. If the text is not legible because it appears blurry, and you've already set the `gridFitType` to `PIXEL` or `SUBPIXEL`, then increase the sharpness value.

See Also

[Recipe 9.17](#)

Replacing Text

Problem

You want to replace text.

Solution

Use the *replaceSelectedText()* method to replace the highlighted text and *replaceText()* to replace a range of text.

Discussion

The *replaceSelectedText()* method enables you to replace the selected text in a text field. Simply pass the method the string to use as the replacement text. For the method to work, the text field must have focus:

```
_field.replaceSelectedText("new text");
```

Use the *replaceText()* method to replace text within a text string given a starting and ending index. The following replaces the text from index 100 to index 150, with the new string specified by the third parameter:

```
_field.replaceText(100, 150, "new text");
```

Retrieving a List of System Fonts

Problem

You want to retrieve a list of fonts on the user's system.

Solution

Use the static *TextField.fontList* property.

Discussion

When you want to use system fonts (rather than embedding the font or using a font group), first determine which fonts the user has installed. You can retrieve an array of system fonts on the user's computer with the *TextField.fontList* property.

```
trace(TextField.fontList);
```

Retrieving the list of available system fonts simply yields an array of strings. To apply a font to the text, you'll have to use one of the techniques discussed in *Recipe 9.16*.

Index

*

* (asterisks)

in password input fields, 6

+

+= operator, appending text, 8

-

- (dash)

pattern matching, 7

<

<a> HTML tag, 9

hyperlinks, adding to text, 31

 HTML tag, 9

 HTML tag, 9

 HTML tag, 9

embedding fonts, 23

setting fonts in text fields, 22

<i> HTML tag, 9

 HTML tag, 9

 HTML tag, 9

<p> HTML tag, 9

<textformat> HTML tag, 9

<u> HTML tag, 9

A

addChild() method

text fields, making visible, 3

advanced text layout, 33

anti-aliasing, 35

antiAliasType property, 36

appendText() method, 8

asterisks (*)

in password input fields, 6

autoSize property, 10

B

backgroundColor property, 4

backslash (\\)

pattern matching, 7

blockindent attribute (<textformat> tag), 9

border property, 4

borderColor property, 4

bottomScrollV property (text fields), 11

C

caret (^)

pattern matching, 7

caretIndex property, 28

Cascading StyleSheets (CSS), 15

embedding fonts with, 23

hyperlinks, adding to text, 32

setting fonts in text fields, 22

case-sensitivity

filtering text input, 7

color attribute (tag), 9

color CSS property, 16

color property (TextFormat object), 31

condenseWhite property, 9

Courier, 23

Courier New, 23

CSS (Cascading StyleSheets), 15

embedding fonts with, 23

hyperlinks, adding to text, 32

setting fonts in text fields, 22

D

dash (-)

pattern matching, 7

defaultTextFormat property, 21

deselected text, responding to, 28

display ActionScript property, 16

dynamic text fields, 5

E

encrypting passwords, 6

endIndex parameter

setSelection() method, 27

setTextFormat() method, 22

escaping characters, 7

event protocol, 32

F

face attribute (tag), 9

flash.display package

TextField class, 3

flash.display.TextFieldType class, 5

flash.events.Event class, 14

flash.events.FocusEvent objects, 28

flash.events.TextEvent type, 33

flash.events.TextEvent.TEXT_INPUT constant, 30

flash.geom.Rectangle object, 33

flash.net.URLLoader class, 18

flash.text.AntiAliasType.ADVANCED, 36

flash.text.GridFitType class, 36

flash.text.TextField class, 3

flash.text.TextFieldAutoSize class, 10

FocusEvent.preventDefault() method, 28

focusIn event, 28

focusOut event, 28

font groups, 23

font property (TextFormat object), 22

font-family CSS property, 16

font-size CSS property, 17

font-style CSS property, 17

font-weight CSS property, 17

fontFamily ActionScript property, 16

fonts

- embedding, 23
 - rotating, 24
 - text fields, setting in, 22
- fontSize ActionScript property, 17
- fontStyle ActionScript property, 17
- fontWeight ActionScript property, 17

G

- getCharBoundaries() method, 33
- getCharIndexAtPoint() method, 33
- getFirstCharInParagraph() method, 33
- getLineIndexAtPoint() method, 33
- getLineIndexOfChar() method, 33
- getLineLength() method, 33
- getLineMetrics() method, 33
- getLineOffset() method, 33
- getLineText() method, 33
- getParagraphLength() method, 33
- gridTypeFit property, 36

H

- HTML (Hypertext Markup Language)
 - <a> tag, 9, 31
 - tag, 9
 -
 tag, 9
 - tag, 9, 22, 23
 - <i> tag, 9
 - tag, 9
 - tag, 9
 - <p> tag, 9
 - <textformat> tag, 9
 - displaying, 8
 - fonts, setting in text fields, 22
 - formatting text, 15
- htmlText property, 9
 - hyperlinks, adding to text, 31
- http protocol, 32
- https protocol, 32
- hyperlinks, 9
 - adding to text, 30
 - calling ActionScript from, 32

I

- indent attribute (<textformat> tag), 9
- index parameter (setTextFormat() method), 21
- input fields, 5
 - filtering input, 6
 - maximum length, setting, 7
 - password, 5
 - user-input
 - formatting, 21
- insertion points, setting, 28

K

- kerning ActionScript property, 17
- kerning CSS property, 17
- keyFocusChange events, 28

L

- leading attribute (<textformat> tag), 9
- leftmargin attribute (<textformat> tag), 9
- letter-spacing CSS property, 17

- letterSpacing ActionScript property, 17

M

- mailto links, 32
- margin-left CSS property, 17
- margin-right CSS property, 17
- marginLeft ActionScript property, 17
- marginRight ActionScript property, 17
- maxChars property, 7
- maxScrollH property (text fields), 11
- maxScrollV property (text fields), 11
- mouseFocusChange event, 28
- mouseWheelEnabled property, 14

N

- numLines property, 33

O

- onTextScroll() method, 14
- outlines, creating around text fields, 4

P

- parseCSS() method, 18
- password input fields, 5
- password property, 5

R

- regular expressions, 7
- relatedObject property, 28
- replaceSelectedText() method, 37
- replaceText() method, 37
- restrict property, filtering text input with, 6
- rightmargin attribute (<textformat> tag), 9

S

- scroll events, responding to, 14
- scrollH property (text fields), 11
- scrollV property (text fields), 11
- Secure Socket Layer (SSL), 6
- selected text, responding to, 28
- setSelection() method, 27
- setStyle() method, 18
- setTextFormat() method, 15
 - formatting portions of text, 21
- sharpness property, 36
- size attribute (tag), 9
- SSL (Secure Socket Layer), 6
- Stage.focus property, 26
- startIndex parameter
 - setSelection() method, 27
 - setTextFormat() method, 22
- StyleSheet object, 14
- StyleSheet property, 14
- systemFont attribute, 24

T

- tabstops attribute (<textformat> tag), 9
- TextField.setTextFormat() method, 14
- text, 3
 - advanced layout, working with, 33
 - anti-aliasing, applying advanced, 35
 - backgrounds, creating, 4

- displaying, 8
- fonts, 22
 - embedding, 23
- formatting, 14
 - portions of, 21
- HTML, displaying, 8
- hyperlinks, 30
 - calling ActionScript from, 32
- input fields, 5
 - filtering, 6
 - maximum length, setting, 7
 - password, 5
- insertion points, setting, 27
- outlines around, 4
- replacing, 37
- responding to user entries, 29
- rotating, 24
- scroll events, responding, 14
- scrolling programmatically, 11
- selected/deselected, responding to, 28
- selecting with ActionScript, 27
- sizing fields, 10
- Unicode, displaying, 25
- user-input, formatting, 21
 - whitespace, condensing, 9
- text-align CSS property, 17
- text-decoration CSS property, 17
- text-indent CSS property, 17
- textAlign ActionScript property, 17
- textDecoration ActionScript property, 17
- TextField class, 3
- TextField.password property, 6
- TextField.restrict property, 6
- TextField.setSelection() method, 27, 28
- TextFormat object, 14
- textFormat parameter (setTextFormat() method), 21, 22
- textIndent ActionScript property, 17
- textInput event, 29

- Times, 23
- Times New Roman, 23
- type property, 5

U

- underline property (TextFormat object), 31
- Unicode, 25
- user input
 - fields, 5
 - formatting, 21
 - responding to, 29

W

- whitespace (text)
 - condensing, 9

\

- \\, 23
- \\\\ (backslash)
 - pattern matching, 7

^

- ^ (caret)
 - pattern matching, 7

_

- _sans font group (sans font group), 23
- _serif font group, 23
- _typewriter font group, 23

- character, 7