MX

macromedia®

# FLASH™ MX Professional 2004

## Macromedia Flash Lite 1.1 Authoring Guidelines

First Edition: June 2004

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

# CONTENTS

# CHAPTER 1
# Introduction

*Macromedia Flash Lite Authoring Guidelines for* covers tips, techniques, and sample code for developing Macromedia Flash content for mobile phones using Macromedia Flash Lite 1.1.

Running Macromedia Flash Lite 1.1 on mobile phones allows users to view and interact with a wide range of Flash content, such as games, informational guides, and dynamically updated applications.

In addition to the information described in this guide, the developer kit includes numerous examples and sample code to help clarify some of the ideas and concepts presented.

## Using Macromedia Flash Lite 1.1

Macromedia Flash Player is broadly distributed on a variety of platforms, from Windows, Macintosh, and UNIX-based desktop computers, to mobile phones, PDAs, and set-top boxes.

The Macromedia Flash Player application is approximately 500 KB, depending on the CPU. This and its runtime memory requirements make it too large for most mobile phones. Therefore, Macromedia created a new version of Flash Player called Macromedia Flash Lite, designed for consumer devices, including mobile phones. For a new generation of mobile phones, an updated version has been created, Flash Lite 1.1.

Macromedia Flash Lite 1.1 for the mobile phones lets Flash designers, developers, and content providers quickly create engaging content for mobile phones using the ActionScript scripting language, drawing tools, and templates.

## Getting started

To create content for mobile phones, you must have the following items on your computer:

- The latest version of Macromedia Flash MX Professional 2004 (7.0.1)
- The new FlashLite1_1.dll (FlashLite1_1.dmg on the Mac) file for testing Flash applications in the Flash Lite 1.1 authoring environment
- The new FlashLite1_1.xml file for publishing Flash Lite 1.1 SWF files
- The DevicesMsg.cfg configuration file for customizing the features that are supported in Flash Lite 1.1.

## Installing the Flash MX Professional 2004 7.0.1 update

To export Flash Lite 1.1 contents for mobile phones correctly, you need to have the latest version of Macromedia Flash MX Professional 2004 (7.0.1). You can download the updater program from the Macromedia website: www.macromedia.com/support/flash/downloads.html.

## Installing the FlashLite1_1.dll (FlashLite1_1.dmg on the Mac) file

The FlashLite1_1.dll (FlashLite1_1 on the Mac) file is part of the Flash Lite 1.1 Authoring Updater. This DLL is to be used to test content when you select Test Movie to validate your content. This new DLL is used when Flash Lite 1.1 is selected as the Flash version to publish to (using the publish setting interface). Copy the appropriate file to the following location:

- Windows:

  C:\Program Files\Macromedia\Flash MX 2004*language*\Configuration\Players
- Mac OS X:

  Macintosh HD::Applications:Macromedia Flash MX 2004:Configuration:Players

## Installing the FlashLite1_1.xml file

To author content using the FSCommand and the new FSCommand2 ActionScript, copy the FlashLite1_1.xml file, which is available in the Installs folder of the CDK, into the following location:

- Windows:

  C:\Program Files\Macromedia\Flash MX 2004*language*\Configuration\Players
- Mac OS X:

  Macintosh HD::Applications:Macromedia Flash MX 2004:Configuration:Players

## Installing the configuration file

The Flash Lite 1.1 Test Movie command allows users to customize the features that are supported in Flash Player. From the Flash Lite 1.1 Authoring Updater, copy the DeviceMsg.cfg configuration file into the following location:

- Windows 2000/ WindowsXP:

  C:\Documents and Settings*user name*\Local Settings\Application Data\Macromedia\Flash MX 2004\language\Configuration\
- Windows 98(SE):

  C:\Windows\Profiles*user name*\Application Data\Macromedia\Flash MX 2004\language\Configuration\
- Macintosh:

  Macintosh HD::Users:*user name*:Library:Application Support:Macromedia:Flash MX 2004:language:Configuration:

## Supported Devices

For details about mobile phones that support Flash Lite functionality, see the Macromedia Developer Center web site at www.macromedia.com/devnet/devices/.

# CHAPTER 2
# Optimizing Content

This chapter describes considerations for creating Macromedia Flash Lite content that runs on mobile phones, from general functionality to performance and size constraints.

## Navigation and key events

Macromedia Flash Lite 1.1 for mobile uses three keys for navigation: Up, Down, and Select. These three keys correspond to the Shift+Tab, Tab, and Enter keys on the Windows versions of Macromedia Flash Player.

The keys 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *, and # are also available. These correspond to the same keys on the desktop versions of Flash Player. You can attach ActionScript to these keys and to the Enter key as you would normally in Flash. ActionScript attached to other keys is ignored.

## Fonts and text

Flash Lite 1.1 includes support for both device fonts and embedded fonts. Although embedded fonts give you more control over the design of your content, they increase the size of the SWF file. Supported mobile phones support multiple device fonts providing content developers with multiple options for using device text fonts helping keep your file size small.

When using device fonts, Flash Lite 1.1 limits text-formatting options in dynamic text fields to justification (left, center, right) and color. Formatting options such as superscript, subscript, and kerning are not supported.

When you create Flash Lite content, you can use Flash to embed text. If you place text inside the application or graphics, use a typeface that is designed specifically for small screens. Choosing readable fonts is always an important design consideration. This section describes several options for using fonts and text for Flash Lite content.

### Device fonts

When you create static text, you can specify that Flash Player use device fonts to display certain text blocks. Using device fonts can decrease the file size of your SWF file, because the SWF file does not contain font outlines for the text.

Supported mobile phones support multiple system fonts, which can be accessed in a SWF file by setting the associated font style and selecting the Device Fonts check box. Some mobile phones support multiple fonts. For more details, see the Macromedia Developer Center web site at www.macromedia.com/devnet/devices/.

### Alias text support

Because of the limited screen size of mobile phones, it's important to use font sizes that are legible. With Flash MX Professional 2004, Macromedia has added a new option for rendering text, the Alias Text button.

### Alias Text button

The Alias Text button in the Property inspector lets you render text so that it appears more readable at small sizes.

**To enable the Alias Text feature:**

- In Flash MX Professional 2004, select Window > Properties.

Flash Lite 1.1 for mobile phones supports static, input, and dynamic text areas when using the Alias Text option.

### Alias text rendered in Flash MX Professional 2004

The Alias Text option makes text more readable by aligning text outlines along pixel boundaries. This makes the text appear aliased, even when anti-aliasing is enabled.

## Pixel fonts

It is very important to use the right type fonts for the Flash Lite content you intend for display on mobile phones, which have small screens. Standard fonts such as Arial or Verdana are not easy to read, because Flash Player handles anti-aliasing in all but the low-quality mode. In this case, you should consider using pixel fonts that are displayed without anti-aliasing.

Pixel fonts make text more readable because text outlines are aligned along pixel boundaries. Because these fonts use pixels to create each character, they remain sharp and easy to read. They can be used on all types of screen displays, regardless of the screen resolution. The font sizes need to be in increments of 8 points (8, 16, 24, and so on) to remain crisp and legible. Use an 8-point font to get the maximum amount of text on the screen yet keeping it legible.

When using pixel fonts, follow these guidelines:

- Place text on absolute x and y values (10.0, not 10.2, for example).
- If you create input or dynamic text boxes, make sure you embed your fonts. Otherwise, your Flash content is displayed in the default system fonts.
- To make your text stand out, use a combination of different fonts, bold and normal styles, and contrasting colors.

For more information about pixel fonts, see: www.miniml.com, www.fontsforflash.com, and www.ultrafonts.com.

## ActionScript and properties

Flash Lite 1.1 supports most Flash 4 ActionScript commands. The following are notable exceptions:

- Use the `add` operator instead of the `&` command to concatenate strings.
- Button mouse events such as `dragOver`, `dragOut`, and `releaseOutside` cannot be used to trigger ActionScript code attached to buttons. However, in addition to keypress events, you can use the `press`, `release`, `rollOver`, and `rollOut` events to trigger ActionScript when attached to buttons and accessed through key-based navigation.
- Draggable movie clip functions and properties (for example: `startDrag`, `stopDrag`, and `_dropTarget` properties) are not supported.
- Use the `eq` operator to compare strings and the == operator for numeric comparison.
- URL encoding must be done manually using ActionScript. The `escape()` ActionScript function is not a Flash 4 function and is not available in Flash Lite 1.1.
- Two new FSCommand2 commands have been added to encode a string into a format that is safe for network transfer: `Escape` and `Unescape`. For more information, see Chapter 5, "New FSCommand and FSCommand2 commands" .
- The default Quality level for Flash Lite during playback is medium, and there is no support for bitmap smoothing.
- Flash Lite 1.1 supports `loadMovie()`, `loadMovieNum()`, `loadVariables()`, and `loadVariablesNum()`. Only one LoadMovie and LoadVars action is processed per frame or per event handler. Certain handsets restrict these actions to keyEvents only, in which case the action call is processed only if it is triggered in a keyEvent handler. Even under such circumstances, only one such action is processed per event handler. For more information, see Appendix A, "Supported ActionScript" .
- Only one `getURL()` action is processed per frame or per event handler. Certain handsets restrict the `getURL()` action to keyEvents only, in which case the `getURL()` call is processed only if it is triggered in a keyEvent handler. Even under such circumstances, only one `getURL()` action is processed per event handler.

  An example of using the `tel` protocol would be the following:

```
on (release, keyPress "#"){
  getURL("tel:117");
}
```

- A button action can be assigned to launch an e-mail composition window with the address, subject, and body text fields already populated. There are two ways to do this: Method 1 can be used for either Shift-JIS or English character encoding, while method 2 supports only English character encoding.

  **Method 1**

  Set variables for each of the desired parameters, for example:

```
on (release, keyPress "#"){
  subject = “email subject”;
  body = “email body”;
  getURL(“mailto:somebody@anywhere.com”, ““, “GET”);
}
```

  **Method 2**

  Define each parameter within the getURL action, for example:

```
on (release, keyPress "#"){

  getURL(“mailto:somebody@anywhere.com?cc=cc@anywhere.com&bcc=bcc@anywhere.
  com&subject=I am the email subject&body=I am the email body”);
}
```

  Method 1 results in automatic URL encoding while method 2 preserves the spaces in the strings. For instance, the resulting string of using method 1 is as follows:

```
email+subject
email+body
```

  whereas method 2 results in the following strings:

```
email subject
email body
```

- Key events can be attached only to the keys 0-9, #, *, and the Enter key.
- Sound functionality is limited to event sound. Only the first event sound in a keypress statement block is played, and all subsequent sounds in the same block are ignored.
- The range of valid integers that can be represented is -2,147,483,648 to 2,147,483,647.
- Math functions are not natively supported. In Flash Lite, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported in Flash Player 5 and later.
- The `_url` property is not supported.
- The `Number()` and `String()` functions are not supported.

  ***Note:*** Flash 4 ActionScript does not support arrays. However, they can be emulated using the `eval()` function. For more information, see Macromedia TechNote 14219, “How to use Eval to emulate an array,” at www.macromedia.com/go/flash_support (English) or www.macromedia.com/go/flash_support_jp (Japanese).

ActionScript commands that are not recognized are ignored. For a detailed listing of supported ActionScript and properties, see Appendix A, “Supported ActionScript” and Appendix B, “Supported ActionScript Properties” .

## Sound

Using audio in Flash content helps to create a richer user experience that goes beyond a typical mobile phone application. For more information about embedding sound into Flash Lite content for mobile phones, see Chapter 3, "Working with Sound" .

## Network access

It's possible for Flash content that resides on a mobile phone to download new data from a web server by using various functions, which are described below.

The Flash Lite 1.1 specification supports the `getURL()`action which is processed once per frame or per event handler. The `getURL()` action can be associated with the following keys: 0-9, *, #, or the Select key. Only the first `getURL()` call in a keypress statement block is executed; all subsequent `getURL()` calls in the same block are ignored.

The `getURL()` function can be used to load another SWF or HTML page (`http`), a secured (SSL-Secure Sockets Layer) HTTP page (`https`), send e-mail (`mailto`), or dial a phone number (`tel`).

With Flash Lite 1.1, it is possible to load data and SWF files from a web server using the `loadMovie()`, `loadMovieNum()`, `loadVariables()`, and `loadVariablesNum()` functions. By using these functions you can update Flash content that resides on a mobile phone. These actions will be processed once per frame or per event handler.

## SWF file size and memory

Supported mobile phones impose limitations on the size of Flash Lite SWF files and on the amount of runtime memory they use. The SWF file size is a larger issue for mobile phones than for desktop computers because mobile phones don't have as much RAM as desktop computers.

There is a prescribed limit on how large a web page can be, whether or not it includes Flash Lite content. For most mobile phones, this limit is 100 KB.

The runtime memory available to Flash Lite applications running on mobile phones is limited and might vary among models. Generally, for mobile phones, this limit is not less than 1 MB. Because Flash MX Professional 2004 does not provide a mechanism for checking a phone's runtime memory consumption, Macromedia strongly recommends that you test all content on actual mobile phones.

## Performance optimization

CPU speed in mobile phones varies among models and is typically much slower than the CPU speed in current desktop computers. Therefore, it is extremely important to consider application performance and optimization from the beginning of each project for creating Flash Lite content created for mobile phones.

***Note:*** In Flash MX Professional 2004, you can find tips on optimizing Flash applications. (Select Help › Using Flash -> Search and enter optimizing movies in the Keyword Searchtext box.)

If you follow the simple guidelines described in this document to author your Flash Lite content, you can create rich and compelling content despite CPU limitations.

## Animation

When creating animated content for a mobile phone, it is important to keep in mind the phone's CPU limitations. The following guidelines can help prevent your Flash Lite content from running slowly:

- If you need to provide intense or complex animation, experiment with changing the quality setting of the content. The default quality setting is Medium.

  To change the quality setting in Flash MX Professional 2004, select File > Publish Settings, and select the HTML tab. Select a quality setting from the Quality pop-up menu.

  Because changing the quality setting might noticeably affect the visual quality of the Flash Lite content, make sure to thoroughly test the SWF file.
- You can also use ActionScript to control the rendering quality of a SWF file, by using either the `_quality` property or the new FSCommand2 `setQuality()` function.

  For the `_quality` property, valid values are `LOW`, `MEDIUM`, and `HIGH`. The following code sets the rendering quality to `LOW`:

  ```
  _quality = "LOW";
  ```

  For more information about the `setQuality` function, see Chapter 5, "New FSCommand and FSCommand2 commands" .
- Limit the number of simultaneous tweens.
- Use Alpha effects on symbols sparingly, as they are very CPU intensive. In particular, it is generally not a good idea to tween symbols that have alpha levels that are not fully opaque (less than 100%).
- Avoid intensive visual effects. These include large masks, extensive motion, alpha blending, extensive gradients, and complex vectors.
- Although animating with ActionScript may produce more desirable results, in general, you should avoid unnecessary use of ActionScript because it can become processor intensive.
- Experiment with combinations of tweens, key frame animations, and ActionScript-driven movement to produce the most efficient results.
- If possible, test animations frequently on your target phones.

## Bitmap graphics

Macromedia recommends optimizing bitmap graphics to 16 bits before importing them into Flash MX Professional 2004. Doing so reduces Flash Lite movie size and gives you more control over the final output. Also, make sure that bitmaps are imported at the size they need to be in the Flash Lite movie. Using larger than required bitmaps results in higher runtime memory requirements.

## Bitmap versus vector graphics

Flash Lite generally uses vector graphics to define content, which can tax a phone's CPU when rendering complex graphics and animations. In general, the more vectors that are manipulated on the Stage, the more CPU power is required. This is also true for Flash movies delivered on desktop computers. However, a mobile phone is far less powerful than desktop computer, so you should avoid taxing the CPU.

When creating content for mobile phones, it is sometimes better to use bitmaps instead of vectors because they require less CPU power to animate. For example, a road map of a large city would have too many complex shapes to scroll and animate well on a mobile phone if it were created as a vector graphic; a bitmap would work much better.

Using bitmaps produces larger files than using vector images, so take care during development to find the right balance of CPU versus file size and runtime memory requirements. Because of mobile phones' smaller screens, slower data transmission speeds, limited memory, and slower CPU speeds, you should take extra care in planning and testing.

If you are using bitmaps, you can set image compression options that reduce your SWF file size.

**To set bitmap image compression:**

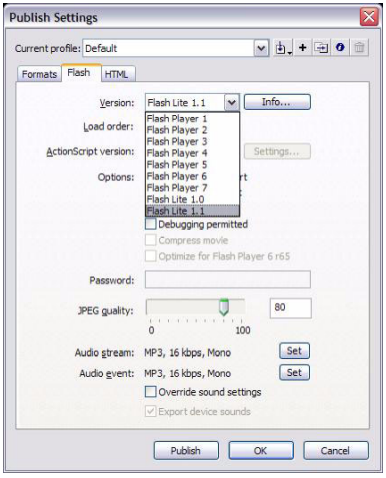
1. Start Flash and create a new document.
2. Select a bitmap in the Library window.
3. Right-click (Windows) or Control-click (Macintosh) the bitmap's icon in the Library window.
4. Select Properties from the options menu. The Bitmap Properties dialog box appears:

5. In the Compression pop-up menu, select one of the following options:
   - Select Photo (JPEG) for images with complex color or tonal variations, such as photographs or images with gradient fills. This option produces a JPEG format file. Select the Use Imported JPEG Data check box to use the default compression quality specified for the imported image. To specify a new quality compression setting, deselect Use Imported JPEG Data and enter a value between 1 and 100 in the Quality text box. A higher setting produces a higher image quality, but also a larger file size, so adjust the value accordingly.
   - For images with simple shapes and relatively few colors, select Lossless (PNG/GIF) to compress the image with lossless compression, in which no data is discarded from the image. Save the bitmap as a PNG file.
6. Click Test to determine the results of the file compression. Compare the original file size to the compressed file size to determine if the selected compression setting is acceptable.

You can also globally adjust the compression settings for JPEG files.

**To globally set bitmap compression for JPEG files:**

1. Select File > Publish Settings, and then select the Flash tab. The Publish Settings dialog box with the Flash tab options appears:




2. Adjust the JPEG Quality slider or enter a value.

   A higher JPEG quality value results in a higher image quality, but a larger SWF file size. As with the compression settings previously described, lower image quality produces a smaller SWF file. Try different settings to determine the best trade-off between size and quality.

### Vector graphics

Whenever possible, do not use borders in your vector graphics as this greatly diminishes the number of rendered lines.

### Optimizing ActionScript

Because of CPU limitations, you should follow these general guidelines when developing ActionScript for Flash Lite content deployed on mobile phones:

- Keep the ActionScript as simple as possible.
- Limit the number of loops that you use and the amount of code that each loop contains.
- Stop frame-based looping as soon as it is no longer needed.
- Avoid string and emulated array processing—it can be extremely CPU intensive.

  ***Note:*** Flash 4 ActionScript does not support arrays. However, they can be emulated using the `eval()` function. For more information, see Macromedia TechNote 14219, "How to use Eval to emulate an array," at www.macromedia.com/go/flash_support (English) www.macromedia.com/go/flash_support_jp (Japanese).

## Device speed and frames per second

If the project contains static images, it's not likely that the device processor speed will be an issue. The complexity of Flash requires some important trade-offs when developing content for mobile phones. Until mobile phones have faster processors and there are improvements to other internal components, you must make adjustments to provide an experience that does not appear sluggish to users; otherwise, they won't use the application.

Try to avoid full-screen wipes, fades, and animations. Remember that updating many pixels at a time can be slow, depending on the content. The performance of your Flash application depends on the number of open applications, available phone memory, processor speed, and screen resolution.

## Development checklist

When you develop Flash content for mobile phones, make sure to check the following items:

- Does the Flash content work?
- Is the Flash content intuitive and easy to interact with?
- Does the Flash content load data and SWF files without any problems?
- Can you optimize the images or rewrite code to further reduce the file size and memory requirements while improving performance?
- Are all bitmap images successfully decoded and rendered on the mobile phone?

# CHAPTER 3
# Working with Sound

This section describes the various aspects of sound in relationship to Macromedia Flash Lite 1.1 for the mobile phones.

## Audio formats

Flash Lite 1.1 supports MIDI, MFi, SMAF, uncompressed PCM (or WAV), compressed ADPCM, and compressed MP3 audio formats.

## Event sound

Event sound is the ability to play sound independent of the Timeline; any event can be used to trigger an event sound. Event sound data must download completely before it begins playing, and it continues playing until either the end of the sound buffer has been reached or it is explicitly stopped. It is possible to loop event sounds within a SWF file.

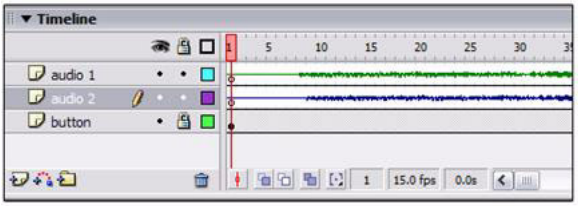
## Streaming sound

Streaming sounds begin playing as soon as enough data for the first few frames has been downloaded; stream sounds are synchronized to the Timeline for playing on a mobile phone.

Flash Lite 1.1 supports uncompressed PCM (or WAV), compressed ADPCM, and compressed MP3 audio formats for streaming sound.

## Embedding sound

Because Flash MX Professional 2004 does not natively support certain audio formats such as MIDI or SMAF, you must temporarily substitute a proxy sound in a recognized format such as MP3. You can use options in the Sound Properties dialog box and the Flash Publish Settings dialog box to link the proxy sound file to a MIDI file.




*Sound files that have been substituted are displayed in green; blue sound waves are files that haven't been substituted.*

For information on how to substitute sounds in your Flash Lite content, see Chapter 6, "Creating Content" .

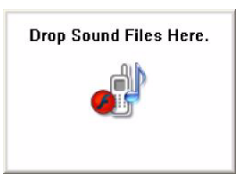
## Compound sound

Flash Lite 1.1 provides the ability to encapsulate device-specific sounds of multiple formats into a single tagged data block. This provides content developers with the ability to create a single piece of content that is compatible with multiple devices. As an example, a single Flash movie can contain the same sound represented in both MIDI and MFi formats. This Flash movie can be played back both on a device that supports only MIDI and on a device that supports only MFi, with each device playing back the specific sound format that it natively supports.

During content creation, content developers identify the sound files in the formats that they want to bundle together. An external tool (FlashLiteSoundBundler.exe) is available to bundle the identified sound files into one sound data block, to be played when triggered by an event. When the appropriate event is triggered, Flash Lite 1.1 processes this bundled sound data block and plays the sound data in the specific format supported by the device. The sound bundle file generated by the FlashLiteSoundBundler.exe program creates a file with the extension .fls.

**The steps to create a Sound Bundle File are:**

1. Launch FlashLiteSoundBundler.exe.
2. Drag and drop a sound file to be bundled into the target window.




*The FlashLiteSoundBundler.exe allows you to create compound sounds.*

***Note:*** Right click on this window to trigger the Exit button.

3. Flash Lite 1.1 Compound Information window will launch.
4. Drag and drop the rest of the sound files to be bundled.
5. Click on Save Bundle to save your Sound Bundle File in a specific location.

When the appropriate event is triggered, Flash Player processes this bundled sound data block and plays the appropriate sound data contained in the sound bundle.

For information on how to substitute sounds in your Flash Lite content, see Chapter 6, "Creating Content" .

## Adding a Sound Bundle File to a Flash document

With Flash MX Professional 2004, you can include event sounds when authoring documents for playback on mobile devices. The general process is described in this section. For detailed information on authoring for mobile devices, see the Content Development Kits on the Mobile and Devices Development Center at www.macromedia.com/devnet/devices.

Flash does not support sound file formats used for mobile devices (such as MIDI and others); when authoring for mobile devices, you must temporarily place a proxy sound in a supported format such as MP3, WAV, or AIFF in the Flash document. The proxy sound in the document is then linked to an external mobile device sound, such as a MIDI file or a Sound Bundle file. During the document publishing process, the proxy sound is replaced with the linked external sound. The SWF file generated contains the external sound or sound bundle data and processes it appropriately for playback on a mobile device.

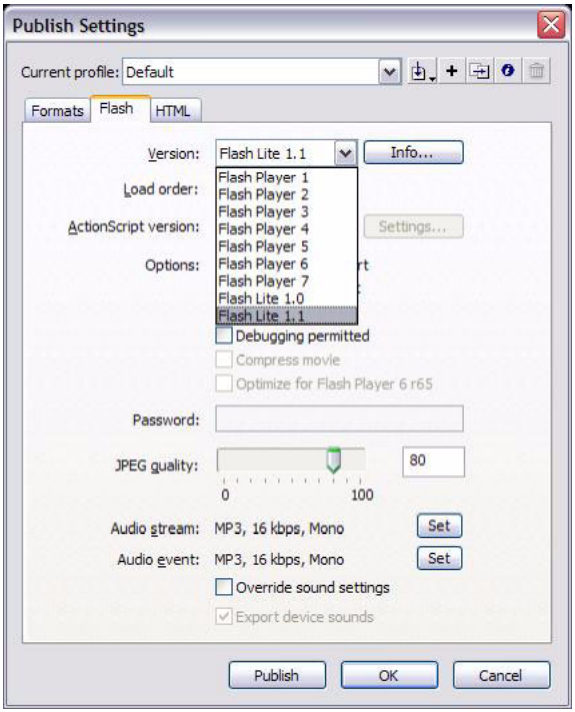
When adding device specific sounds or sound bundles to Flash documents for playback on mobile devices, keep the following in mind:

- This feature works with event sounds only.
- The Effect, Sync, and Edit options are not supported when linking a sound.
- You must specify an external device sound file for each sound in a document.
- As with all external files, the device sound file or the sound bundle file must be available during the publishing process but is not needed by the SWF file for playback.

**The steps to add a Sound Bundle File to a Flash document are:**

1. Import a sound file to the library in the Flash document (File > Import > Import to Library).
2. In the Library panel, right-click (Windows) or Control-click (Macintosh) the sound and select Properties.
3. In the Device sound text box, enter a path or click the folder icon and browse to the location where the Sound Bundle File is located. Click OK to close the Property inspector.
4. Add a button instance to the Stage from the Buttons common library (Window > Other Panels > Common Libraries > Buttons).
5. Add the linked sound to the Hit frame of the button.
6. Open the Publish Settings dialog box (File > Publish Settings), and click the Flash tab.

7. Select Flash Lite 1.1 from the version menu.




8. The SWF file now contains the linked Sound Bundle File.
9. Select Control > Test Movie to test your Flash application.
10. Select File > Publish to save the SWF file that contains the Sound Bundle File created earlier.

# CHAPTER 4
# ActionScript Enhancements for Flash Lite 1.1

Macromedia Flash Lite 1.1 supports two new ActionScript functions: `FSCommand()` and `FSCommand2()`. Many new `FSCommand` and `FSCommand2` commands have been introduced in Flash Lite 1.1 For a complete list of ActionScript expressions supported on mobile phones, see Appendix A, "Supported ActionScript" .

## New ActionScript functions

Almost all of these new ActionScript functions are available only for creating Flash Lite 1.1 content; however, not all of them are applicable to all mobile phones. Be sure to check the functions and commands you plan on using before integrating them with Flash Lite content for specific mobile phones.

### FSCommand()

Flash Lite 1.1 supports the `FSCommand()` function, which enables Flash Lite content to communicate with Macromedia Flash Player, the host application, and the device hosting the player.

### FSCommand2()

The `FSCommand2()` function is a new ActionScript function that is supported in Flash Lite 1.1 but is not yet supported in the standard desktop version of Flash Player. The `FSCommand2()` and `FSCommand()` provide similar functionality, with the following main differences:

- `FSCommand2()` can take an arbitrary number of arguments.
- During the playback of a Flash application, the `FSCommand2()` function is executed immediately, whereas `FSCommand()` is executed at the end of the frame being processed.
- The `FSCommand2()` function returns a value that can be used to report success, failure, or the result of the command.

See Chapter 5, "New FSCommand and FSCommand2 commands" for more information.

## Platform capabilities and variables

The following variables are used to specify whether certain capabilities are available in Flash Lite, the device, the host application, or Flash Player.

### _capCompoundSound

The `_capCompoundSound` variable indicates whether Flash Lite can process compound sound data. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

**Example**

```
mVarValue = _capCompoundSound;
```

### _capEmail

The `_capEMail` variable indicates whether Flash Lite can send e-mail messages by means of the `GetURL()` ActionScript command. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

**Example**

```
myVarValue = _capEmail;
```

### _capMMS

The `_capMMS` variable indicates whether Flash Lite can send MMS messages by using the `GetURL()` ActionScript command. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

**Example**

```
myVarValue = _capMMS;
```

### _capSMS

The `_capSMS` variable indicates whether Flash Lite can send SMS messages by using the `GetURL()` ActionScript command. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

**Example**

```
myVarValue = _capSMS;
```

### _capStreamSound

The `_capStreamSound` variable indicates whether the device can playing streaming (synchronized) sound. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

**Example**

```
myVarValue = _capStreamSound;
```

## $version

The `$version` variable contains the version number of Flash Lite. It contains a major number, minor number, build number, and an internal build number, which is generally 0 in all released versions (for example, 5,2,1,141).

**Example**

```
myVarValue = $version;
```

## _capMFi

The `_capMFi` variable indicates whether the device can play sound data in the MFi audio format. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

**Example**

```
myVarValue = _capMFi;
```

## _capMIDI

The `_capMIDI` variable indicates whether the device can play sound data in the MIDI audio format. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

**Example**

```
myVarValue = _capMIDI;
```

## _capSMAF

The `_capSMAF` variable indicates whether the device can play sound data in the SMAF audio format. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

**Example**

```
myVarValue = _capSMAF;
```

## _capLoadData

The `_capLoadData` variable indicates whether the host application can dynamically load additional data through calls to `loadMovie()`, `loadMovieNum()`, `loadVariables()`, and `loadVariablesNum()` functions. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

**Example**

```
myVarValue = _capLoadData;
```

### _cap4WayKeyAS

The `_cap4WayKeyAS` variable indicates whether Flash Player executes ActionScript expressions attached to key event handlers associated with the Right, Left, Up and Down keys. This variable is defined and has a value of 1 only when the host application uses four-way key navigation mode to navigate between Flash controls (buttons and input text fields). Otherwise, this variable is undefined.

If the value of this variable is 1 and one of the four-way keys is pressed, Flash Player first looks for a handler for that key. If none is found, Flash control navigation is performed. However, if an event handler is found, no navigation action occurs for that key. In other words, the presence of a keypress handler for a Down key disables the user's ability to navigate down.

**Example**

```
myVarValue = _cap4WayKeyAS;
```

## New ActionScript properties

The following properties are new in ActionScript.

### scroll

You can use the `scroll` property to retrieve and set a text field. When the `scroll` property of a text field is retrieved, it indicates the number of the line currently displayed as the first line in the text field's viewable area. When you set the `scroll` property to a specific value, the text field scrolls so that the line with that number appears at the top of the field's viewable region. This property is normally used with the `maxscroll` property to create text-scrolling interfaces.

**Example**

```
on (release) {
  myText.scroll = myText.scroll + 1;
  }
```

### maxscroll

The `maxscroll` property returns the largest allowable scroll value for a text field. It represents the number of the last line in a text field that can be used as the top line in its viewable region. This property can be used with the `scroll` property with a function to create text-scrolling interfaces.

**Example**

```
textBoxMax = myText.maxscroll
```

# CHAPTER 5
# New FSCommand and FSCommand2 commands

This chapter discusses the new `FSCommand()` and `FSCommand2()` commands in Macromedia Flash Lite 1.1. These new commands fall into these categories: general commands, commands controlling Flash playback, and platform integration commands.

## General commands

The commands in this section provide general control of Flash Lite content on mobile phones.

### URL Encoding

Two new commands have been added to encode a string into a format that is safe for network transfer to a server and back to the mobile phone: `Escape` and `Unescape`.

### Escape

The `Escape` function encodes an arbitrary string into a format that is safe for network transfer. All characters that are not alphanumeric are replaced with a hexadecimal escape sequence (%xx, or %xx%xx in the case of double-byte characters). The encoded string is returned in a variable that is passed into the SWF file by name.

This function is executed immediately upon invocation.

#### Syntax

```
status = FSCommand2( "Escape", original, encoded )
```

In this code example, `original` is the string to be encoded into a format safe for URLs, and `encoded` is the resulting encoded string.

#### Return value

A value of 0 upon failure; 1 upon success.

## Unescape

The `Unescape` function decodes an arbitrary encoded string that is safe for network transfer into its normal form. All characters that are in hexadecimal format, that is, a percent character (%) followed by two hexadecimal digits, are converted into their decoded form. The decoded string is returned in a variable that is passed in by name.

This function is executed immediately upon invocation.

### Syntax

```
status = FSCommand2( "Unescape", original, encoded )
```

In this example, `original` is the string to be decoded from a format safe for network transfer and `encoded` is the resulting decoded string.

### Return value

A value of 0 upon failure; 1 upon success.

### Example

```
original_string = "hello, how are you?";
status = fscommand2("Escape", original_string, "encoded_string");
original_string2 = "Hello%7B%5BWorld%5D%7D";
status = fscommand2("Unescape", original_string2, "normal_string");
```

## Input text fields

The commands in this section control the input text fields of Flash content on mobile phones.

## SetInputTextType()

In Flash Lite, input text functionality is supported by asking the host application to start the generic, device-specific, text-input interface, often referred to as the Front End Processor (FEP). The `SetInputTextType()` function specifies the mode in which the input text field should be opened. The available options are `Numeric`, `Alpha`, `Alphanumeric`, `Latin`, `NonLatin` and `NoRestriction`.

These options are mutually exclusive and cannot be combined. When this command is not used, the FEP is opened in default mode. The following rules apply when the following text-input interface options are not supported on certain mobile phones:

- If `Numeric` mode is not supported, the FEP is opened in `Alphanumeric` mode.
- If `Alpha` mode is not supported, the FEP is opened in `Alphanumeric` mode.
- If `Alphanumeric` mode is not supported the FEP is opened in `Latin` mode.
- If `Latin` mode is not supported, the FEP is opened in `NoRestriction` mode.

  Similarly, if `NonLatin` mode is not supported, the FEP is opened in `NoRestriction` mode.

  ***Note:*** Not all mobile phones support these input text field types. For this reason, you must validate the input text data.

The `SetInputTextType()` function is executed immediately upon invocation.

#### Syntax

```
status = FSCommand2( "SetInputTextType", variableName, type )
```

In the preceding example, `variableName` is the name of the variable associated with the input text field and `type` is one of the following values:

`Numeric`: sets the FEP to numbers only mode [0-9].

`Alpha`: sets the FEP to alpha characters only mode [A-Z, a-z].

`Alphanumeric`: sets the FEP to alphanumeric characters only mode [0-9, A-Z, a-z].

`Latin`: sets FEP to Latin characters only mode [Alphanumeric and punctuation].

`NonLatin`: sets FEP to non-Latin characters only mode [example: Kanji and Kana].

`NoRestriction`: sets no restriction on the FEP—the FEP is started in default mode.

#### Return value

A value of 0 upon failure; 1 upon success.

#### Example

```
status = fscommand2("SetInputTextType", "input1", "Numeric");
```

## Controlling Flash playback

The commands in this section control the playback of Flash content on mobile phones.

### Display

The commands in this section control the display aspect of Flash content on mobile phones.

#### FullScreen()

The `FullScreen()` function sets the size of the display area to be used for rendering. The size can be either full screen or less than full screen. Set the `size` argument to `true` to indicate full screen and to `false` otherwise.

The `FullScreen()` function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

This command is supported only when Flash Lite is running in stand-alone mode. It is not supported when the player is running in the context of another application (for example, as a plug-in to a browser).

#### Syntax

```
status = FSCommand2( "FullScreen", size )
```

In this example, `size` is either a defined variable or a constant string value (for example, `"true"`).

#### Return value

A value of -1 if the function is not supported; 0 if it's supported.

#### Supported applications

This feature is not supported in all mobile phones.

### SetQuality()

The `SetQuality()` function sets the quality of the rendering of the animation. The value of the `quality` argument must be `high`, `medium`, or `low`.

The `SetQuality()` function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "SetQuality", quality )
```

Here, `quality` is either a defined variable or a constant string value (for example, `"medium"`).

#### Return value

A value of -1 if the function is not supported; 0 if it's supported.

## Key configuration

The commands in this section describe how to control the soft keys for Flash content on mobile phones.

### SetSoftKeys()

The `SetSoftKeys()` function is used to remap the left and right soft keys of mobile phones, provided that they can be accessed and remapped. The `left` and `right` parameters to this command specify the text to be displayed for the left and right soft keys, respectively. After this function is executed, pressing the left key generates a `PageUp` keypress event, and pressing the right key generates a `PageDown` keypress event. ActionScript associated with the `PageUp` and `PageDown` keypress events is executed when the respective key is pressed.

This function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

This function is supported only when Flash Lite is running in stand-alone mode. It is not supported when the player is running in the context of another application (for example, as a plug-in to a browser).

#### Syntax

```
status = FSCommand2( "SetSoftkeys", left, right )
```

In this example, `left` and `right` are either defined variables or constant string values (for example, `"label"`)

#### Return value

A value of -1 if the function is not supported; 0 if it's supported.

#### Supported applications

This feature is not supported in all mobile phones.

### ResetSoftKeys()

The `ResetSoftKeys()` function resets the soft keys to their original settings. It is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

The `ResetSoftKeys()` function is supported only when Flash Lite is running in stand-alone mode. It is not supported when the player is running in the context of another application (for example, as a plug-in to a browser).

#### Syntax

```
status = FSCommand2( "ResetSoftKeys" )
```

#### Return value

A value of -1 if the function is not supported; 0 if it's supported.

#### Supported applications

This feature is not supported in all mobile phones.

## Player operation commands

The commands in this section provide the mobile phone's memory value to Flash content on the mobile phone.

### GetFreePlayerMemory()

The `GetFreePlayerMemory()` function returns the amount of memory, in kilobytes, currently available to Flash Lite. This function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetFreePlayerMemory" )
```

#### Return value

A value of -1 if the function is not supported; otherwise, the amount of memory available, in kilobytes.

### GetTotalPlayerMemory()

The `GetTotalPlayerMemory()` function returns the total amount of memory, in kilobytes, allocated to Flash Lite. This function is executed immediately upon invocation. If the function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetTotalPlayerMemory" )
```

#### Return value

A value of -1 if the function is not supported; otherwise, the amount of memory available, in kilobytes.

### Launch()

This function starts another application on the mobile phone. The name of the application being launched and the parameters to it, separated by commas, are passed in as a single parameter.

***Note:*** This feature is operating-system dependent.

The `launch()` function is supported only when Flash Lite is running in stand-alone mode. It is not supported when the player is running in the context of another application (for example, as a plug-in to a browser).

#### Syntax

```
status = FSCommand( "Launch", "application-path,arg1,arg2,...,argn" )
```

#### Supported applications

This feature is not supported in all mobile phones.

### Quit()

The `Quit()` function causes Flash Player to stop playback and exit. It is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

The `Quit()` function is supported only when Flash Lite is running in stand-alone mode. It is not supported when the player is running in the context of another application (for example, as a plug-in to a browser).

#### Syntax

```
status = FSCommand2( "Quit" )
```

#### Return value

A value of -1 if the function not supported.

#### Supported applications

This feature is not supported in all mobile phones.

## Platform integration commands

A standard set of commands has been created to get and set platform-specific information. These include information such as current time and date, network status, signal strength, battery level, and so on. The implementations of these commands all rely on either `FSCommand` or `FSCommand2` commands.

### Date and time

The commands in this section provide the mobile phone's date and time information to Flash content on the mobile phone.

### GetDateDay()

The `GetDateDay()` function returns the day of the current date. It is a numeric value (without a leading zero). Valid days are 1–31.

The `GetDateDay()` function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetDateDay" )
```

#### Return value

A value of -1 if the function is not supported; otherwise, the current day, returned as a number (1-31).

### GetDateMonth()

The `GetDateMonth()` function returns the month of the current date. It is a numeric value (without a leading zero). Valid months are 1–12.

The `GetDateMonth()` function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetDateMonth" )
```

#### Return value

A value of -1 if the function is not supported; otherwise, the current month, returned as a number (1-12).

### GetDateWeekday()

The `GetDateWeekday()` function returns a numeric value that is the name of the day of the current date. Valid days are 0–6, where 0 represents Sunday, 1 represents Monday, 2 represents Tuesday, 3 represents Wednesday, 4 represents Thursday, 5 represents Friday, and 6 represents Saturday.

The `GetDateWeekday()` function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetDateWeekday" )
```

#### Return value

A value of -1 if the function is not supported; otherwise, the current weekday, returned as a number (0-6).

### GetDateYear()

The `GetDateYear()` function returns a numeric, four-digit value that is the year of the current date.

The `GetDateYear()` function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetDateYear" )
```

#### Return value

A value of -1 if the function is not supported; otherwise, the current year, returned as a number (for example, 2004).

### GetLocaleLongDate()

The `GetLocaleLongDate()` function sets a parameter to a string representing the current date, in long form, formatted according to the currently defined locale. The parameter is passed in by name. The value returned through it is a multiple-character, variable-length string. The actual formatting depends on the mobile phone and the locale.

The `GetLocaleLongDate()` function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetLocaleLongDate", "longdate" )
```

#### Return value

A value of -1 if the function is not supported; 0 if it's supported.

Sample resultant values for `longdate`:

```
October 16, 2004
16 October 2004
```

### GetLocaleShortDate()

The `GetLocaleShortDate()` function sets a parameter to a string representing the current date, in abbreviated form, formatted according to the currently defined locale. The parameter is passed in by name. The value returned is a multiple-character, variable-length string. The actual formatting depends on the mobile phone and the locale.

The `GetLocaleShortDate()` function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetLocalShortDate", "shortdate" )
```

#### Return value

A value of -1 if the function is not supported; 0 if it's supported.

Sample resultant values for `shortdate`:

```
10/16/2004
16-10-2004
```

## GetLocaleTime()

The `GetLocaleTime()` function sets a parameter to a string representing the current time, formatted according to the currently defined locale. The parameter is passed in by name. The value returned is a multiple-character, variable-length string. The actual formatting depends on the mobile phone and the locale.

The `GetLocaleTime()` function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

### Syntax

```
status = FSCommand2( "GetLocalTime", "time" )
```

### Return value

A value of -1 if the function is not supported; 0 if it's supported.

Sample resultant values for `time`:

```
6:10:44 PM
18:10:44
```

## GetTimeHours()

The `GetTimeHours()` function returns the hour of the current time of day, based on a 24-hour clock. It is a numeric value (without a leading zero). Valid hours are 0–23.

The `GetTimeHours()` function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

### Syntax

```
status = FSCommand2( "GetTimeHours" )
```

### Return value

A value of -1 if the function is not supported; otherwise, the current hour, returned as a number (0-23).

## GetTimeMinutes()

The `GetTimeMinutes()` function returns the minute of the current time of day. It is a numeric value (without a leading zero). Valid minutes are 0–59.

The `GetTimeMinutes()` function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

### Syntax

```
status = FSCommand2( "GetTimeMinutes" )
```

### Return value

A value of -1 if the function is not supported; otherwise, the current minute, returned as a number (0-59).

### GetTimeSeconds()

The `GetTimeSeconds()` function returns the second of the current time of day. It is a numeric value (without a leading zero). Valid seconds are 0–59.

The `GetTimeSeconds()` function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetTimeSeconds" )
```

#### Return value

A value of -1 if the function is not supported; otherwise, the current second, returned as a number (0-59).

### GetTimeZoneOffset()

The `GetTimeZoneOffset()` function sets a parameter to the number of minutes between the local time zone and universal time (UTC). The parameter is passed in by name. The value returned is numeric, and may be a positive or negative number.

The `GetTimeZoneOffset()` function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetTimeZoneOffsett", "timezoneoffset" )
```

#### Return value

A value of -1 if the function is not supported; 0 if it's supported.

Sample resultant values for `timezoneoffset`:

`540`: Japan standard time

`-420`: Pacific daylight savings time

## Volume

The commands in this section provide the mobile phone's volume information to Flash content on the mobile phone.

### GetMaxVolumeLevel()

The `GetMaxVolumeLevel()` function returns the maximum volume level of the mobile phone. It is a numeric value greater than zero.

The `GetMaxVolumeLevel()` function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetMaxVolumeLevel" )
```

#### Return value

A value of -1 if the function is not supported; otherwise, the maximum volume level, returned as a number.

### GetVolumeLevel()

The `GetVolumeLevel()` function returns the current volume level of the mobile phone. It is a numeric value, in the range of 0 to the maximum value returned by `GetMaxVolumeLevel`.

The `GetVolumeLevel()` function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetVolumeLevel" )
```

#### Return value

A value of -1 if not supported; otherwise, the volume level, returned as a number.

## Vibrate

The commands in this section provide the mobile phone's vibration information to Flash content on the mobile phone.

### StartVibrate()

The `StartVibrate()` function starts the phone's vibration feature. The pulse of the vibration is specified by an "on" time followed by an "off" time. Both the on time and the off time are specified in milliseconds, and neither can exceed 5 seconds. The pulse can be repeated sequentially, up to three times.

If a vibration is already occurring, that vibration is stopped before the new specified one is started. Vibrations are also stopped when the Flash application playback is stopped or paused, and when Flash Player is exited.

This function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "StartVibrate", time_on, time_off, repeat )
```

In this example, `time_on` is the amount of time in milliseconds (the maximum is 5 seconds) that the vibration is on, `time_off` is the amount of time in milliseconds (the maximum is 5 seconds) that the vibration is off, and `repeat` is the number of times (a maximum of three) to repeat this vibration.

#### Return value

A value of -1 if the function is not supported; 0 if the vibration is started; 1 if an error occurred and the vibration could not be started.

### StopVibrate()

The `StopVibrate()` function stops the current vibration, if any.

This function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "StopVibrate" )
```

#### Return value

A value of -1 if the function is not supported; 0 if the vibration is stopped.

## Power

The commands in this section provide the mobile phone's power information to Flash content on the mobile phone.

### GetBatteryLevel()

The `GetBatteryLevel()` function returns the current battery level. It is a numeric value, in the range of 0 to the maximum value returned by the `GetMaxBatteryLevel()`.

This function is executed immediately upon invocation. If this function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetBatteryLevel" )
```

#### Return value

A value of -1 if not supported; otherwise, the battery level, returned as a number.

### GetMaxBatteryLevel()

The `GetMacBatteryLevel()` function returns the maximum battery level of the mobile phone. It is a numeric value greater than zero.

This function is executed immediately upon invocation. If the `GetMacBatteryLevel()` function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetMaxBatteryLevel" )
```

#### Return value

A value of -1 if the function is not supported; otherwise, the maximum battery level, returned as a number.

### GetPowerSource()

The `GetPowerSource()` function returns a value indicating whether the power source is currently supplied a battery or externally supplied.

This function is executed immediately upon invocation. If the `GetPowerSource()` function is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetPowerSource" )
```

#### Return value

A value of -1 if the function is not supported; 0 if the mobile phone is operating on battery power; 1 if the mobile phone is operating on an external power source.

## Network information

The commands in this section provide the mobile phone's network information to Flash content on the mobile phone.

### GetMaxSignalLevel()

The `GetMaxSignalLevel()` function returns the maximum signal strength level. It is a numeric value greater than zero.

This function is executed immediately upon invocation. If `GetMaxSignalLevel()` is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetMaxSignalLevel" )
```

#### Return value

A value of -1 if the function is not supported; otherwise, the maximum signal level, returned as a number.

### GetNetworkConnectStatus()

The `GetNetworkConnectStatus()` function returns a value indicating the current network connection status.

This function is executed immediately upon invocation. If `GetNetworkConnectStatus()` is not supported, a value of -1 is returned.

#### Syntax

```
status = FSCommand2( "GetNetworkConnectStatus" )
```

#### Return value

A value of -1 if the function is not supported; otherwise, one of the following values:

`0`: There is currently an active network connection.

`1`: The mobile phone is in the process of attempting to connect to the network.

2: There is currently no active network connection.

3: Network connection is in a suspended state.

4: The network connection is in an indeterminable state.

## GetNetworkName()

The GetNetworkName() function sets a parameter to the name of the current network. The parameter is passed in by name. The value returned is a string representing the network name.

If no network is registered, the parameter containing the name is set to a zero-length string, and a value of 0 is returned. If the network is registered but the name cannot be determined, the parameter containing the name is set to a zero-length string, and a value of 1 is returned. If the network is registered and its name can be determined, then the parameter containing the name is set to be the network name, and a value of 2 is returned.

This function is executed immediately upon invocation. If GetNetworkName() is not supported, a value of -1 is returned.

### Syntax

```
status = FSCommand2( “GetNetworkName”, “networkname” )
```

### Return value

A value of -1 if the function is not supported; otherwise, the following values are returned:

0: No network registered, and networkname is not set.

1: Network registered, but network name is not known; networkname is not set.

2: Network registered, and network name is known; networkname is set.

Sample resultant values for networkname:

AT&T Wireless: Phone is currently on the AT&T Wireless network.

KPN Mobile: Phone is currently on the KPN Mobile network.

## GetNetworkRequestStatus()

The GetNetworkRequestStatus() function returns a value indicating the status of the most recent HTTP request.

This function is executed immediately upon invocation. If GetNetworkRequestStatus() is not supported, a value of -1 is returned.

### Syntax

```
status = FSCommand2( "GetNetworkRequestStatus” )
```

### Return value

A value of -1 if the function is not supported; otherwise, one of the following values:

0: There is a pending request, a network connection has been established, the server’s host name has been resolved, and a connection to the server has been made.

1: There is a pending request, and a network connection is being established.

2: There is a pending request, but a network connection has not yet been established.

3: There is a pending request, a network connection has been established, and the server's host name is being resolved.

4: The request failed because of a network error.

5: The request failed because of a failure in connecting to the server.

6: The server returned an HTTP error (for example, 404).

7: The request failed because of a failure in accessing the DNS server or in resolving the server name.

8: The request has been successfully fulfilled.

9: The request failed because of a timeout.

10: The request has not yet been made.

## GetNetworkStatus()

The `GetNetworkStatus()` function returns a value indicating the network status of the phone (that is, whether there is a network registered and whether the phone is currently roaming).

This function is executed immediately upon invocation. If The `GetNetworkStatus()` is not supported, a value of `-1` is returned.

### Syntax

```
status = FSCommand2( "GetNetworkStatus" )
```

### Return value

A value of `-1` if the function is not supported; otherwise, one of the following values:

0: No network registered.

1: On home network.

2: On extended home network.

3: Roaming (away from home network).

## GetSignalLevel()

The `GetSignalLevel()` function returns the current signal strength. It is a numeric value, in the range of `0` to the maximum value returned by `GetMaxSignalLevel()`.

This function is executed immediately upon invocation. If `GetSignalLevel()` is not supported, a value of `-1` is returned.

### Syntax

```
status = FSCommand2( "GetSignalLevel" )
```

#### Return value

A value of `-1` if the function is not supported; otherwise, the current signal level, returned as a number.

## Device user settings

The commands in this section provide the mobile phone's language setting to Flash content on the mobile phone.

### GetLanguage()

The `GetLanguage()` function sets a parameter that identifies the language currently used by the mobile phone. The language is returned as a string in a variable that is passed in by name..

This function is executed immediately upon invocation. If `GetLanguage()` is not supported, a value of `-1` is returned.

#### Syntax

```
status = FSCommand2( "GetLanguage", “language”)
```

#### Return value

A value of `-1` if the function is not supported; `0` if it's supported.

Resultant values for the `language`:

`cs`: Czech.

`da`: Danish.

`de`: German.

`en-UK`: UK or international English.

`en`: USA English.

`es`: Spanish.

`fi`: Finnish.

`fr`: French.

`hu`: Hungarian.

`it`: Italian.

`jp`: Japanese.

`ko`: Korean.

`nl`: Dutch.

`no`: Norwegian.

`pl`: Polish.

`pt`: Portuguese.

`ru`: Russian.

`sv`: Swedish.

`tr`: Turkish.

`xu`: The language cannot be determined.

`zh-CN`: Simplified Chinese.

`zh-TW`: Traditional Chinese.

## Device and player identification

The commands in this section provide the mobile phone's ID and platform information to Flash content on the mobile phone.

### GetDeviceID()

The `GetDeviceID()` function sets a parameter that represents the unique identifier of the mobile phone (for example, serial number).

This function is executed immediately upon invocation. If `GetDeviceID()` is not supported, a value of `-1` is returned.

#### Syntax

```
status = FSCommand2( "GetDeviceID" “id” )
```

#### Return value

A value of -1 if the function is not supported; 0 if it's supported.

#### Supported applications

This feature is not supported in all mobile phones.

### GetPlatform()

The `GetPlatform()` function identifies the current platform. The platform broadly describes the class of the mobile phone. For mobile phones with open operating systems, this identifier is typically the name and version of the operating system.

The name of the platform is returned in a variable that is passed in by name, in the form of a string.

This function is executed immediately upon invocation. If `GetPlatform()` is not supported, a value of `-1` is returned.

#### Syntax

```
status = FSCommand2( "GetPlatform”, “platform” )
```

#### Return value

A value of `-1` if the function not supported; `0` if it's supported.

Sample resultant values for `platform`:

`506i` indicates that the device is a 506i phone.

`FOMA1` indicates that the device is a FOMA1 phone.

## GetDevice()

The `GetDevice()` function identifies the mobile phone on which Flash is running. This identifier is typically the model name.

The name of the mobile phone is returned in a variable that is passed in by name. The device identifier is a string.

This function is executed immediately upon invocation. If `GetDevice()` is not supported, a value of -1 is returned.

### Syntax

```
status = FSCommand2( "GetDevice", “device”)
```

### Return value

A value of `-1` if it's not supported; `0` if it's supported.

# CHAPTER 6
# Creating Content

This document contains numerous code examples and detailed reference information. With all of this information available to you for creating Macromedia Flash Lite 1.1 content for mobile phones, it's important to understand how to publish your Flash Lite content.

This chapter provides instructions for publishing your Flash Lite 1.1 content so that it plays back on mobile phones. It also describes how to embed sound into your Flash applications.

## Flash Lite 1.1 publish settings

When you open a new Flash document in Macromedia Flash MX Professional 2004, the Flash Player export settings are set to Flash Player 7 by default. When authoring content for mobile phones, you need to make sure the Flash export settings are set to Flash Lite 1.1 before you start creating any content.

There are three ways to adjust your Flash export settings for developing Flash content for mobile phones: you can manually change settings, create a publish profile, or use a publishing template.

### Manually change settings

Changing the Flash publish settings manually is straightforward but you must repeat these steps for every FLA file for which you want to export Flash Lite 1.1 content.

**To change Flash publish settings manually:**

1. Open a new document in Flash MX Professional 2004.
2. Open the Property inspector and click the Settings button. The Publish Settings dialog box appears.
3. In the Publish Settings dialog box, click the Player version pop-up menu. Select Flash Lite 1.1 and click OK.


The Flash Player export version is now set to Flash Lite 1.1. Now, whenever you test or export a SWF file it will be exported as a Flash Lite 1.1 SWF file.

### Creating a publish profile

Another way of reusing specific publish settings for multiple files and projects is to save them as a publishing profile. When you export your publish settings using a publishing profile, all of the selected options for all of the enabled tabs are saved.

**To create a publish profile:**

1. Open a new document in Flash MX Professional 2004, and save it.
2. Open the Publish Settings dialog box (File > Publish Settings), select the Flash tab, and click the Create New Profile button. Give your profile a descriptive name and click OK.
3. Still in the Publish Settings pop-up, make the changes to the export settings and then select Export from the Import/Export Profile button. Save your publish settings to the default location in the save window and click OK.

Your publish settings have been exported. You can import them into any new or existing project document and share them with others member of your team if you work in a group environment.

## Creating a simple movie for Flash Lite 1.1 (no sound)

You can create a Flash application (without sound) that runs on a mobile phone. Make sure you have installed the necessary updater files before you begin this procedure. See "Getting started" on page 7 for more information.

**To create a Flash Lite 1.1 compatible SWF file:**

1. In Flash MX Professional 2004, create a new document and name it FlashLiteTest.fla.
2. Select File > Publish Settings, and then the Flash tab. In the Version pop-up menu, select Flash Lite 1.1. Click OK.
3. From the Property inspector select the Size button, and change your document properties so that width = 240, height = 266, and Frame Rate = 15. Click OK. Make sure to use the appropriate frame rate on the actual devices.
4. Select Window > Other Panels > Common Libraries > Buttons. Select a button and drag it to the Stage.
5. If the button is not selected on the Stage, click it once to select it. Open the Actions panel if it is not already open (Window > Development Panels > Actions) and enter the following:

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

6. Select Control > Test Movie.

   Flash MX Professional 2004 executes the Test Movie command.

7. To simulate the user input of a mobile phone, you must disable the keyboard shortcuts (from the Test Movie window, select Control > Disable keyboard shortcuts). Use the Enter and Tab keys to interact with the SWF file.

   You can now interact with the Flash application. When you click the button in the SWF file, a browser opens at www.macromedia.com.

8. Select File > Publish to save the SWF file as FlashLiteTest.swf.

   In the mobile phone web browser or from a desktop that can transfer a file using desktop-to-phone synchronization software, transfer the file to the mobile phone and verify that it works correctly.

## Adding sound to your Flash Lite 1.1 application

You can add sound to your Flash Lite 1.1 application by associating a SMAF sound file with an ActionScript sound symbol so you can test your SWF file by using the Test Movie command in Flash MX Professional 2004 Flash.

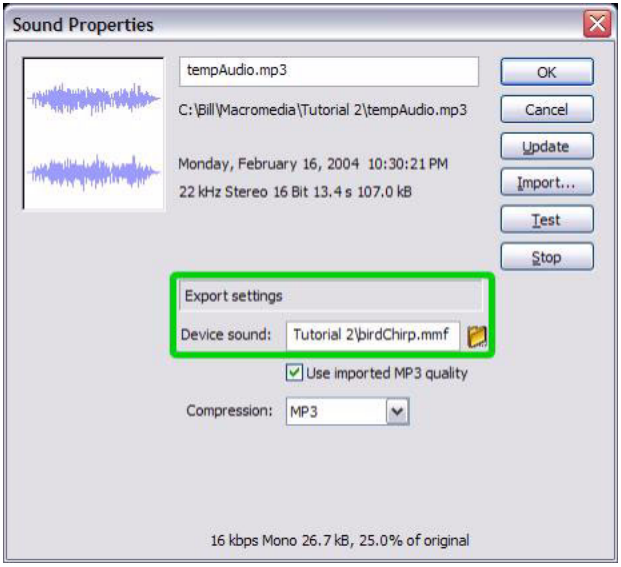
In the procedure below you will learn how to add sound to a Flash Lite application by embedding the SMAF sound file birdChirp.mmf, located in the CDK\Tutorials folder, into a Flash Lite 1.1 SWF file.

**To associate a SMAF sound file with an ActionScript sound symbol:**

1. In Flash MX Professional 2004, create a new document and name it FlashLiteSound.fla. Save it in the same directory as birdChirp.mmf (CDK\Tutorials folder).
2. Select File > Publish Settings and select the Flash tab. In the player version pop-up menu, select Flash Lite 1.1. Click OK.
3. From the Property inspector select the Size button, and change your document properties so that Width = 240, Height = 266, and Frame Rate = 15. Click OK. Make sure to use the appropriate frame rate on the actual devices.
4. Select File > Import > Import to Library. Locate the CDK\Tutorials folder and select the tempAudio.mp3 file. Click OK.
5. Select Window > Other Panels > Common Libraries > Buttons. Select a button and drag it to the Stage.
6. Double-click the new button. The Timeline changes to allow editing of the button and displays frames named Up, Over, Down, and Hit.
7. Select Insert > Timeline > Layer to create a new layer. Select Modify > Timeline > Layer Properties and change the name of the layer to Sound.
8. Select the Down frame in the Sound layer and insert a keyframe.
9. Select the testAudio.mp3 file from the Library window and drag it to the keyframe.

10. Associate the proxy sound with the birdChirp.mmf file by doing the following:

    - Select Window > Library and find the sound that you added in step 9. Select the sound and right-click it to open the context menu. Select Properties from the context menu. The Sound Properties dialog box appears:




    - In the Device sound text box, use the file browser to find and select birdChirp.mmf.
    - Click OK.

11. Select Control > Test Movie to start the Flash MX Professional 2004 Flash Lite 1.1 to test your SWF file.

12. Select File > Publish to save the SWF file as FlashLiteSound.swf.

When you add sound files to your Flash Lite SWF file, keep the following points in mind:

- The Effect, Sync, and Edit options are not supported on mobile devices.
- You must specify an external device sound for each sound in a document if you want the sound to play on a mobile device.
- As with all external files, the device sound files must be available during the Publish process. However, the file is not needed during the SWF playback.

In the mobile phone web browser or from a desktop that can transfer a file using desktop-to-phone synchronization software, transfer the file to the mobile phone and verify that it works correctly.

# CHAPTER 7
# Testing Content

Anyone can make mistakes while developing content, and it's a good idea to test your Macromedia Flash content frequently as you progress. Consider asking someone who is not familiar with your application to look at it and provide feedback.

## Testing considerations

Test your Macromedia Flash Lite 1.1 SWF content frequently on actual mobile phones. This step may seem obvious, but it is often overlooked. It is especially important when you develop Flash Lite 1.1 SWF files for mobile phones. No matter how much phone emulation you do, the final delivery remains the most important part of the development cycle. Emulation is helpful for much of the testing, but it is no substitute for testing on actual mobile phones.

You provide the most benefit for users by delivering a well-designed UI. If the application is slow, difficult to use, or not viewable, your development effort is wasted.

You should use the following methods to test the Flash Lite content you develop for mobile phones:

- The Flash Player Test Movie command (Control > Test Movie)
- Flash Player on the manufacturer's phone

The Macromedia Flash MX Professional 2004 Test Movie command recognizes and plays Flash Lite SWF files. When you select Control > Test Movie or Control > Test Scene, new information, warnings, and error messages specifically related to Flash Lite applications are displayed in a separate Output panel.

Whenever an unknown tag is encountered, warning messages are displayed so that you can modify the content appropriately. Not all invalid Flash content is flagged as an error (for example, invalid ActionScript and invalid key input).

For a detailed explanation of all messages related to Flash Lite 1.1, see Appendix C, "Warning and Error Messages" . This appendix lists all of the warning and error messages that you might see when creating Flash Lite 1.1 SWF files for mobile phones.

## Using the optional configuration file

The Flash Lite 1.1 External (Test Movie) player provides the user the ability to customize the features that are supported in the Flash Lite 1.1 player. The user can also add their platform specific strings in the configuration file. A sample configuration file is provided in the installation package.

**The steps to install the configuration file are:**

1. For Windows:

   Copy and paste DeviceMsg.cfg to the Flash MX 2004 configuration folder which usually is C:\Documents and Settings\<user name>\Local Settings\Application Data\Macromedia\Flash MX 2004\language\Configuration\

2. For Macintosh:

   Copy and paste DeviceMsg.cfg to Macintosh HD::Users:<user name>:Library:Application Support:Macromedia:Flash MX 2004:language:Configuration:

The configuration consists of multiple feature lines. Each line starts with a feature tag name followed by an equal sign. Lines starting with "//" are considered to be comments and are not processed.

There are two methods to set up a feature line. The first method is to add a double quoted string after the equal sign. When this string is present, that feature is turned on no matter what the original settings of the Test Movie player are. This string added by the user will be displayed in the authoring tool output window to remind the content developer of the platforms on which this feature is not supported. The second method is to put "on" or "off" after the equal sign to turn on/off that feature.

A special feature line can also be set up to change the sound order preference when creating content with compound sound data. One example is:

{Midi,SMAF_MA2,SMAF_MA3,SMAF_MA5,MFI,MFI_Fujitsu,MFI_Mitsubishi,MFI_NEC,MFI_Panasonic,MFI_Sharp,MFI_Sony}

In this example, Midi sound is set to have the highest priority when a compound sound bundle is processed. All the supported device sound formats are listed in this example. Once the user specifies this special feature line in the configuration file, playback of the device sound formats specified in this line will be supported in the test movie player. Sound formats not specified in this line will not be supported in the test movie player.

The details of tags that can be specified in the configuration file are explained in the following table:

| Tag name | Default Value(note) | Tag Usages | Notes |
|---|---|---|---|
| AddSound | On | Sound playback is allowed | |
| ADPCM | On | ADPCM sound format is allowed | |
| PCM | On | PCM sound format is allowed | |

| Tag name | Default Value(note) | Tag Usages | Notes |
|---|---|---|---|
| MIDI | On | MIDI sound format is allowed | |
| SMAF | On | All the SMAF sound format is allowed. capSMAF is set to 1; | This is a generic tag for all SMAF sound formats. |
| SMAF MA2 | On | MA2 SMAF sound format is allowed | This is set to “On” when the “SMAF” tag is set to “On” |
| SMAF MA3 | On | MA3 SMAF sound format is allowed | This is set to “On” when the “SMAF” tag is set to “On” |
| SMAF MA5 | On | MA5 SMAF sound format is allowed | This is set to “On” when the “SMAF” tag is set to “On” |
| MFI | On | Generic Mfi sound format sound is allowed | |
| MFI Fujitsu | On | Mfi sound with Fujitsu extension is allowed | |
| MFI Mitsubishi | On | Mfi sound with Mitsubishi extension is allowed | |
| MFI NEC | On | Mfi sound with NEC extension is allowed | |
| MFI Panasonic | On | Mfi sound with Panasonic extension is allowed | |
| MFI Sharp | On | Mfi sound with Sharp extension is allowed | |
| MFI Sony | On | Mfi sound with Sony extension is allowed | |
| mp3codec | On | MP3 sound format is allowed | |
| SingleSoundOnly | On | Only one device sound can be played at one time, no mixing allowed | |
| AddSoundKeyOnly | On | When turned on, sound will be played only when it is associated with a key press | |
| StreamingSound | On | Stream Sound is supported, _capStreamSound is set to 1. | |
| LoadVarsOnePerKey | Off | When turned on, only loadVars calls associated with a key press are allowed | |

| Tag name | Default Value(note) | Tag Usages | Notes |
|---|---|---|---|
| LoadVarsOnePerKeyOrFrame | On | loadVars call does not have to be associated with a key press, but only one call is allowed per key and per frame. | |
| LoadMovieOnePerKey | Off | When turned on, only loadMovie calls associated with a key press are allowed | |
| LoadMovieOnePerKeyOrFrame | On | loadMovie call does not have to be associated with a key press, but only one call is allowed per key and per frame. | |
| GetURLOnePerKey | Off | When turned on, only getUrl calls associated with a key press are allowed | |
| GetURLOnePerKeyOrFrame | On | GetUrl call does not have to be associated with a key press, but only one call is allowed per key and per frame. | |
| FSCommandOnePerKey | Off | When turned on, only FSCommand calls associated with a key press are allowed | |
| FSCommandOnePerKeyOrFrame | On | FSCommand call does not have to be associated with a key press, but only one call is allowed per key and per frame. | |
| KeySetFull | Off | When turned on, all the key events will be handled. | |
| KeySetPhone | On | Only the keys used on cell phones ('0'-'9'',*,#) are processed. | |
| Inputtext | On | Input text is allowed | |
| Mouse | Off | Extra Mouse events are handled | |
| Navi4Way | On | Four way navigation mode is on, _cap4WayKeyAS is set to 1. | |
| Navi4WayWrapAround | Off | Four way navigation with wrap around mode is on | Only valid if Navi4Way feature is turned on. |

| Tag name | Default Value(note) | Tag Usages | Notes |
|---|---|---|---|
| Email | Off | When turned on, _capEmail is set to 1; otherwise, it is set to 0. | |
| SMS | On | When turned on, _capSMS is set to 1; otherwise, it is set to 0. | |
| MMS | Off | When turned on, _capMMSis set to 1; otherwise, it is set to 0. | |
| LoadData | Off | When turned on, _capLoadData is set to 1; otherwise, it is set to 0. | |

***Note:*** The default value in the table is the value that the test movie player uses for the feature when there is no corresponding feature line specified in the configuration file.

# CHAPTER 8
# Development Kit Examples

The development kit includes a variety of sample files (FLA and SWF files) that demonstrate many of the concepts and applications that are described in this document. These examples are included to help you create content for mobile phones. The files include capabilities examples, processor detectors, and data-driven examples. Be sure to view the readme.txt file in the folder associated with each sample file.

# CHAPTER 9
# Resources and Support

As you develop Flash content for mobile phones, it's important to use all of the resources available throughout the community. Websites, books, tutorials, articles, and discussion groups are great ways to enhance and share your knowledge with others.

## Let us know about your application

If you have created a Flash Lite 1.1 application for a mobile phone, Macromedia would like to hear more about it. Send e-mail to mobile-applications@macromedia.com.

## Web resources

For more information about Macromedia Flash Lite 1.1 for mobile phones, visit the following sites:

- Macromedia Mobile and Devices Developer Center
  www.macromedia.com/devnet/devices/
- Flash Devices—Flash Development Resource for Mobile Devices
  www.flashdevices.net
- Flash the Future—Developer site for Flash on devices
  www.flashthefuture.com
- miniml—Pixel fonts for use with Flash on small screens
  www.miniml.com
- Fonts For Flash—Pixel fonts for use with Flash on small screens
  www.fontsforflash.com
- Ultra Fonts—Grayscale-enabled outline pixel fonts for use with Flash on small screens.
  www.ultrafonts.com

## Books

There are many books about Flash, but currently only two specifically address the development of Flash applications for mobile devices. Both of these books offer insight into real-world scenarios and complement each other well.

*Flash Enabled: Flash Design & Development for Devices*
by Phillip Torrone, Branden Hall, Bill Perry, et al.
New Riders Publishing
ISBN: 0735711771

*Flash: The Future*
by Jon Warren Lentz, Ian Chia, Bill Turner, et al.
No Starch Press
ISBN: 1886411964

## Discussion groups

- Macromedia Flash Support Forums – Flash Handhelds
  webforums.macromedia.com/flash/categories.cfm?catid=195

# APPENDIX A
# Supported ActionScript

This appendix lists the Macromedia Flash Lite 1.1 ActionScript commands.

| Action name | Description | Support |
|---|---|---|
| // (comment) | Comment; indicates the beginning of a script comment. Any characters that appear between the comment delimiter // and the end-of-line character are interpreted as a comment. | Fully supported |
| , (comma) | Operator; a separator between two expressions that causes the value of the second expression to be the return value. | Fully supported |
| . (dot) | Operator; used to navigate movie clip hierarchies to access nested (child) movie clips, variables, or properties. | Fully supported |
| " " (string delimiter) | String delimiter; when used before and after characters, double quotes indicate that the characters have a literal value and are considered a string and not a variable, numerical value, or other ActionScript element. | Fully supported |
| -- (decrement) | Operator; a pre-decrement and post-decrement unary operator that subtracts 1 from an expression. | Fully supported |
| ++ (increment) | Operator; a pre-increment and post-increment unary operator that adds 1 to an expression. | Fully supported |
| + (add) | A numeric operator used for adding numbers. | Fully supported |
| += (addition assignment) | Operator (arithmetic); assigns to `expression1` the value of `expression1 + expression2`<br>For example, the following two statements have the same result:<br>`x += y;`<br>`x = x + y;` | Fully supported |

| Action name | Description | Support |
|---|---|---|
| `(-)` subtract | Operator (arithmetic); used for negating or subtracting. When used for negating, it reverses the sign of the numerical expression. When used for subtracting, it performs an arithmetic subtraction on two numerical expressions, subtracting `expression2` from `expression1`.<br>Example 1: The following statement reverses the sign of the expression 2 + 3.<br>-(2 + 3)<br>The result is -5.<br>Example 2: The following statement subtracts the integer 2 from the integer 5.<br>5 - 2<br>The result is 3. | Fully supported |
| `-=` (subtraction assignment) | Operator (arithmetic); assigns to `expression1` the value of `expression1 - expression2`<br>For example, the following two statements have the same result:<br>`x -= y;`<br>`x = x - y;` | Fully supported |
| `*` (multiply) | Operator (arithmetic); multiplies two numerical expressions. | Fully supported |
| `*=` (multiplication assignment) | Operator (arithmetic); assigns to `expression1` the value of `expression1 * expression2`<br>`x *= y;`<br>`x = x * y;` | Fully supported |
| `/` (divide) | Operator (arithmetic); divides `expression1` by `expression2`.<br>For example, the following statement sets the value of x to 25:<br>`y = 50;`<br>`x = y/2;` | Fully supported |
| `/=` (division assignment) | Operator (arithmetic); assigns to `expression1` the value of `expression1 / expression2`<br>For example, the following two statements are the same:<br>`x /= y;`<br>`x = x / y;` | Fully supported |
| = (numeric equality) | A numeric equality operator used to test two expressions for equality. The result is `true` if the expressions are equal. | Fully supported |

| Action name | Description | Support |
|---|---|---|
| < (less than) | Operator (comparison); compares two expressions and determines whether `expression1` is less than `expression2` (`true`), or whether `expression1` is greater than or equal to `expression2` (`false`). In Flash Lite (and Flash 4), < is a numeric operator and is only used for expressions and not strings.<br>The following examples illustrate `true` and `false` results for < comparisons.<br>`3 < 10;`<br>`// true`<br><br>`10 < 3;`<br>`// false` | Fully supported |
| <= (less than or equal to) | Operator (comparison); compares two expressions and determines whether `expression1` is less than or equal to `expression2` (`true`), or whether `expression1` is greater than `expression2` (`false`).<br>The following examples illustrate `true` and `false` results for <= comparisons:<br>`5 <= 10;`<br>`// true`<br><br>`2 <= 2;`<br>`// true`<br><br>`10 <= 3;`<br>`// false` | Fully supported |
| > (greater than) | Operator (comparison); compares two expressions and determines whether `expression1` is greater than `expression2` (`true`), or whether `expression1` is less than or equal to `expression2` (`false`).<br>The following examples illustrate `true` and `false` results for > comparisons.<br>`10 > 3;`<br>`// true`<br><br>`3 > 10;`<br>`// false` | Fully supported |
| >= (greater than or equal to) | Operator (comparison); compares two expressions and determines whether `expression1` is greater than or equal to `expression2` (`true`), or whether `expression1` is less than `expression2` (`false`).<br>The following examples illustrate `true` and `false` results for >= comparisons:<br>`10 >= 5;`<br>`// true`<br><br>`2 >= 2;`<br>`// true`<br><br>`3 >=10;`<br>`// false` | Fully supported |

| Action name | Description | Support |
|---|---|---|
| <> (inequality) | Operator (equality); tests the opposite of the equality operator. If expression1 is equal to expression2, the result is false.<br>The following examples illustrate true and false returns for the <> operator.<br>`3 < > 10;`<br>`// true`<br><br>`3 <> 3;`<br>`// false` | Fully supported |
| % (modulo) | Operator; calculates the remainder of expression1 divided by expression2.<br>For example, the following statement sets the value of x to 3:<br>`x = 45 % 6;` | Fully supported |
| %= (modulo assignment) | Operator (assignment); assigns to expression1 the value of expression1 % expression2.<br>For example, the following two expressions are the same:<br><br>`x %= y`<br>`x = x % y` | Fully supported |
| \|\| (logical OR) | Operator (logical); evaluates expression1 and, if expression1 is false, evaluates expression2. The result is true if either or both expressions evaluate to true; the result is false only if both expressions evaluate to false.<br>The following example uses the \|\| operator in an if statement. The second expression evaluates to true so the final result is true:<br>`x = 10;`<br>`y = 250;`<br>`if (x > 25 \|\| y > 200) {`<br>`  z = 5;`<br>`  }`<br>`  else {`<br>`  z=0;`<br>`}`<br>`// z has a value of 5 after the code above has executed.` | Fully supported |
| ! (logical NOT) | Operator (logical); inverts the Boolean value of a variable or expression. | Fully supported |

| Action name | Description | Support |
|---|---|---|
| && (logical AND) | Operator (logical); evaluates expression1 and, if expression1 is true, evaluates expression2. The result is true if both expressions evaluate to true; the result is false if either expression evaluates to false.<br>The following example uses the && operator in an if statement. Both expressions evaluate to true, so the final result is true:<br>`x = 30;`<br>`y = 250;`<br>`if ( x > 25 && y > 200) {`<br>`z = 5;`<br>`}`<br>`else {`<br>`z = 0;`<br>`}`<br>`// z has a value of 5 after the code above has executed.` | Fully supported |
| ?: (conditional) | Operator (conditional); evaluates expression1, and returns the value of expression2 if expression1 is true; otherwise, returns the value of expression3.<br>The following statement assigns the value of variable x to variable z because expression1 evaluates to true:<br>`x = 5;`<br>`y = 10;`<br>`z = (x < 6) ? x : y;`<br>`// z has a value of 5` | Fully supported |
| & (string concatenation) | Operator; used for concatenating strings. | Fully supported |
| add | Operator; concatenates (combines) two or more strings. | Fully supported |
| and | Operator; performs a logical AND operation. If both expressions evaluate to true, then the entire expression is true. | Fully supported |
| break | Action; appears within a loop (for, for...in, do...while or while). The break action skips the rest of the loop body, stopping the looping action, and executes the statement following the loop statement. Use the break action to break out of a series of nested loops. | Fully supported |
| call | Action; switches the context from the current script to the script attached to the frame being called. | Fully supported |
| case | Keyword; defines a condition for the switch action. | Fully supported |
| chr() | String function; converts ASCII code numbers to characters. | Fully supported |
| continue | Action; used to control code execution in loops. | Fully supported |
| do... while | Action; executes the statements inside the loop, and then evaluates the condition of the loop for as long as the condition is true. | Fully supported |
| duplicateMovieClip | Action; creates an instance of a movie clip while the movie is playing. | Fully supported |

| Action name | Description | Support |
|---|---|---|
| `else` | Action; specifies the actions, clauses, arguments, or other conditional to run if the initial `if` statement returns `false`. | Fully supported |
| `else if` | Action; evaluates a condition and specifies the statements to run if the condition in the initial `if` statement returns `false`. | Fully supported |
| `eq` (string equal) | Comparison operator; compares two expressions for equality and returns `true` if `expression1` is equal to `expression2`; otherwise, returns `false`. This action is string specific.<br>The following examples illustrate `true` and `false` results for the `eq` operator:<br>`x ="Amy";`<br>`y="Fred";`<br>`x eq "Amy";`<br>`// true`<br>`x eq y;`<br>`// false` | Fully supported |
| `eval()` | Function; accesses variables. The value of the variable is returned. | Fully supported |
| `fscommand()` | Action; allows the Flash application to communicate with the program hosting Flash Player. | Partially supported |
| `ge` (string greater than or equal) | Comparison operator; returns `true` if the string representation for `expression1` is greater than or equal to the string representation for `expression2`; otherwise, returns `false`. This action is string specific.<br>The following examples illustrate `true` and `false` results for the `ge` operator:<br>`x ="Amy";`<br>`y="Fred";`<br>`x ge y;`<br>`// false`<br>`x ge "Amy";`<br>`// true`<br>`y ge x;`<br>`// true` | Fully supported |
| `getProperty()` | Function; returns the value of the specified property for the movie clip instance. | Partially supported. (See "Supported ActionScript" on page 61.) |
| `getTimer()` | Function; returns the number of milliseconds that have elapsed since the SWF file started playing. | Fully supported |

| Action name | Description | Support |
|---|---|---|
| `getURL()` | Action; loads a document from a specific URL into a window or passes variables to another application at a defined URL. When sending variables, specify whether to load variables using a `GET` or `POST` method. `GET` appends the variables to the end of the URL, and is used for small numbers of variables. `POST` sends the variables in a separate HTTP header and is used for long strings of variables. | Partially supported (The URL protocols http, https, mailto, and tel are supported, once per event action.) |
| `gotoAndPlay()` | Action; sends the playhead to the specified frame in a scene and plays from that frame. If a scene is not specified, the playhead goes to the specified frame in the current scene. | Fully supported |
| `gotoAndStop()` | Action; sends the playhead to the specified frame in a scene and stops it. If no scene is specified, the playhead is sent to the frame in the current scene. | Fully supported |
| `gt` (string greater than) | Comparison operator; returns `true` if the string representation for `expression1` is greater than the string representation for `expression2`; otherwise, returns `false`. This action is string specific. | Fully supported |
| `if` | Action; evaluates a condition to determine the next action in a movie. If the condition is `true`, Flash runs the statements that follow. | Fully supported |
| `ifFrameLoaded()` | Action; checks whether the contents of a specific frame are available locally. Use `ifFrameLoaded()` to start playing a simple animation while the rest of the SWF file downloads. | Fully supported |
| `int()` | Function; converts a decimal number to the closest integer value. | Fully supported |
| `le` (string less than or equal) | Comparison operator; returns true if the string representation for `expression1` is less than or equal to the string representation for `expression2`; otherwise, returns `false`. This action is string specific.<br>The following examples illustrate `true` and `false` results for the `le` operator:<br>`x =”Amy”;`<br>`y=”Fred”;`<br>`y le x;`<br>`// false`<br>`x le “Amy”;`<br>`// true`<br>`x le y;`<br>`// true` | Fully supported |
| `length()` | String function; returns the length of the specified string or variable name. | Fully supported |

| Action name | Description | Support |
|---|---|---|
| `loadMovie()` | Action; plays additional movies without closing Flash Lite. Normally, Flash Lite displays a single Flash application (SWF file) and then closes. The `loadMovie()` action lets you display several SWF files at once or switch between them without loading another HTML document. | Fully supported |
| `loadMovieNum()` | Action; loads a SWF file into a level in Flash Lite while the originally loaded movie is playing. | Fully supported |
| `loadVariables()` | Action; reads data from an external file, such as a text file or text generated by a CGI script, Active Server Pages (ASP), or Personal Home Page (PHP), and sets the values for variables in a SWF file or movie clip. | Fully supported |
| `loadVariablesNum()` | Action; reads data from an external file, such as a text file or text generated by a CGI script, Active Server Pages (ASP), or PHP, or Perl script, and sets the values for variables in a Flash Lite level. | Fully supported |
| `lt` (string less than) | Operator (comparison); compares `expression1` to `expression2` and returns `true` if `expression1` is less than or equal to `expression2`; otherwise, returns `false`. This action is string specific.<br>The following examples illustrate `true` and `false` results for the `lt` operator:<br>`x ="Amy";`<br>`y="Fred";`<br>`y lt x;`<br>`// false`<br>`x lt "Jane";`<br>`// true` | Fully supported |
| `mbchr()` | String function; converts an ASCII code number to a multibyte character. | Fully supported |
| `mblength()` | String function; returns the length of the multibyte character string. | Fully supported |
| `mbord()` | String function; converts the specified character to a multibyte number. | Fully supported |
| `mbsubstring()` | String function; extracts a new multibyte character string from a multibyte character string. | Fully supported |
| `ne` (string not-equal) | Comparison operator; compares two expressions for inequality and returns `true` if `expression1` is not equal to `expression2`; otherwise, returns `false`. This action is string specific.<br>The following examples illustrate `true` and `false` results for the `ne` operator:<br>`x ="Amy";`<br>`y="Fred";`<br>`y ne"Amy";`<br>`// true`<br>`x ne "Amy";`<br>`// false` | Fully supported |

| Action name | Description | Support |
|---|---|---|
| `nextFrame()` | Action; sends the playhead to the next frame and stops it. | Fully supported |
| `nextScene()` | Action; sends the playhead to frame 1 of the next scene and stops it. | Fully supported |
| `Number()` | Function; converts the argument `x` to a number and returns a value as follows:<br>If `x` is a number, the return value is `x`.<br>If `x` is a Boolean value, the return value is 1 if `x` is `true`, 0 if `x` is `false`.<br>If the value of `x` is a string, the function attempts to parse `x` as a decimal number with an optional trailing exponent, that is, 1.57505e-3.<br>If `x` is undefined, the return value is 0. | Not supported |
| `on` (event) | Handler; specifies the mouse event or keypress that triggers an action. | Partially supported (Events supported are `keyPress`, `press`, `release`, `rollOver` and `rollOut`. Keys supported in Flash Lite are: 0-9, *, # and Select.) |
| `ord()` | String function; converts characters to ASCII code numbers. | Fully supported |
| `play()` | Action; moves the playhead forward in the Timeline. | Fully supported |
| `prevFrame()` | Action; sends the playhead to the previous frame and stops it. | Fully supported |
| `prevScene()` | Action; sends the playhead to Frame 1 of the previous scene and stops it. | Fully supported |
| `random()` | Function; returns a random integer between 0 and the integer specified in the `value` parameter. | Fully supported |
| `removeMovieClip()` | Action; deletes a movie clip instance that was created with the `duplicateMovieClip()` action. | Fully supported |
| `set()` | Action; assigns a value to a variable. A variable is a container that holds information. | Fully supported |
| `setProperty()` | Action; changes the property of a movie clip as the SWF file plays. | Partially supported. (See "Supported ActionScript" on page 61.) |
| `startDrag()` | Action; makes the target movie clip draggable while the SWF file is playing. Only one movie clip can be dragged at a time. | Not supported |

| Action name | Description | Support |
|---|---|---|
| `stop()` | Action; stops the SWF file that is currently playing. | Fully supported |
| `stopAllSounds()` | Action; stops all sounds currently playing in a movie without stopping the playhead. | Fully supported |
| `stopDrag()` | Action; stops the current drag operation. | Not supported |
| `String()` | Function; returns a string representation of the specified argument as follows:<br>If `x` is a Boolean value, the return string is `true` or `false`.<br>If `x` is a number, the return string is a decimal representation of the number.<br>If `x` is a string, the return string is `x`.<br>If *x* is a movie clip, the return value is the target path of the movie clip in slash (/) notation.<br>If `x` is undefined, the return value is an empty string. | Not supported |
| `substring()` | String function; extracts part of a string. | Fully supported |
| `switch()` | Action; creates a branching structure for ActionScript statements. The switch action tests a condition and executes statements if the condition returns a value of `true`. | Fully supported |
| `tellTarget()` | Action; Can be used to apply instructions to a particular Timeline or movie clip. For example, `tellTarget()` can be assigned to buttons that stop or start movie clips on the Stage or prompt movie clips to jump to a particular frame. | Fully supported |
| `toggleHighQuality()` | Action; turns anti-aliasing on and off in Flash Lite. Anti-aliasing smooths the edges of objects but results in slower movie playback. The `toggleHighQuality()` action affects all movies in Flash Lite. | Fully supported |
| `trace()` | Action; evaluates the expression and displays the results in the Output panel when you run the Test Movie command. | Fully supported |
| `unloadMovie()` | Action; removes a movie clip from Flash Lite that was previously loaded or created using the `loadMovie()` or `duplicateMovieClip()` actions. | Fully supported |
| `unloadMovieNum()` | Action; removes a movie at a specified level from Flash Lite that was previously loaded or created using the `loadMovie()` action. | Fully supported |
| `while()` | Action; runs a statement or series of statements repeatedly in a loop as long as the condition argument is `true`. | Fully supported |

# APPENDIX B
# Supported ActionScript Properties

This appendix lists the Macromedia Flash Lite 1.1 ActionScript properties and points out any exceptions.

| Properties | Description | Support |
|---|---|---|
| `/`(slash notation) | Property; specifies or returns a reference to the root SWF file Timeline. Functionality provided by this property is similar to that provided by the `_root` property in Flash 5. | Fully supported |
| : | Used in conjunction with "/" to reference variables and properties of other movie clips that are contained in the current SWF file. It is also used with the `Call()` action to reference a frame label of a movie clip. | Fully supported |
| `_alpha` | Property; sets or retrieves the alpha transparency (*value*) of the movie clip. Valid values are 0 (fully transparent) to 100 (fully opaque). | Fully supported |
| `_currentframe` | Property (read-only); returns the number of the frame where the playhead is currently located in the Timeline. | Fully supported |
| `_droptarget` | Property (read-only); returns the absolute path in slash syntax notation of the movie clip instance on which the *`draggableInstanceName`* (the name of a movie clip instance that was the target of a `startDrag()` action) was dropped. This property always returns a path that starts with /. | Not supported |
| `_focusrect` | Property (global); specifies whether a yellow rectangle appears around the button that has the current focus. The default value true (nonzero) displays a yellow rectangle around the currently focused button or text field as the user presses the Tab key to navigate. | Fully supported |

| Properties | Description | Support |
|---|---|---|
| _framesloaded | Property (read-only); the number of frames that have been loaded from a streaming movie. This property is useful for determining whether the contents of a specific frame, and all the frames before it, have loaded and are available locally in a user's browser. | Fully supported |
| _height | Property (read-only); retrieves the height of the space occupied by a movie's content.<br>In Flash Lite, _height is a read-only property. | Fully supported |
| _highquality | Property (global); specifies the level of anti-aliasing applied to the current movie. This property can be used to control bitmap smoothing as well. | Partially supported (bitmap smoothing not supported) |
| _level | In Flash Lite, SWF files are assigned a number according to the order in which they are loaded. The SWF file that is loaded first is loaded at the bottom level, level 0. The SWF file in level 0 sets the frame rate, background color, and frame size for all subsequently loaded SWF files. SWF files are then stacked in higher-numbered levels above the SWF file in level 0.<br>This property is a reference to the root movie clip Timeline of *levelN*. | Fully supported |
| Maxscroll | Property; a read-only property that works with the Scroll property to control the display of information in a text field. This property can be retrieved, but not modified. | Fully supported |
| _name | Property; specifies the movie clip instance name. | Fully supported |
| _rotation | Property; specifies the rotation, in degrees, of the movie clip. | Fully supported |
| Scroll | Controls the display of information in a text field associated with a variable. The Scroll property defines where the text field begins displaying content. After you set it, Flash Lite updates it as the user scrolls through the text field. The Scroll property is useful for directing users to a specific paragraph in a long passage, or creating scrolling text fields. | Fully supported |
| _soundbuftime | Property (global); establishes the number of seconds it takes to stream sound to the prebuffer. | Not supported |
| _target | Property (read-only); returns the target path of the movie clip instance specified as argument. | Fully supported |
| _totalframes | Property (read-only); evaluates the movie clip specified as argument and returns the total number of frames in the SWF file. | Fully supported |
| _url | Property (read only); retrieves the URL of the SWF file from which the movie clip was downloaded. | Not supported |

| Properties | Description | Support |
|---|---|---|
| `_visible` | Property; determines whether the specified movie clip is visible. Movie clips that are not visible (when the property is set to `false`) are disabled. | Fully supported |
| `_width` | Property (read-only); retrieves the width of the space occupied by a movie's content. In Flash Lite, `_width` is a read- only property. | Fully supported |
| `_x` | Property; sets the *x* coordinate of the movie clip relative to the local coordinates of the parent movie clip. | Fully supported |
| `_xscale` | Property; determines the horizontal scale (percentage) of the movie clip as applied from the registration point of the movie clip. | Fully supported |
| `_y` | Property; sets the *y* coordinate of the movie clip relative to the local coordinates of the parent movie clip. | Fully supported |
| `_yscale` | Property; sets the vertical scale (percentage) of the movie clip as applied from the registration point of the movie clip. | Fully supported |

# APPENDIX C
# Warning and Error Messages

This appendix lists the possible information and warning messages you might encounter when creating Macromedia Flash Lite 1.1 content for mobile phones.

## Flash authoring tool warning and error messages

| Message Identifier | Message | Explanation |
|---|---|---|
| SWFS016 | Detected `loadMovie()` - will be ignored. | Flash Player detected that the SWF file contains a `loadMovie()` ActionScript command, which the specified device's Flash Lite does not support. No modifications are made to the device-specific SWF file–this is just a warning. |
| SWFS017 | Detected `loadVariables()` – will be ignored. | Flash Player detected that the SWF file contains a `loadVariables()` ActionScript command, which the specified device's Flash Lite does not support. No modifications are made to the device-specific SWF file–this is just a warning. |
| SWFS018 | Detected `getURL()` - restrictions may apply. | Flash Player detected that the SWF file contains a `getURL()` ActionScript command, which has some runtime restrictions when played by the specified device's Flash Lite. No modifications are made to the device-specific SWF file–this is just a warning. |
| SWFS019 | `startDrag()` action not supported. | Flash Player detected that the SWF file contains a `startDrag()` ActionScript command, which Flash Lite does not support. No modifications are made to the device- specific SWF file–this is just a warning. |
| SWFS020 | `stopDrag()` action not supported. | Flash Player detected that the SWF file contains a `stopDrag()` ActionScript command, which Flash Lite does not support. No modifications are made to the device-specific SWF file–this is just a warning. |

| Message Identifier | Message | Explanation |
|---|---|---|
| SWFS021 | `_droptarget` property not supported. | Flash Player detected that the SWF file contains a `getProperty()` or `setProperty()` ActionScript command referring to the `droptarget` property, which Flash Lite does not support. No modifications are made to the device-specific SWF file–this is just a warning. |
| SWFS023 | `_soundbuftime` property not supported. | Flash Player detected that the SWF file contains a `getProperty()` or `setProperty()` ActionScript command referring to the `_soundbuftime` property, which Flash Lite does not support. No modifications are made to the device-specific SWF file–this is just a warning. |
| SWFS027 | File saved as *filename* | Flash Player displays this message to indicate the name it is using for the device-specific SWF file. |
| SWFS028 | File size after substitution: *nnn* kilobytes | Flash Player displays this message to indicate the size of the device-specific SWF file after substitution or removal of sounds. This is an informational message only. |
| SWFS032 | Detected `fscommand()` - will be ignored. | The Flash player detected that the SWF file contains an `fscomamnd()` ActionScript command, which Flash Lite does not support. No modifications are made to the device-specific SWF file–this is just a warning. |
| SWFS033 | Not enough memory to perform operation. | Flash Player was unable to get enough memory to finish the operation. |
| SWFS035 | `_url` property not supported. | Flash Player detected that the SWF file contains a `getProperty()` or `setProperty()` Actionscript command referring to the `_url` property, which is not supported by Flash Lite. No modifications are made to the device specific SWF file–this is just a warning |
| SWFS040 | Uncompressed sound found. | Flash Player detected that the SWF file contains uncompressed sound. which is not supported by the specified device's Flash Player. No modifications are made to the device-specific SWF file–this is just a warning. |
| SWFS041 | ADPCM sound found. | Flash Player detected that the SWF file contains ADPCM sound, which is not supported by the specified device's Flash Player. No modifications are made to the device-specific SWF file–this is just a warning. |
| SWFS042 | Nellymoser sound found. | Flash Player detected that the SWF file contains Nellymoser sound, which is not supported by the specified device's Flash Player. No modifications are made to the device-specific SWF file–this is just a warning. |

| Message Identifier | Message | Explanation |
|---|---|---|
| SWFS043 | MP3 sound found. | Flash Player detected that the SWF file contains MP3 sound, which is not supported by the specified device's Flash Player. No modifications are made to the device-specific SWF file—this is just a warning. |
| SWFS044 | Export tag *subst:sound file name* was found and ignored. Please use the Device sound feature. | Flash Player detected that the SWF file contains a `subst:file name` export tag used in the old Flash 6 updater, which is not supported by the Flash Lite 1.0 Test Movie command. The author should use the new Device Sound feature. No modifications are made to the device-specific SWF file—this is just a warning. |
| SWFS045 | MIDI sound found. | Flash Player detected that the SWF file contains MIDI sound, which is supported by Flash Lite. |
| SWFS046 | MFi sound with *manufacturer* extension found. | Flash Player detected that the SWF file contains MFi sound with certain manufacturer extension, which is supported by Flash Lite. |
| SWFS047 | Unsupported device sound format found. | Flash Player detected that the SWF file contains a sound format that is not supported by Flash Lite. No modifications are made to the device-specific SWF file—this is just a warning. |
| SWFS048 | Sound bundle found | Flash Player detected that the SWF file contains a sound bundle while parsing the movie. The playback bundled sound on the real device might be different. |
| SWFS049 | SMAF sound found. | Flash Player detected that the SWF file contains SMAF sound, which is supported in certain configurations. |
| FTPE001 | The key will not be processed: *key* keycode: *nnn* | While testing a movie clip, a key was pressed that Flash Lite does not support—the keypress is ignored. |
| FTPA005 | The call to `getURL()` for *URL* was ignored because there was more than one request per keypress. | While testing a movie clip, multiple ActionScript `getURL()` commands were called during a keypress event. Flash Lite allows only one `getURL()` command per keypress, so only the first command is processed—the others are ignored. |
| FTPA007 | `getProperty` or `setProperty` not supported for: *property name* | While testing a movie clip, a `getProperty()` or `setProperty()` ActionScript command was encountered for a property that the specified device's Flash Player does not support. The command is ignored. |
| FTPA008 | `getProperty` or `setProperty` not fully supported for: *property name* | While testing a movie clip, a `getProperty()` or `setProperty()` ActionScript command was encountered for a property that Flash Lite does not completely support. The command is performed, but the results might not be as expected. |

| Message Identifier | Message | Explanation |
|---|---|---|
| FTPA009 | `startDrag()` and `stopDrag()` are not supported. | While testing a movie clip, a `startDrag()` or `stopDrag()` ActionScript command was encountered. Flash Lite does not support these commands and ignores them. |
| FTPS011 | Only a single sound can be played at a time (no mixing). | While testing a movie clip, a sound was started while another sound was already playing. Flash Lite does not support sound mixing, so the first sound is stopped to allow the second sound to play. |
| FTPS022 | ADPCM sounds not supported. | While testing a movie clip, an ADPCM sound was encountered. The specified device's Flash Player does not support ADPCM sound format. |
| FTPS023 | MP3 sounds not supported. | While testing the movie clip, an MP3 sound was encountered. The specified device's Flash Player does not support MP3 sound format. |
| FTPS024 | MIDI/MFI sounds not supported. | While testing the movie, MIDI/MFI sound was encountered. The specified device's Flash Player does not support MIDI/MFI sound format. |
| FTPS025 | PCM sounds not supported. | While testing the movie clip, an PCM sound was encountered. The specified device's Flash Player does not support PCM sound format. |
| FTPS026 | Debug movie is not supported in the specified test movie player | While the Flash Player is specified in the publish settings, an attempt was made to debug the movie using the Flash Lite 1.0 Test Movie command, which is not supported. |
| FTPS027 | Compound sound found. | Flash Player detected that the SWF file contains compound sound. The playback of compound sound on the real device might be different. |
| FTPS028 | Invalid FSCommand2 command found. | Flash Player detected an invalid `FSCommand2() object` ActionScript command. |
| FTPS029 | FSCommand2 *command name* command found. | Flash Player detected a valid `FSCommand2()` ActionScript command. |
| FTPS030 | FSCommand2 *command name* command not supported in the emulator, please test it on the device. | Flash Player detected an `FSCommand2()` ActionScript command that is not supported in the emulator. The user is advised to test it on the device. |
| FTPS031 | More than one instance of URL request calls found. | Flash Player detected more than one instance of URL request (`getUrl()`, `loadMovie()`, `loadVars()` and `fsCommand()`) calls. Only one URL request per frame or event handler is allowed. |

| Message Identifier | Message | Explanation |
|---|---|---|
| FTPS032 | A call to `getURL` *URL* found, limitation might apply. | Flash Player detected a `getURL`() call. Different limitations might apply on different devices. |
| FTPS033 | A call to `loadVariables` *URL* found, limitation might apply. | Flash Player detected a `loadVariables()` call. Different limitations might apply on different devices. |
| FTPS034 | A call to `FSCommmand` *URL* found, limitation might apply. | Flash Player detected an `FSCommmand()` call. Different limitations might apply on different devices. |
| FTPS035 | A call to `loadMovie` *URL* found, limitation might apply. | Flash Player detected a `loadMovie()` call. Different limitations might apply on different devices. |
| FTPS036 | *size* kilobytes of *file name with extension* sound found in compound sound. | Flash Player detected a device sound in the compound sound, the format and the size of the device sound are reported. |
| FTPS037 | SMAF sounds not supported. | Flash Player detected that the SWF file contains SMAF sound, which is not supported by the specified device's Flash Player. No modifications are made to the device-specific SWF file–this is just a warning. |
| FTPS038 | The call to `StartVibrate`/ `StopVibrate` was ignored because there was more than one request per frame or event. | Flash Player detected more than one instance of `FSCommand2()` `StartVibrate()` and `StopVibrate()` calls. Only one call per frame or event handler is allowed. |
| FTPS039 | `FSCommmand2 SetInputTextType(`*`text type`*`)` found, not supported in the emulator, please test it on the device. | Flash Player detected the `SetInputTextType()` command. It is not supported in the emulator; the type settings in the command are reported in the Output panel. |
| FTPS040 | MIDI sound found, not supported on these platforms: *platform name*. | Flash Player detected a MIDI sound. MIDI sound playback is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS041 | Mfi sound found, not supported on these platforms: *platform name*. | Flash Player detected a Generic Mfi sound. Mfi sound playback is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |

| Message Identifier | Message | Explanation |
|---|---|---|
| FTPS042 | SMAF sound found, not supported on these platforms: *platform name* | Flash Player detected a SMAF sound. SMAF sound playback is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS043 | MP3 sound found, not supported on these platforms: *platform name*. | Flash Player detected an MP3 sound. MP3 sound playback is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS044 | Streaming sound found, not supported on these platforms: *platform name*. | Flash Player detected a stream sound. Stream sound playback is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS045 | Input text field found, not supported on these platforms: *platform name*. | Flash Player detected an input text field. Input text is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS046 | Four way navigation is not supported on these platforms: *platform name*. | Flash Player is in four-way navigation mode. This mode is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS047 | Four way navigation with wraparound is not supported on these platforms: *platform name*. | Flash Player is in four-way navigation with wraparound mode. This mode is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS048 | Four way navigation is not supported. | Four-way navigation mode is not supported in the current configuration. |
| FTPS049 | Four way navigation with wraparound is not supported. | Four-way navigation with wraparound mode is not supported in the current configuration. |
| FTPS050 | Generic MFI sounds not supported. | Flash Player detected a generic MFI sound, it is not supported in current configuration. |
| FTPS051 | Unsupported mouse event found. | Flash Player detected an unsupported mouse event. |
| FTPS052 | ADPCM sound found, not supported on these platforms: *platform name*. | Flash Player detected an ADPCM sound. ADPCM sound playback is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |

| Message Identifier | Message | Explanation |
|---|---|---|
| FTPS053 | PCM sound found, not supported on these platforms: *platform name*. | Flash Player detected a PCM sound. PCM sound playback is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS054 | Sound found, not supported on these platforms: *platform name*. | Flash Player detected a sound. Sound playback is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS055 | Multiple Sounds found, sound mixing is not supported on these platforms: *platform name*. | Flash Player detected multiple sounds. Sound mixing is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS056 | Sound was ignored because it was not associated with a keypress, this feature is not supported on these platforms: *platform name*. | While testing a movie clip, a sound was encountered outside of a keypress event. The specified device's Flash Player allows sounds to be handled only during keypress events. Sounds outside of a keypress event are ignored. This feature is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS058 | `StartDrag` and `EndDrag` found, not supported on these platforms: *platform name*. | Flash Player detected a `StartDrag()` or `EndDrag()` ActionScript event. These events are not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS059 | Specific mouse event found, not supported on these platforms: *platform name*. | Flash Player detected certain mouse events.These events are not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS060 | A call to `loadVariables` found, might not be supported on these platforms: *platform name*. | Flash Player detected a `loadVariables` call. It will be executed, but it is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS061 | A call to `loadMovie` found, might not be supported on these platforms: *platform name*. | Flash Player detected a `loadMovie` call. It will be executed, but it is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |

| Message Identifier | Message | Explanation |
|---|---|---|
| FTPS062 | A call to `getURL` found, might not be supported on these platforms: *platform name*. | Flash Player detected a `getURL` call. It will be executed, but it is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS063 | A call to `fscommand()` found, might not be supported on these platforms: *platform name*. | Flash Player detected an `fscommand()` call. It will be executed. But it is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS064 | SMAF (MA-2) sound found, not supported on these platforms: *platform name*. | Flash Player detected a SMAF MA-2 sound. It is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS065 | SMAF (MA-3) sound found, not supported on these platforms: *platform name*. | Flash Player detected a SMAF MA-3 sound. It is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS066 | SMAF (MA-5) sound found, not supported on these platforms: *platform name*. | Flash Player detected a SMAF MA-5 sound. It is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS067 | SMAF(MA-2) sounds not supported. | Flash Player detected a SMAF MA-2 sound. It is not supported in the current configuration. |
| FTPS068 | SMAF(MA-3) sounds not supported. | Flash Player detected a SMAF MA-3 sound. It is not supported in the current configuration. |
| FTPS069 | SMAF(MA-5) sounds not supported. | Flash Player detected a SMAF MA-5 sound. It is not supported in the current configuration. |
| FTPS070 | MFI sounds with Fujitsu extension not supported. | Flash Player detected an MFI sound with the Fujitsu extension. It is not supported in the current configuration. |
| FTPS071 | MFI sounds with Mitsubishi extension not supported. | Flash Player detected an MFI sound with the Mitsubishi extension. It is not supported in the current configuration. |
| FTPS072 | MFI sounds with NEC extension not supported. | Flash Player detected an MFI sound with NEC extension. It is not supported in the current configuration. |
| FTPS073 | MFI sounds with Panasonic extension not supported. | Flash Player detected an MFI sound with the Panasonic extension. It is not supported in the current configuration. |
| FTPS074 | MFI sounds with Sharp extension not supported. | Flash Player detected an MFI sound with the Sharp extension. It is not supported in the current configuration. |

| Message Identifier | Message | Explanation |
|---|---|---|
| FTPS075 | MFI sounds with Sony extension not supported. | Flash Player detected an MFI sound with the Sony extension. It is not supported in the current configuration. |
| FTPS076 | MFI sounds with Fujitsu extension not supported on these platforms: *platform name*. | Flash Player detected an MFI sound with Fujitsu extension. It is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS077 | MFI sounds with Mitsubishi extension not supported on these platforms: *platform name*. | Flash Player detected an MFI sound with the Mitsubishi extension. It is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS078 | MFI sounds with NEC extension not supported on these platforms: *platform name*. | Flash Player detected an MFI sound with the NEC extension. It is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS079 | MFI sounds with Panasonic extension not supported on these platforms: *platform name*. | Flash Player detected an MFI sound with Panasonic extension. It is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS080 | MFI sounds with Sharp extension not supported on these platforms: *platform name*. | Flash Player detected an MFI sound with Sharp extension. It is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS081 | MFI sounds with Sony extension not supported on these platforms: *platform name*. | Flash Player detected an MFI sound with Sony extension. It is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS082 | ActionScript processing error found. *ActionScript command.* | Flash Player detected an ActionScript processing error. The warning message contains the ActionScript command that caused the processing error. |
| FTPS083 | Invalid entry found in configuration file: Line *line number*. | Flash Player detected an invalid entry on line *line number* in the configuration file. |
| FTPS084 | Configuration file found. | Flash Player detected a cofiguration file. |

| Message Identifier | Message | Explanation |
|---|---|---|
| FTPS085 | `loadVariables` requests are allowed only when associated with a keypress, not supported on these platforms: *platform name*. | Flash Player detected a `loadVariables` call associated with a keypress.<br>It will be executed, but it is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS086 | `loadMovie` requests are allowed only when associated with a keypress, not supported on these platforms: *platform name*. | Flash Player detected a `loadMovie` call associated with a keypress.<br>It will be executed, but it is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS087 | `getURL` requests are allowed only when associated with a keypress, not supported on these platforms: *platform name*. | Flash Player detected a `getURL()` call associated with a keypress.<br>It will be executed, but it is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS088 | `FSCommand()` requests are allowed only when associated with a keypress, not supported on these platforms: *platform name*. | Flash Player detected an `FSCommand()` call associated with a keypress.<br>It will be executed but it is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS089 | `loadVariables` requests are allowed on frame, not supported on these platforms: *platform name*. | A `loadVariables` call is detected by Flash Player. The call is allowed on a frame or when associated with a keypress, but this feature is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS090 | `loadMovie` requests are allowed on frame, not supported on these platforms: *platform name*. | Flash Player detected a `loadMovie()` call. The call is allowed on frame or when associated with a keypress, but this feature is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS091 | `getURL` requests are allowed on frame, not supported on these platforms: ‹platform name›. | Flash Player detected a `getURL()` call. The call is allowed on frame or when associated with a keypress, but this feature is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS092 | `FSCommand()` requests are allowed on frame, not supported on these platforms: ‹platform name›. | Flash Player detected an `FSCommand()` call. The call is allowed on frame or when associated with a keypress, but this feature is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |

| Message Identifier | Message | Explanation |
|---|---|---|
| FTPS093 | All the keys are allowed, this feature is not supported on these platforms: *platform name*. | Flash Player supports the full key set in the current configuration. This feature is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS094 | Only keys on the phone are allowed, this feature is not supported on these platforms: *platform name*. | Flash Player supports only the keys on the cell phone in the current configuration. This feature is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS095 | `_capEmail` is set to 1, this feature is not supported on these platforms: *platform name*. | Flash Player has set the platform capability variable `_capEmail` to 1 in the current configuration. This feature is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS096 | `_capSMS` is set to 1, this feature is not supported on these platforms: *platform name*. | Flash Player has set the platform capability variable `_capSMS` to 1 in the current configuration. This feature is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS097 | `_capMMS` is set to 1, this feature is not supported on these platforms: *platform name*. | Flash Player has set the platform capability variable `_capMMS` to 1 in the current configuration. This feature is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS098 | `_capLoadData` is set to 1, this feature is not supported on these platforms: *platform name*. | Flash Player has set the platform capability variable`_capLoadData` to 1 in current configuration. This feature is not supported on certain platforms. This message is shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS099 | Print Commands are not supported. | A Call to the Print command has been detected by the Flash Player, this feature is not supported. |
| FTPS100 | ‹sound format› sound is chosen in sound bundle. | A Flash Lite sound bundle has been detected by the Flash Player, the device sound in ‹sound format› has been chosen to be played back by the Flash Player. |
| FTPS101 | None of the format in the sound bundle file is supported. | A Flash Lite sound bundle has been detected by the Flash Player, none of the device sound format in the sound bundle is supported in the Flash Player. |

| Message Identifier | Message | Explanation |
|---|---|---|
| FTPS102 | SMAF sound playback not supported in the test movie player. | A SMAF format device sound has been detected in the Flash Player, while it is supported in the real device, it is not supported in the test movie player used in the authoring tool. |
| FTPS103 | Invalid tag ‹tag name› found in the local configuration file. | An invalid tag is detected in the local configuration file. |
| FTPS104 | No key is allowed, this feature is not supported on these platforms: ‹platform names› | The Flash Player has detected that the user set KeySetNone to “on” in the local configuration file and a keypress has been detected. This feature is not supported on certain platforms. This message will be shown only when platform strings are specified in corresponding flags in the configuration file. |
| FTPS105 | The SWF file is not in Flash Lite format. | The Flash Lite test movie player detected that the current SWF movie is not in Flash 4 format. |