,ch18.17514 Page 383 Monday, April 16, 2001 1:55 PM

18

On-Screen Text Fields

Because Flash is fundamentally a visual environment, movies often present on-screen information to users. Similarly, because Flash is an interactive environment, movies often retrieve information from users through a GUI. To display the value of a variable on screen or to allow a user to type data into a Flash movie, we use *text fields*.

Text fields provide a means of both setting and retrieving the values of variables that have a visual representation. Text fields come in two varieties—dynamic text fields, which we use to display information to the user, and user-input text fields, which we use to retrieve information from the user.

Dynamic Text Fields

A dynamic text field is like a variable viewport—it displays the value of a specified variable as a text string. Dynamic text fields are created using the Text tool in Flash. However, unlike regular static text, the content of a dynamic text field is connected to a variable and can be changed or retrieved via ActionScript.

By retrieving a text field's value, we can capture on-screen information for use in a script. By setting a text field's value, we cause that value to display on screen.

Creating a Dynamic Text Field

To make a new dynamic text field, follow these steps:

1. Select the Text tool.
2. Click and drag a rectangle on the Stage. The outline that you create will define the size of the new text field.

,ch18.17514 Page 384 Monday, April 16, 2001 1:55 PM

3. Select Text → Options. The Text Options panel appears.
4. In the Text Type menu, choose Dynamic Text (the other options are Static Text and Input Text).
5. Under Variable, type a name for the dynamic text field, following the rules we learned in Chapter 2, *Variables*, for constructing legal variable names.

After creating a dynamic text field, you'd normally set the new field's options, as described later.

Changing the Content of a Dynamic Text Field

Once a text field is created, we can use it to display a value on the screen. For example, if we create a dynamic text field named `myText`, we can set the content of that text field using the following statements:

```
myText = 10;                        // Display a number in the text field myText
myText = "Welcome to my web site"; // Display a string instead

var msg = "Please make a selection";
myText = msg;                       // Display the value of msg in myText
```

Whenever the value of the variable `myText` changes, the content of the `myText` dynamic text field updates to reflect the change. However, before a value is sent to a dynamic text field for display, it is first converted to a string. The actual content is therefore governed by string-conversion rules described in Table 3-2.

Like normal variables, text fields are tied to the movie clip timeline on which they reside. To access a dynamic text field in a remote movie clip timeline, we use the techniques described in Chapter 2 under "Accessing Variables on Different Timelines."

Retrieving the Value of a Dynamic Text Field

We can retrieve a dynamic text field's value by simply using its name. For example, if `myTextField` were a dynamic text field in our movie, we could retrieve and assign its value to another variable like so:

```
welcomeMessage = myTextField;
```

Text field assignment and retrieval are often combined in one statement. You can use the `+=` operator to append text to a text field's current contents:

```
// Set a text field's value
myTextField = "Today's Headlines...";
// Create a new message
var newText = "Update! The Party Has Been Cancelled!"
// Add the new message to the existing text field content
myTextField += newText;
```

,ch18.17514 Page 385 Monday, April 16, 2001 1:55 PM

User-Input Text Fields

User-input text fields differ from dynamic text fields only in that they may be edited by the user while the movie is playing. That is, a user can type into a user-input text field to change its value. ActionScript can then retrieve and manipulate the user-entered value. User-input text fields are useful for guest books, order forms, password-entry fields, or anywhere you request information from the user.

Creating a User-Input Text Field

To create a user-input text field, follow the same steps described earlier under "Creating a Dynamic Text Field," but choose Input Text instead of Dynamic Text from the Text Type menu.

Changing the Content of an Input Text Field

Like dynamic text fields, user-input text fields may be changed at any time simply by setting the value of the named text field with an assignment statement:

```
myInputText = "Type your name here";
```

Because user-input text fields are normally used to accept data rather than display data, we don't usually set their contents except to provide a default value for the user's input.

Retrieving and Using the Value of an Input Text Field

You can retrieve the value of a text field by simply referring to it by name in a script. For example, to display the value of an input text field called `myInput`, use:

```
trace(myInput);
```

Because data entered by the user into a user-input text field is always a string datatype, we should convert it explicitly before using it in a non-string context, as demonstrated in this simple calculator example that totals two user-input text fields:

```
// Suppose the user sets myFirstInput to 5 and mySecondInput to 10,
then we total the fields
// WRONG: "Adding" the fields together sets myOutput to "Total: 510"
// because the + operator is interpreted as a string concatenator
myOutput = "Total: " + (myFirstInput + mySecondInput);
// RIGHT: Convert the fields to numbers first in order to get the right result
myOutput = "Total: " + (parseFloat(myFirstInput) + parseFloat(mySecondInput));
```

,ch18.17514 Page 386 Monday, April 16, 2001 1:55 PM

User-Input Text Fields and Forms

User-input text fields are often used for fill-in forms submitted to a server-side application such as a Perl script. When variables are submitted to a server via *loadVariables()*, only the variables defined in the current movie clip are sent. Hence, when a form contains user-input text fields, the fields should be stored in a single, separate movie clip so that they can be submitted easily as a group to a server. See Chapter 17, *Flash Forms*, and Part III, *Language Reference* for additional details on *loadVariables()*.

Text Field Options

Dynamic text fields and user-input text fields share most, but not all, options used to configure their display and input features. Figure 18-1 shows the Text Options panel for user-input and dynamic text fields.

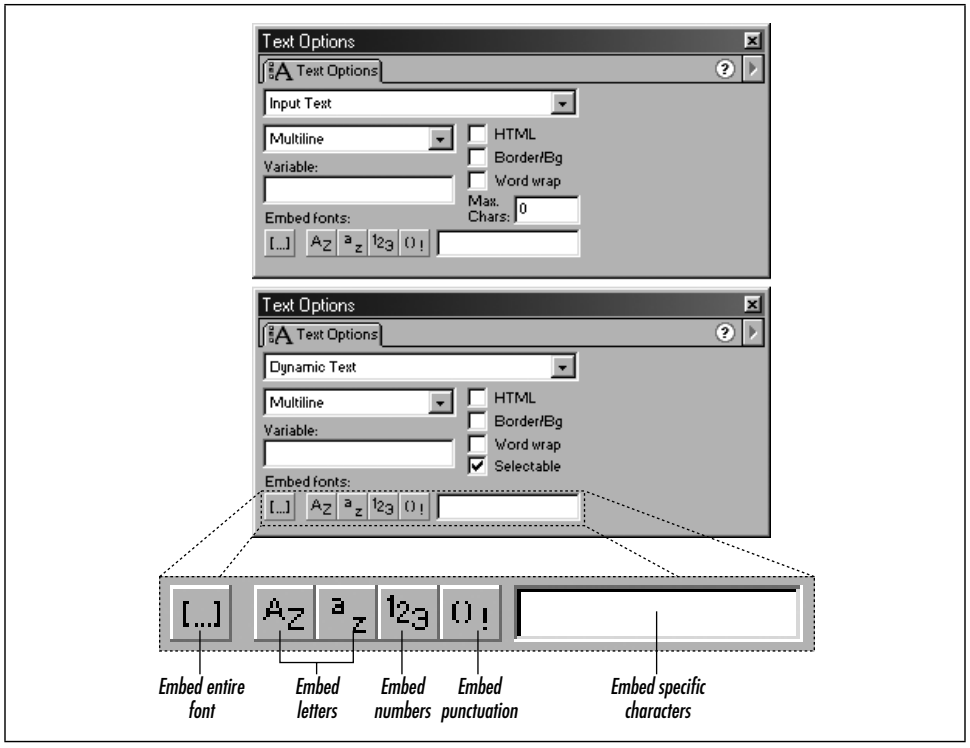

Figure 18-1. The Text Options panel

,ch18.17514 Page 387 Monday, April 16, 2001 1:55 PM

Line Display

To set the layout and input style of a text field or to disguise the user's input, we use the Line Display menu. There are three Line Display options:

Single Line

The Single Line option prevents users from entering more than one line of text in the field, effectively disabling the Enter key during text entry.

The Single Line setting also affects text entered without line breaks in the authoring tool; text that "soft wraps" automatically during authoring will not wrap in the Player. Instead, the text will be displayed on one line, even if it overflows the field to the right. Hard carriage returns entered during authoring, however, are unaffected by the Single Line setting; text with hard returns will display in the Player as it appeared in the authoring tool.

The Single Line option applies primarily to user-input text fields. When used with dynamic text fields, its behavior is the same as that of a Multiline dynamic text field unless the Word Wrap option is also selected.

Both the `\n` escape sequence and the `newline` keyword insert line breaks despite the Single Line setting. For example, if we set a text field variable to the value `"this is\na test"`, the text `"this is"` and `"a test"` will be displayed on separate lines.

Multiline

The Multiline setting allows users to enter more than one line of text in the text field. Carriage returns are permitted in user input when Multiline is selected.

Multiline has no effect on the output of a dynamic text field unless used in combination with the Word Wrap option. If Word Wrap is not on, Multiline text fields behave exactly like Single Line text fields.

Password

The Password option is used to conceal characters entered into a form and applies only to user-input text fields. It behaves like a Single Line text field except that all characters, including spaces, are masked with asterisks (*). For example, the words `"hi there"` would be displayed as `"********"`.

It is possible to cause the words in a Password text field to wrap due to a quirk in the Flash interface. If you set Line Display to Multiline and select Word Wrap, and then set the Line Display to Password, the Word Wrap setting will be retained. However, Multiline password entry is not advised as it is confusing to most users.

,ch18.17514 Page 388 Monday, April 16, 2001 1:55 PM

Variable

The Variable option in the Text Options panel is used to name a dynamic or user-input text field. Text fields *must* be named in order to be manipulated with ActionScript. When naming text fields, follow the rules for constructing legal variable names described in Chapter 2, *Variables*, and Chapter 14, *Lexical Structure*.

Border/Bg

When set in the Text Options panel, the Border/Bg option causes a black outline to be displayed around a text field and a white background to be placed behind the viewable region of the field. These colors and styles are not customizable. To produce a custom background for a text field, unset the Border/Bg option and manually draw a shape behind the text field.

Word Wrap

When used in conjunction with the Multiline setting of the Line Display option, Word Wrap soft wraps lines of text that would otherwise exceed the width of the field. This setting applies to both text entered by users and text displayed via ActionScript.

If you set the Word Wrap option while Multiline is selected and then choose Single Line, the Word Wrap setting will still apply. Be sure to unset Word Wrap if you do not want text to wrap at the end of each line.

Selectable

The text in a dynamic text field may be selected by the user only if the field's Selectable option is set. Even then, the dynamic text may be copied but not cut or edited. User-input text fields are always selectable, and their text can always be copied, cut, or edited.

Text must be copied, cut, and pasted via the Windows right-click context menu in Flash (or Ctrl-click on Macintosh). Keyboard accelerators such as Ctrl-C and Ctrl-V (in Windows) or Cmd-C and Cmd-V (on Macintosh) are ignored.

Max Characters

Used only with user-input text fields, the Max Characters option limits the amount of text a user can enter into a text field. By default, Max Characters is set to `0`,

,ch18.17514 Page 389 Monday, April 16, 2001 1:55 PM

which allows an unlimited amount of text to be entered. Other settings allow the specified number of characters to be entered.

Max Characters is often used with forms that require a certain format for their data. For example, we could use it to limit a date entry to a two-digit day, a two-digit month, and a four-digit year.

Embed Fonts

By default, all dynamic and input text fields use *device fonts* (the fonts installed on the user's system). When device fonts are used, if the user has the font specified in the Character panel for the text field, the text appears on the user's system as it appeared during authoring (but without antialiasing). If the user does not have the font, an alternative font is used, which is not always desirable.

To ensure that text will render in a particular font, we embed that font in the movie using the Embed Fonts options, shown enlarged in Figure 18-1.

We can:

- Embed the entire font using the `[...]` button.
- Embed any combination of the letters, numbers, or punctuation using the `AZ`, `az`, `123`, and `()!` buttons.
- Embed specific characters by typing them into the field provided.

Embedding a complete Roman font typically adds 20–30 KB to a movie (Asian fonts can be much larger). If we're using only a subset of the characters, we can save file space by embedding only the characters we need. Characters that we don't embed cannot be entered by the user or displayed via ActionScript. We can use this to our advantage to restrict text entry to certain characters.

You must set the Embed Fonts option separately for *every* text field that uses a particular font, even if multiple text fields use the same font. However, file size is not affected when multiple text fields embed the same font—only one copy of the font is downloaded with the movie. To apply the same Embed Fonts option to many text fields at once, select the desired fields and then set the Embed Fonts option as usual.

Text displayed in text fields with embedded fonts is always antialiased. Therefore, using embedded fonts with sizes smaller than 10 point is not recommended, because antialiased text becomes unreadable below 10 point in most fonts. To prevent a font from antialiasing, use device fonts (i.e., system fonts) by unselecting all Embed Fonts options. Device fonts are never antialiased.

,ch18.17514 Page 390 Monday, April 16, 2001 1:55 PM

The contents of a text field that is rotated or masked will not show up on screen unless its font is embedded. That is, you can't rotate or mask text fields that use device fonts.

See "Using HTML as Output" later in this chapter for more important details on fonts in text fields.

Text Field Properties

When the body of text in a text field spans more lines than can be accommodated by the physical viewable region of the field, extra lines of text are hidden. The extra lines, however, are still part of the text field. To view those lines, we can click in the field and press the down arrow key until the excess lines appear. Obviously, we can't expect users to use the arrow keys to scroll through text in a text field. Instead, we should provide buttons that scroll the text using the `scroll` and `maxscroll` properties, both of which use an index number to refer to the lines in a text field. The top line is number 1, and line numbers increase for every line in the text field, including those that exceed the viewable boundaries of the field. Figure 18-2 shows a sample text field's line index values.

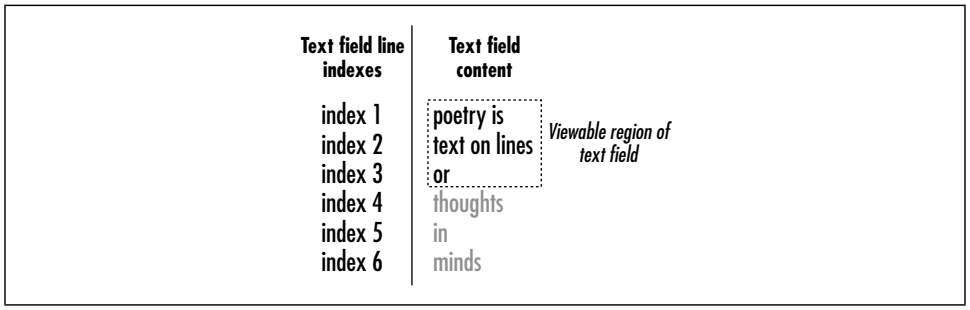

Figure 18-2. Text field line indexes

The scroll Property

The `scroll` property represents the line number of the topmost line currently displayed in a text field and can be accessed using *`textFieldName`*`.scroll`.

When a text field contains more lines than it can display at once, we can change which lines are shown in the field's viewable region by setting the `scroll` property. For example, if we were to set the `scroll` property of the text field shown in Figure 18-2 to 3, the text field would display:

,ch18.17514 Page 391 Monday, April 16, 2001 1:55 PM


```
or
thoughts
in
```

The maxscroll Property

The `maxscroll` property tells us how far a field can be scrolled (i.e., how far it must be scrolled until the last line becomes visible). It is always the index of the field's last line minus the number of lines that can be displayed in the viewable region at once, plus one. For example, the `maxscroll` property of the text field in Figure 18-2 would be 4 (the last line is 6, minus 3 lines in viewable region, plus 1). Note that `maxscroll` is *not* equal to the number of text lines.

We can retrieve (but not set) the `maxscroll` property using *`textFieldName`*`.maxscroll`.

Typical Text-Scrolling Code

In combination, the `scroll` and `maxscroll` properties can be used to scroll a text field. This code scrolls text down one line for each click of a button:

```
on (press) {
  if (textField.scroll < textField.maxscroll) {
    textField.scroll++;
  }
}
```

And here's how we scroll text up one line with each click:

```
on (press) {
  if (textField.scroll > 1) {
    textField.scroll--;
  }
}
```

For an example of simple scroll buttons used in a movie, download the sample scrollers posted at the online Code Depot.

Build 30 of the Flash 5 Player, released with the Flash 5 authoring tool, had a text field display bug. When antialiased text fields were scrolled, remnants of the scrolled text did not always disappear. To work around the problem, place a border around your text field to cover up the residual text. This bug was fixed in build 41 of the Flash 5 Player, released in December 2000. Use the global *getVersion()* function to check the version of the Player.

The _changed Event

In Flash 4 and Flash 5, changes to the content of a user-input text field can be detected via the undocumented *_changed* event. The *_changed* event triggers a

,ch18.17514 Page 392 Monday, April 16, 2001 1:55 PM

specially-named Flash 4-style subroutine whenever the user adds text to or deletes text from a user-input text field. To create a *_changed* event for a text field, follow these steps:

1. Create an input text field on any timeline.
2. Name the text field `myField`.
3. On the same timeline as the text field, label a frame `myField_changed`.
4. Attach any code to the frame `myField_changed`. For example:

```
trace("myField was changed");
```

5. Export the movie using Control → Test Movie.
6. Type characters into the `myField` text field. The code on the frame `myField_changed` is executed, and "myField was changed" appears in the Output window.

Of course, the name `myField` is arbitrary; you can use whatever text field name you like as long as the corresponding frame label is set to the same name. Note that setting the value of a text field with ActionScript does not trigger the field's *_changed* event. Only user keystrokes trigger *_changed*.

The *_changed* event is an undocumented feature. In future versions of Flash, a new, more standard method of event handling for text fields will likely be adopted.

HTML Support

The Character panel lets us set a text field's font size, font face, and font style, but it sets the attributes of the entire text field only. To set styles on a character-by-character basis and to add hypertext links, use HTML (which was added as a text field feature in Flash 5).

Though HTML can be used with both dynamic text fields and user-input text fields, we normally use HTML text fields for display purposes only. To add HTML support to a text field, select the HTML option in the Text Options panel.

The set of HTML tags supported by text fields is limited to: `<B>`, `<I>`, `<U>`, `<FONT>`, `<P>`, `<BR>`, and `<A>`.

<B> (Bold)

The `<B>` tag renders text in bold, provided that a boldface exists for the font in question:

```
<B>This is bold text</B>
```

,ch18.17514 Page 393 Monday, April 16, 2001 1:55 PM

<I> (Italics)

The <I> tag renders text in italics, provided that an italic face exists for the font in question:

```
<I>This is italic text</I>
```

<U> (Underline)

The <U> tag renders the tagged text with an underline beneath it. For example:

```
<U>This is underlined text</U>
```

Because linked text is not underlined in Flash, you should use the <U> tag to identify hyperlinks:

```
<A HREF="http://www.thesquarerootof-1.com"><U>Click here</U>
</A> to visit a neat site.
```

<FONT> (Font Control)

The <FONT> tag supports the following three attributes:

FACE

The FACE attribute specifies the name of the font to use. Note that a list of multiple font faces is not supported in Flash as it is in HTML. Flash attempts to render only the first font listed in the FACE attribute. For example, in the code <FONT FACE="Arial, Helvetica">my text</FONT>, Flash will not render "my text" in Helvetica if Arial is missing. Instead, text will be rendered in the default font.

SIZE

The SIZE attribute specifies the size of the tagged text as a fixed point size (such as <FONT SIZE="18">) or as a relative size. Relative point sizes are preceded by a + or – sign and are specified relative to the text size in the Character panel. For example, if the point size is 14 in the Character panel, then <FONT SIZE="-2"> displays the tagged text at 12 point.

COLOR

The COLOR attribute specifies the color of the tagged text, as a hexadecimal number, preceded by the pound sign (#). For example: <FONT COLOR="#FF0000">this is red text</FONT>. Specify the hexadecimal number as an RGB series of three two-digit numbers from 00 to FF. Note that Flash's implementation of the COLOR attribute is more strict than HTML's—the pound sign (#) is required, and color names such as "green" and "blue" cannot be used as COLOR values.

,ch18.17514 Page 394 Monday, April 16, 2001 1:55 PM

Here are some <FONT> examples:

```
<FONT FACE="Arial">this is Arial</FONT>
<FONT FACE="Arial" SIZE="12">this is 12pt Arial</FONT>
<FONT FACE="Lucida Console" SIZE="+4" COLOR="#FF0000">this is red,
+4pt Lucida Console</FONT>
```

See "Using HTML as Output" later in this chapter for more important details on fonts in Flash.

<P> (Paragraph Break)

The <P> tag demarcates paragraphs, but in Flash it behaves quite differently than its HTML counterpart. First of all, unterminated <P> tags do not cause line breaks in Flash as they do in regular HTML. Note the difference between Flash and web browser output:

```
I hate filling out forms. <P>So sometimes I don't.
// Flash output:
I hate filling out forms. So sometimes I don't.
// Web browser output:
I hate filling out forms.
So sometimes I don't.
```

Closing </P> tags are required by Flash in order for line breaks to be added. For example:

```
<P> I hate filling out forms.</P> So sometimes I don't.
```

Furthermore, in Flash, <P> causes a single line break, exactly like <BR>, whereas in web browsers, <P> traditionally causes a double line break. Consider the following:

```
<P>This is line one.</P><P>This is line two.</P>
```

In Flash, that code would be rendered with no gap between the lines, as in:

```
This is line one.
This is line two.
```

In a web browser, the code would be rendered with a gap between the lines, as in:

```
This is line one.

This is line two.
```

Because Flash's <P> tag behavior differs from web browsers, we often use the <BR> tag instead. However, the ALIGN attribute of the <P> tag is still useful to center, right-justify, or left-justify text, as follows:

```
<P ALIGN="CENTER">Centered text</P>
<P ALIGN="RIGHT">Right-justified text</P>
<P ALIGN="LEFT">Left-justified text</P>
```

,ch18.17514 Page 395 Monday, April 16, 2001 1:55 PM

*
 (Line Break)*

The <BR> tag causes a line break in a body of text and is functionally equivalent to the \n escape sequence or the newline keyword. Consider the following:

```
This is line one. <BR>This is line two.
This is line one. \nThis is line two.
```

Both would be rendered in Flash as:

```
This is line one.
This is line two.
```

<A> (Anchor or Hypertext Link)

The <A> tag creates a hypertext link. When the user clicks text tagged with <A>, the document specified in the HREF attribute of the tag loads into the browser. If the Player is running in standalone mode, the default web browser on the system is launched and the document is loaded into that browser.

The generic syntax of the <A> tag is:

```
<A HREF="documentToLoad.html">linked text</A>
```

For example, to link to a good video game, we could use:

```
<A HREF="http://www.quake3arena.com/">nice game</A>
```

As with HTML, the URL can be absolute or relative to the current page. Normally, links followed via an anchor tag cause the current movie to be replaced with the document specified in the HREF of the anchor tag. However, an anchor tag may also cause a secondary browser window to launch. Using the TARGET attribute, we can specify the name of a window into which to load the linked document, as follows:

```
<A HREF="documentName" TARGET="windowName">linked text</A>
```

If a window named *windowName* does not already exist, the browser launches a new window and assigns it the name *windowName*. To launch each document in its own, anonymous window, we can use the _blank keyword, as in:

```
<A HREF="mypage.html" TARGET="_blank">linked text</A>
```

Note that when we launch windows through the TARGET attribute, we have no control over the size or toolbar arrangement of the new window. To launch specifically sized windows from a link, we must use JavaScript. Techniques for launching customized secondary windows with JavaScript are described at:

http://www.moock.org/webdesign/flash

,ch18.17514 Page 396 Monday, April 16, 2001 1:55 PM

For more information on communicating with JavaScript from ActionScript, see the global functions *fscommand()* and *getURL()* in Part III, and "Executing JavaScript from HTML Links" later in this chapter.

The TARGET attribute can also be used to load documents into frames, as in:

```
<A HREF="documentName" TARGET="frameName">linked text</A>
```

Flash anchor tags do not always behave exactly like HTML anchor tags. We cannot, for example, use the NAME attribute of the anchor tag in Flash, so internal links within a body of text are not possible. Furthermore, Flash links are not underlined or highlighted in any way. Link underlines and colors must be inserted manually with the <U> and <FONT> tags described earlier.

Anchor Tag Tab Order

In Flash 5, anchor tags are not added to the tab order of the movie and are therefore not accessible via the keyboard. If your content must be accessible to keyboards and alternative input devices, you should use buttons, not anchor tags, for links.

Quoting Attribute Values

Outside Flash, HTML attribute values may be quoted with single quotes, double quotes, or not at all. The following tags are all valid in most web browsers:

```
<P ALIGN=RIGHT>
<P ALIGN='RIGHT'>
<P ALIGN="RIGHT">
```

But in Flash, unquoted attribute values are not allowed. For example, the syntax <P ALIGN=RIGHT> is illegal in Flash. However, both single and double quotes may be used to delimit attribute values. When composing text field values that include HTML attributes, we must be careful to quote our attributes correctly, using one type of quote to demarcate the string itself and another to demarcate attribute values. For example:

```
// These examples are both valid
myText = "<P ALIGN='RIGHT'>hi there</P>";
myText = '<P ALIGN="RIGHT">hi there</P>';
// This example would cause an error because double quotation marks are
// used to demarcate both the string and the attribute
myText = "<P ALIGN="RIGHT">hi there</P>";
```

For more information on using quotation marks to form strings, see "String Literals" in Chapter 4, *Primitive Datatypes*.

,ch18.17514 Page 397 Monday, April 16, 2001 1:55 PM

Unrecognized Tags and Attributes

Like web browsers, Flash ignore tags and attributes it does not recognize. For example, if we assign the following value to an HTML text field in Flash:

```
<P>Please fill in and print this form</P>
<FORM><INPUT TYPE="TEXT"></FORM>
<P>Thank you!</P>
```

The output would be:

```
Please fill in and print this form
Thank you!
```

The `FORM` and `INPUT` elements are not supported by Flash so both are ignored. Similarly, if we use container elements such as `<TD>`, the content is preserved but the markup is ignored. For example:

```
myTextField = "<TABLE><TR><TD>table cell text</TD></TR></TABLE>";
```

outputs the following line without table formatting:

```
table cell text
```

Using HTML as Output

HTML text entered manually into a text field using the Text tool will not be rendered as HTML. To display HTML-formatted text on screen, we must assign HTML text to a dynamic text field via ActionScript. For example:

```
myTextField = "<P><B>Error!</B> You <I>must</I> supply an email address!</P>";
```

Embedding a font for an HTML text field embeds only a single style of a single font. For example, a text field set to bold Arial in the Character panel will only support characters of the Arial bold typeface. If we use HTML to assign a different style of Arial (such as italic) or a different typeface altogether (such as Garamond), the tagged text will be invisible unless the appropriate fonts are embedded with the movie!

Suppose, for example, that we create a text field called `output`. In the Character panel for our `output` text field we select Arial set to Italic. In the Text Options panel, we embed the entire Arial italic font. Then we set `output` to display HTML. Finally, we assign the following value to our text field:

```
output = '<P><I>My</I>, what <B>lovely</B>'
         + '<FONT SIZE="24">eyes</FONT> you have!</P>';
```

When the movie plays, the following text will appear in the text field:

My

,ch18.17514 Page 398 Monday, April 16, 2001 1:55 PM

Everything else we assigned to `output` is missing! Only the italic text in the HTML can be rendered. The rest of the text requires other variations of the Arial font that we didn't embed—"what", "eyes", and "you have" are all nonitalic, and "lovely" is bold.

For every font face and variation we use in an HTML text field, we must embed the appropriate font. We have two means of doing so:

- Make a dummy text field, hidden from view, with the desired font selected in the Character panel and embedded in the Text Options panel.
- Add a new font symbol to the movie's Library and export the font with the movie.

Here are the steps for embedding Arial bold in a movie for use with a text field:

1. Select Window → Library.
2. Select Options → New Font. The Font Symbol Properties dialog box appears.
3. Under Font, select Arial.
4. Under Style, select Bold.
5. Under Name, type **`ArialBold`** (this is a cosmetic name, used only in the Library).
6. In the Library, select the `ArialBold` font symbol.
7. Select Options → Linkage.
8. In the Symbol Linkage Properties dialog box, select Export This Symbol.
9. In the Identifier box, type **`ArialBold`**. For our purposes, the name we type here doesn't matter. Exported symbol identifiers are used only for shared libraries.

Note that every variation of a font style must be embedded individually. If we use Arial bold, Arial italic, and Arial bold italic in a text field, then we must embed all three font variations. Underline is not considered a font variation, nor is font size or color.

If, however, we do not enable *any* of the Embed Fonts options in the Text Options panel, then Flash relies entirely on the user's system for fonts, in which case normal, bold, and italic text will be rendered only if users have the appropriate font variant installed on their systems.

To ensure that text will display consistently across all platforms and user systems, you should embed all the fonts required for your text field.

,ch18.17514 Page 399 Monday, April 16, 2001 1:55 PM

Using HTML as Input

Whereas HTML is normally used with text fields for display purposes, it may also be entered into a movie via an HTML-enabled or a regular (non-HTML) user-input text field.

When regular text is entered into an HTML-enabled user-input text field, HTML markup tags are added automatically. For example, the text "Hi there" would be converted to the HTML value:

```
'<P ALIGN="LEFT"><FONT FACE="Arial" SIZE="10" COLOR="#000000">Hi there</FONT></P>'
```

When HTML tags are typed into an HTML-enabled user-input text field, the `<` and `>` characters are converted to `>` and `<`. For example, the text "`<B>hi there</B>`" would be converted to the value:

```
'<P ALIGN="LEFT"><FONT FACE="Arial" SIZE="10"
 COLOR="#000000">&lt;B&gt;hi there&lt;/B&gt;</FONT></P>'
```

HTML-enabled user-input text fields may be used to create a very simple HTML data entry system.

When regular or HTML text is typed into a normal (non-HTML) user-input text field, no modification of the entered text occurs. Regular user-input text fields allow raw HTML code to be entered into a movie without distortion.

An example showing HTML-enabled and regular user-input text field data entry is available from the online Code Depot.

Executing JavaScript from HTML Links

In most JavaScript-capable web browsers, it is possible to execute JavaScript statements from an anchor tag using the `javascript:` protocol as the value of the `HREF` attribute. For example:

```
<A HREF="javascript:square(5);">find the square of 5</A>
```

In ActionScript, we can also execute JavaScript statements from an `<A>` tag, like this:

```
myTextField = "<A HREF='javascript:alert(5);'>display the number 5</A>";
```

However, to include string values in JavaScript statements, we must use the HTML entity `&quot;` for quotation marks, as in:

```
myTextField = "<A HREF='javascript:alert(&quot;hello world&quot;);'>"
              + "display hello world</A>";
```

,ch18.17514 Page 400 Monday, April 16, 2001 1:55 PM

Calling ActionScript Functions from HTML Links

Though arbitrary statements of ActionScript code cannot be executed from a Flash `<A>` tag, ActionScript *functions* can. To invoke an ActionScript function from an anchor tag, we use the following syntax:

```
<A HREF="asfunction:myFunctionName">invoke the function</A>
```

The function invocation operator ( ) is not allowed and should not be used when invoking an ActionScript function from an anchor tag. In addition to calling an ActionScript function from an anchor tag, we may also pass one parameter to that function using the syntax:

```
<A HREF="asfunction:myFunctionName,myParameter">invoke the function</A>
```

where *myParameter* is the value of the parameter to pass. Inside the invoked function, *myParameter* is always a string. To pass more than one piece of information to a function from an anchor, we use a delimiter in the *myParameter* value and dissect the string ourselves in the function. For example, here's a function invocation that passes two values, separated by a | character, to the *roleCall()* function:

```
<A HREF="asfunction:roleCall,megan|murray">invoke the function</A>
```

And here's the *roleCall()* function. Notice how it separates the values with the *split()* method:

```
function roleCall (name) {
  var bothNames = name.split("|");
  trace("first name: " + bothNames[0]);
  trace("last name: " + bothNames[1]);
}
```

Working with Text Field Selections

When a user selects a portion of a dynamic or user-input text field, the positions of the selected characters are stored in a special built-in object called the *Selection* object. Using the *Selection* object, we can check which part of a text field a user has selected or even select a part of a text field programmatically. The *Selection* object can also tell us which of a series of text fields is currently selected by the user. Finally, we can use the *Selection* object to give keyboard focus to a particular text field, prompting a user to type in a suggested location.

To learn how to work with text field selections, see the *Selection* object in Part III.

,ch18.17514 Page 401 Monday, April 16, 2001 1:55 PM

Empty Text Fields and the for-in Statement

To check all the values of the variables on a timeline, we can use the *for-in* statement as described in Chapter 6, *Statements*. Undefined text fields (those that appear on screen but contain the `undefined` value), however, are not enumerated by the *for-in* statement. (Fields containing the empty string (`""`) or only spaces are not considered empty and are therefore enumerated.)

The invisibility of undefined text fields in *for-in* loops can cause problems for error-checking scripts. Scripts that use a *for-in* loop to cycle through a series of text fields must be written to account for undefined text fields. For example, here we attempt to check a movie clip called `formClip` to see if any of its variables contain the empty string:

```
for (i in formClip) {
  if (formClip[i] == "") {
    trace(i + " is empty! don't submit the form!");
    break;
  }
}
```

As is, that code would not function as desired because undefined text fields would not be enumerated by the loop and would never be checked. To force an undefined text field to be enumerated in a *for-in* loop, we must deliberately assign the empty string to a corresponding timeline variable. For example, we would attach this script to a frame of `formClip` in order to fix our previous example:

```
// Assign our text fields the empty string so that
// they show up in our for-in loop
formField1 = "";
formField2 = "";
```

Onward!

We're almost at that inevitable stage where the guided tour ends and you head off to explore your own projects and ideas. The previous two chapters have taught us how to create forms, display information on screen, and retrieve user input. Our last stop before the reference section—debugging code—teaches survival techniques to use in your uncharted journeys ahead.