macromedia®

FLASH® 8

Learning Flash Lite 1.X ActionScript

Acknowledgments

Project Management: Mary Leigh Burke

Writing: Tim Statler

Managing Editor: Rosana Francescato

Editing: Linda Adler, Geta Carlson, Evelyn Eldridge, Mary Kraemer, Lisa Stanziano

Production Management: Patrice O’Neill, Kristin Conradi, Yuko Yagi

Media Design and Production: Adam Barnett, Aaron Begley, Paul Benkman. John Francis, Geeta Karmarkar, Masayo Noda, Paul Rangel, Arena Reed, Mario Reynoso

Special thanks to Lisa Friendly, Bonnie Loo, Erick Vera, the beta testers, and the entire Flash Lite engineering and QA teams.

First Edition: September 2005

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

Contents

CHAPTER 1

About Flash Lite 1.x ActionScript

1

You use ActionScript to add programming logic and interactivity to your Macromedia Flash Lite applications. The version of ActionScript in Flash Lite 1.0 and 1.1—referred to collectively as Flash Lite 1.x ActionScript—is a hybrid of Flash 4 ActionScript, plus additional commands and properties specific the Flash Lite player, such as the ability to initiate phone calls or text messages, or get time and date information from the device.

This chapter contains the following topics:

Flash Lite 1.x ActionScript overview

Flash Lite 1.x ActionScript consists of the following parts:

Flash Player 4 ActionScript This includes operators (for example, comparison and assignment operators), movie clip properties (for example, `_height`, `_x`, and `_y`), Timeline control functions (for example, `gotoAndPlay()` or `stop()`), and network functions, such as the `loadVariables()` and `loadMovie()` functions (Flash Lite 1.1 only). For a list of unsupported Flash 4 ActionScript, see "Flash 4 ActionScript not supported by Flash Lite 1.x ActionScript" on page 6.

Phone integration commands and properties Flash Lite provides commands that let you, for example, query the date and time information from the device, initiate a phone call or short message service (SMS) text message, or start external applications installed on the device.

Platform capability variables (Flash Lite 1.1 only) These properties provide information about the capabilities of the device or Flash Lite runtime environment. For example, the `_capLoadData` variable indicates if your application can load data over the network.

fscommand2() function Like the `fscommand()` function, you use `fscommand2()` to communicate with the host environment or system—in this case, the mobile phone or device. The `fscommand2()` function provides enhancements to `fscommand()`, including the ability to pass an arbitrary number of arguments and to retrieve immediate return values (rather than having to wait until the next frame, as with `fscommand()`).

Differences between Flash Lite 1.0 and Flash Lite 1.1 ActionScript

The following Flash Lite 1.1 ActionScript features are not available in Flash Lite 1.0:

- Network access or network status information. For example, in Flash Lite 1.0 you cannot use the `loadVariables()` or `loadMovie()` functions to load external data or SWF files, or the various `fscommand2()` commands for determining a device's connection signal strength or the status of a network request.
- Getting time and date information from the device.
- Platform capability variables, which provide information about the capabilities of the Flash Lite platform and of the device.
- The `fscommand2()` function and its associated commands, such as `SetSoftKeys` and `FullScreen`.
- The `scroll` and `maxscroll` text field properties.

Flash 4 ActionScript not supported by Flash Lite 1.x ActionScript

The following Flash 4 ActionScript features are unsupported, or only partially supported, in Flash Lite 1.x ActionScript:

- The `startDrag()` and `stopDrag()` functions.
- Flash Lite 1.x ActionScript supports a subset of the button events supported in Flash Player 4. For more information about handling button events, see Chapter 1, "Creating Interactivity and Navigation" in *Developing Flash Lite Applications*.
- Flash Lite 1.x ActionScript supports a subset of key events supported in Flash Player 4. For more information about supported key events in Flash Lite, see Chapter 1, "Creating Interactivity and Navigation" in *Developing Flash Lite Applications*.
- The `_dropTarget` property.

- The `_soundBufTime` property.
- The `_url` property.
- The `String()` conversion function.

Features not available in Flash Lite 1.x ActionScript

Because Flash Lite player is based on an older version of Flash Player, it does not support all the programming features available in more recent releases of Flash Player or other programming languages that you might be familiar with. This section discusses programming features not available in Flash Lite 1.x ActionScript and available alternatives and work-arounds.

User-defined functions Flash Lite 1.x does not support the ability to define and call custom functions. However, you can use the `call()` function to execute code that resides on an arbitrary frame in the timeline. For more information, see "Using the call() function to create functions" on page 13.

Native arrays, objects, or other complex data types Flash Lite 1.x does not support native array data structures or other complete data types. However, you can emulate arrays using pseudo-arrays, a technique that involves using the `eval()` function to dynamically evaluate concatenated strings. For more information, see "Emulating arrays" on page 11.

Runtime loading of external image or sound files Unlike the desktop version of Flash Player, Flash Lite 1.x ActionScript cannot load external JPEG files or MP3 files. In Flash Lite 1.1 you can use the `loadMovie()` function to load external SWF files. For more information, see "Loading external SWF files" on page 22.

CHAPTER 2

Flash 4 ActionScript Primer

2

Flash Lite 1.x ActionScript is based on the version of ActionScript that was first available in Flash Player 4. Consequently, several programming features available in later versions of Flash Player (for desktop systems) are not available to Flash Lite 1.x applications.

If you're unfamiliar with Flash 4 ActionScript syntax and features or if you've forgotten some of the details from previous Flash development work, this chapter provides a primer on using Flash 4 ActionScript in your Flash Lite applications.

This chapter contains the following topics:

Getting and setting movie clip properties

To get or set a movie clip property (if settable), you can use dot syntax or the `setProperty()` or `getProperty()` functions. You can also use the `tellTarget()` function.

To use dot syntax, specify the movie clip instance name, followed by a dot (.) and then the property name. For example, the following code gets the *x* screen coordinate (represented by the `_x` movie clip property) of the movie clip named `cartoonArea`, and assigns the result to a variable named `x_pos`.

```
x_pos = cartoonArea._x;
```

The following example is equivalent to the previous example, but uses the `getProperty()` function to retrieve the movie clip's *x* position:

```
x_pos = getProperty(cartoonArea, _x);
```

The `setProperty()` function lets you set a property of a movie clip instance, as shown in the following example:

```
setProperty(cartoonArea, _x, 100);
```

The following example is equivalent to the previous example but uses dot syntax:

```
cartoonArea._x = 100;
```

You can also get or set movie clip properties from within a `tellTarget()` statement. The following code is equivalent to the `setProperty()` example shown previously:

```
tellTarget("/cartoonArea") {
  _x = 100;
}
```

For more information about the `tellTarget()` function, see "Controlling other timelines" on page 10.

Controlling other timelines

To specify a path to a timeline, use slash syntax (/) combined with dots (..) to build the path reference. You can also use `_levelN`, `_root`, or `_parent` from Flash 5 notation to refer to, respectively, a specific movie level, the application's root timeline, or the parent timeline.

For example, suppose you had a movie clip instance named `box` on your SWF file's main timeline. The `box` instance, in turn, contains another movie clip instance named `cards`. The following examples target the movie clip `cards` from the main timeline:

```
tellTarget("/box/cards")
tellTarget("_level0/box/cards")
```

The following example targets the main timeline from the movie clip `cards`:

```
tellTarget("../../cards")
tellTarget("_root")
```

The following example targets the parent movie clip `cards`:

```
tellTarget("../cards")
tellTarget("_parent/cards")
```

Using variables

To specify a variable on a timeline, use slash syntax (/) combined with dots (..) and colons (:). You can also use the dot notation.

The following code refers to the `car` variable on the main timeline:

```
/:car
_root.car
```

The following example shows the `car` variable in a movie clip instance that resides on the main timeline:

```
/mc1/mc2/:car
_root.mc1.mc2.car
```

The following example shows the `car` variable in a movie clip instance that resides on the current timeline:

```
mc2/:car
mc2.car
```

Emulating arrays

Arrays are useful for creating and manipulating ordered lists of information such as variables and values. However, Flash Lite 1.1 does not support native array data structures. A common technique in Flash Lite (and Flash 4) programming is to emulate arrays with string processing. An emulated array is also called a pseudo-array. The key to pseudo-array processing is the `eval()` ActionScript function, which lets you access variables, properties, or movie clips by name. For more information, see "Using the eval() function" on page 17.

A pseudo-array typically consists of two or more variables that share the same base name, followed by a numeric suffix. The suffix is the index for each array element.

For example, suppose you create the following ActionScript variables:

```
color_1 = "orange";
color_2 = "green";
color_3 = "blue";
color_4 = "red";
```

You can then use the following code to loop over the elements in the pseudo-array:

```
for (i = 1; i <=4; i++) {
  trace (eval ("color_" add i));
}
```

In addition to letting you reference existing variables, you can also use the `eval()` function on the left side of a variable assignment to create variables at runtime. For example, suppose you want to maintain a list of high scores as a user plays a game. Each time the user completes a turn, you add their score to the list:

```
eval("highScore" add scoreIndex) = currentScore;
scoreIndex++;
```

Each time this code executes, it adds a new item to the list of high scores and then increments the `scoreIndex` variable, which determines each item's index in the list. For instance, you might end up with the following variables:

```
highScore1 = 2000
highScore2 = 1500
highScore3 = 3000
```

Working with text and strings

Flash Lite provides some basic ActionScript commands and properties for working with text. You can get and set the values of text fields, concatenate strings, URL-encode or URL-decode text strings, and create scrolling text fields.

This section contains the following topics:

- "Concatenating strings" on page 12
- "Scrolling text" on page 12

Concatenating strings

To concatenate strings in Flash Lite, you use the `add` operator, as the following example shows:

```
city = "Boston";
team = "Red Sox";
fullName = city add " " add team;
// Result:
// fullName = "Boston Red Sox"
```

Scrolling text

You can use the `scroll` property of dynamic and input text fields to get or set the field's current scroll position. You can also use the `maxscroll` position to determine a text field's current scroll position relative to the maximum scroll position. For an example of how to create a scrolling text field, see "Creating scrolling text (Flash Professional Only)" in *Developing Flash Lite Applications*.

Using the call() function to create functions

You can't define or call custom functions in Flash Lite as you can in Flash Player 5 and later. However, you can use the `call()` ActionScript function to execute code that resides on an arbitrary frame in the timeline. This technique lets you encapsulate commonly used code in a single location, making it easier to maintain.

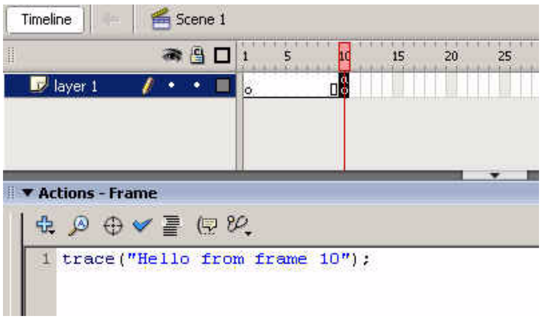
The `call()` function takes a frame number or frame label as a parameter. For example, the following ActionScript calls the code located on the frame labeled `moveUp`:

```
call("moveUp");
```

The `call()` function operates synchronously; any ActionScript that follows a `call()` function call won't execute until all of the ActionScript on the specified frame finishes executing.

To call ActionScript on another frame:

1. In a new Flash document, insert a keyframe on Frame 10.



```
trace("Hello from frame 10");
```

2. With the new keyframe selected, open the Actions panel (Window > Actions), and type the following code:

   ```
   trace("Hello from frame 10");
   ```

3. Select the keyframe on Frame 1, and in the Actions panel, type the following code:

   ```
   stop();
   call(10);
   ```

   This code stops the playhead on Frame 1, and then calls the code on Frame 10.

4. Test the application in the emulator and open the Output panel (Window > Output).

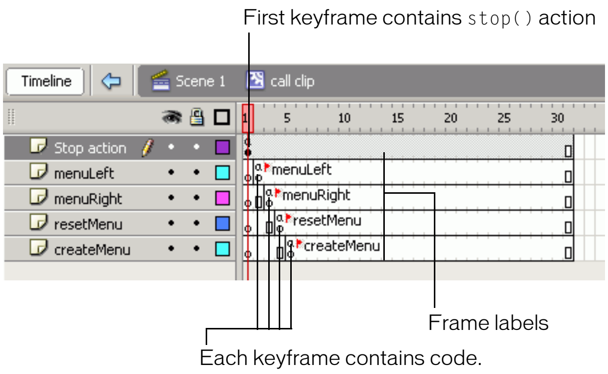
   You should see "Hello from frame 10" displayed in the Output panel.

You can also call code that resides on another timeline, such as a movie clip's timeline. To execute the code, specify the movie clip instance name followed by a colon, and then the frame number or label. For example, the following ActionScript calls the code that resides on the frame labeled `moveUp` in the movie clip instance named `callClip`:

```
call("callClip:moveUp");
```

This technique is often used to create *call clips* or *function clips*—movie clips whose sole purpose is to encapsulate regularly used code. A call clip contains a keyframe for each function you want to create. You typically label each keyframe according to its purpose. Macromedia also recommends that you create a new layer for each new keyframe, and that you give each layer the same name as the frame label you assign to the keyframe.

The following figure shows the Timeline of an example call clip. The first keyframe of a call clip always contains a `stop()` action, which ensures that the playhead doesn't continually loop over the frames in its Timeline. Subsequent keyframes contain code for each "function." Each function keyframe is labeled to identify what it does. To make editing and viewing the call clip easier, each function keyframe is typically inserted on a separate layer.



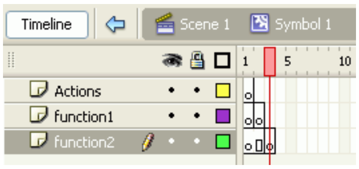
The following procedure explains how to create and use a call clip.

To create and use a call clip:

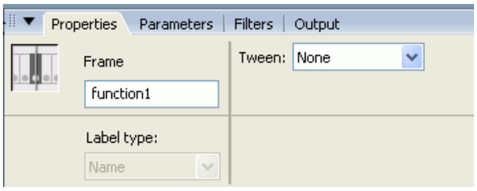
1. In Flash Professional 8, create a new document from the Flash Lite 1.1 Symbian Series 60 document template.
2. Select Insert > New Symbol.
3. In the Create New Symbol dialog box, type **Call Clip** in the Name text box, and then click OK.

   The movie clip opens in editing mode.

4. Click the Add New Layer button the Timeline window twice to insert two new layers. Name the top layer **Actions**, the second layer **function1**, and the third layer **function2**.
5. Insert a keyframe on Frame 2 of the function1 layer, and another keyframe on Frame 3 of the function2 layer, as the following figure shows:



6. Select the keyframe on the Actions layer and open the Actions panel.
7. Add a `stop()` action to the Actions panel.
8. Select the keyframe on Frame 2 of the function1 layer and do the following:
   a. In the Property inspector, type **function1** in the Frame Label text box.

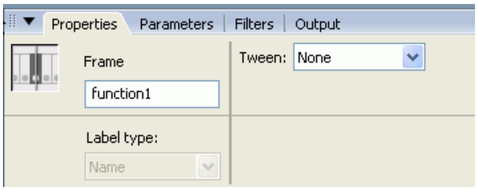


Frame: function1
Tween: None
Label type: Name

   b. In the Actions panel (Window > Actions), type the following code:

```
trace("function1 was called.");
```

9. Select the keyframe on Frame 3 of the function2 layer and do the following:
   a. In the Property inspector, type **function2** in the Frame Label text box.



Frame: function1
Tween: None
Label type: Name

   b. In the Actions panel (Window > Actions), type the following code:

```
trace("function2 was called.");
```

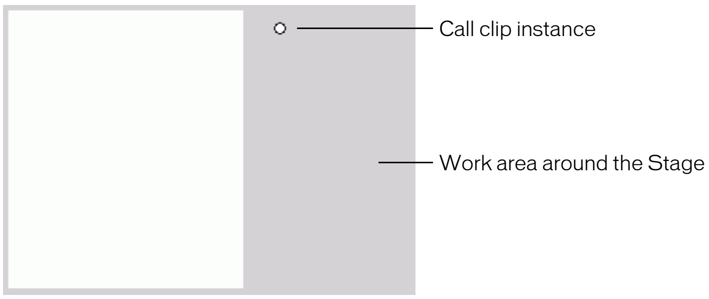
10. Press Control+E (Windows) or Command+E (Macintosh) to return to the main Timeline.

11. Set your document's view to include the work area around the Stage by selecting View > Work Area.

    Because the call clip doesn't need to be visible to the user, you can place it in the work area.

12. Open the Library panel (Window > Library) and drag the Call Clip symbol to the work area around the Stage.

    The call clip doesn't contain any visual elements so it appears on the Stage as a small circle, representing the movie clip's registration point.



TIP

To make your call clip more easily identifiable on the Stage, add some text or other visual element to the first keyframe in the call clip's Timeline.

13. In the Property inspector, type **callClip** in the Instance Name text box.

14. In the Timeline, select Frame 1 on the layer named ActionScript.

15. In the Actions panel, enter the following code:

```
call("callClip:function1");
call("callClip:function2");
```

16. Test your application in the emulator (Control > Test Movie).

    You should see the following text in the Output panel:

```
function1 was called.
function2 was called.
```

Using the eval() function

The `eval()` function lets you dynamically reference variables and movie clip instances at runtime. The `eval()` function takes a string expression as a parameter and returns either the value of the variable represented by that expression or a reference to a movie clip.

For example, the following code evaluates the value of the `name` ActionScript variable and assigns the result to `nameValue`:

```
name = "Jack";
nameValue = eval("name");
// result: nameValue = "Jack"
```

The `eval()` function is often used with `for()` loops and the `add` (string concatenation) operator to create string-based arrays, because Flash Lite doesn't support native array data structures. For more information, see "Emulating arrays" on page 11.

You can also use `eval()` to reference movie clip instances by name. For example, suppose you had three movie clips named `clip1`, `clip2`, and `clip3`. The following `for()` loop increments the *x* position of each clip by 10 pixels:

```
for(index = 1; index <= 3; index++) {
  eval("clip" add index)._x += 10
}
```

CHAPTER 3

Common Scripting Tasks

3

This chapter discusses common Flash Lite scripting tasks for working with the user's device. These include, for example, getting device capability information, initiating phone calls and text messages, and determining network status.

This chapter contains the following topics:

Determining device and platform capabilities

Flash Lite 1.1 includes a number of ActionScript variables that provide information about features and capabilities available to Flash Lite applications running on a particular device. For example, the `_capLoadData` variable indicates if the device supports loading external data, and the `_capSMS` variable indicates if the device supports sending SMS (short message service) messages. For a full list of capability variables, see "Capabilities" in the *Flash Lite 1.x ActionScript Language Reference*.

Typically, you use capability variables to determine if a device supports a specific feature before attempting to use that feature. For example, suppose that you wanted to develop an application that downloads data from a web server using the `loadVariables()` function. Before attempting to load the data, you can first check the value of the `_capLoadData` variable to determine if the device supports that feature, as follows:

```
if(_capLoadData == 1) {
  loadVariables("http://www.macromedia.com/data.txt");
} else {
  status_message = "Sorry, unable to load external data."
}
```

Flash Lite defines capability variables on the root timeline of the main SWF file. So to access these variables from another timeline—for example, from within a movie clip's timeline—you need to qualify the path to the variable. For instance, the following example uses a slash (/) to provide the fully qualified path to the `_capSMS` variable.

```
canSendSMS = /:_capSMS
```

Opening a web page

You use the `getURL()` command to open a web page in the device's web browser. This is the same way you open a web page from a desktop Flash application. For example, the following opens the Macromedia web page:

```
getURL("http:www.macromedia.com");
```

Flash Lite processes only one `getURL()` action per frame or per event handler. Certain handsets restrict the `getURL()` action to keypress events only, in which case the `getURL()` call is processed only if it is triggered within a keypress event handler. Even under such circumstances, only one `getURL()` action is processed per keypress event handler. The following code, attached to a button instance on the Stage, opens a web page when the user presses the Select button on the device:

```
on (keyPress "<Enter>"){
  getURL("http://www.macromedia.com");
}
```

Initiating a phone call

To initiate a phone call from a Flash Lite application, you use the `getURL()` function. Typically, you use this function to open a web page, but in this case you specify `tel:` as the protocol (in place of `http`), and then provide the phone number you wish the phone to dial. When you call this function, Flash Lite displays a confirmation dialog box asking the user for permission to make the call to the specified number.

The following code attempts to initiate call to 555-1212:

```
getURL("tel:555-1212");
```

Flash Lite only processes one `getURL()` action per frame or per event handler. Certain handsets restrict the `getURL()` action to keypress events only, in which case the `getURL()` call is processed only if it is triggered within a keypress event handler. Even under such circumstances, only one `getURL()` action is processed per keypress event handler. The following example starts a phone call when the user presses the Select button on the device:

```
on (keyPress "<Enter>"){
  getURL("tel:555-1212");
}
```

Initiating a text or multimedia message

You can use Flash Lite to initiate a short message service (SMS) or multimedia message service (MMS) message. To initiate an SMS or MMS message in a Flash Lite application, you use the `getURL()` command, passing it the `sms:` or `mms:` protocols in place of the standard `http` protocol, and then the phone number to which you want to send the message.

```
getURL("sms:555-1212");
```

You can optionally specify the message body in the URL query string, as the following code shows:

```
getURL("sms:555-1212?body=More info please");
```

To initiate an MMS message, you use the `mms:` protocol instead of `sms:`, as follows:

```
getURL("mms:555-1212");
```

NOTE

It's not possible to specify an attachment for the MMS message from Flash Lite.

Flash Lite processes only one `getURL()` action per frame or per event handler. Certain handsets restrict the `getURL()` action to keypress events only, in which case the `getURL()` call is processed only if it is triggered within a keypress event handler. Even under such circumstances, only one `getURL()` action is processed per keypress event handler. The following example initiates an SMS message when the user presses the Select button on the device:

```
on (keyPress "<Enter>"){
  getURL("sms:555-1212");
}
```

Initiating an e-mail message

You can use Flash Lite to initiate an e-mail message. To initiate an e-mail message, you use the `getURL()` command and pass it the `mailto:` protocol, followed by the recipient's e-mail address. You can optionally specify the message subject and body in the URL's query string, as follows:

```
getURL("mailto:mobile-developer@macromedia.com?subject=Flash Lite");
```

To specify just the message body in the query string, use the following code:

```
getURL("mailto:mobile-developer@macromedia.com?body=More+info+please");
```

Loading external SWF files

The `loadMovie()` function lets you load SWF files from a network or local file. This feature is only available in Flash Lite 1.1 and later. The following caveats apply when you load external SWF files:

- Flash Lite can load other Flash Lite 1.0 or Flash Lite 1.1 SWF files, or Flash 4-formatted SWF files or earlier. If you attempt to load a SWF file in another format (for example, a Flash Player 6 SWF file), Flash Lite will generate a runtime error.
- Flash Lite cannot directly load external image files, such as JPEG or GIF images. To load these types of media, you need to convert the image data to the SWF file format. You can do this "manually" with the Flash authoring tool by importing the image file into a new document, and then exporting the file to a Flash Lite or Flash 4 SWF file. There are also third-party utilities that can perform this type of conversion for you automatically.

For more information about loading SWF files, see `loadMovie()` in *Flash Lite 1.x ActionScript Language Reference*.

Loading external data

To load external data into a Flash Lite application, you use the `loadVariables()` function. You can load data over the network (from an HTTP address) or from the local file system. This feature is available only in Flash Lite 1.1 and later.

This section demonstrates how to use the `loadVariables()` function to load data from an external file and display that data in dynamic text fields. First you'll create the data file, a text file that contains five name-value pairs separated by ampersand (&) symbols. Then you'll create the Flash Lite application that loads and displays the data contained in the text file.

This example assumes that the data file and the SWF are both located in the same folder, either on your computer (when you test in the emulator) or on the device's memory card (when you test on an actual device). To test the application on the device, you must do one of the following:

- Transfer the text file to your device and put it in the same folder as the SWF file.
- Post the text file to a URL on a web server (for example, www.your-server.com/data.txt). Then modify the `loadVariables()` call in the sample application to point to that URL, as follows:

  ```
  loadVariables("http://www.your-server.com/data.txt", "data_clip");
  ```

  For an example of an application that loads data over the network, see the "Flash Lite news reader" in *Flash Samples*.

To create the data file:

1. Using a text editor (for example, Notepad or SimpleText), create a file that contains the following text:

   ```
   item_1=Hello&item_2=Bonjour&item_3=Hola&item_4=Buon+giorno&item_5=G'day
   ```

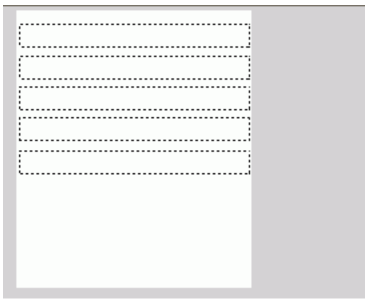
2. Save the file as data.txt.

To create the Flash Lite application to load the data:

1. Create a new document from the Flash Lite 1.1 Symbian Series 60 document template.

   For more information about using Flash Lite document templates, see "Using Flash Lite document templates (Flash Professional Only)" in *Getting Started with Flash Lite*.

2. Save the file as dataloading.fla to the same folder that contains the text file (data.txt) that you created previously.
3. In the Timeline, select Frame 1 of the layer named Content.
4. Using the Text tool, create five dynamic text fields on the Stage, as the following figure shows:



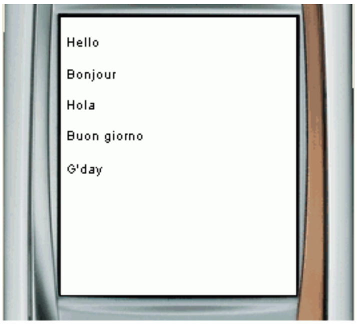
5. Select the first (top-most) text field, and in the Property inspector, type `item_1` in the Var text box.

   This variable name corresponds to the name of the first variable defined in the data.txt file you created previously (`item_1=Hello`).
6. In the same manner as described in the previous two steps, give the remaining four text fields the variable names `item_2`, `item_3`, `item_4`, and `item_5`.
7. Shift-select each text field so that they're all selected, and select Modify > Convert To Symbol.
8. In the Convert to Symbol dialog box, select Movie Clip as the symbol type and click OK.
9. Select the movie clip you just created and, in the Property inspector, type **data_clip** in the Instance Name text box.
10. In the Timeline, select Frame 1 of the Actions layer and open the Actions panel (Window > Actions).
11. Type the following code in the Actions panel:

```
loadVariables("data.txt", "data_clip");
```

12. Save your changes (File > Save) and test the application in the emulator (Control > Test Movie).

    You should see each text field populated with the data in the text file, as the following figure shows:


Hello
Bonjour
Hola
Buon giorno
G'day

Index