macromedia®

FLASH® 8

Flash Lite 1.x ActionScript Language Reference

Acknowledgments

Project Management: Mary Leigh Burke

Writing: Guy Haas, Denise Green, Mike Krisher

Managing Editor: Rosana Francescato

Editing: Linda Adler, Geta Carlson, Evelyn Eldridge

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

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

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

First Edition: September 2005

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

Contents

Introduction

This manual describes the syntax and use of ActionScript elements as you use them to develop applications for Flash Lite 1.0 and Flash Lite 1.1, collectively referred to as Flash Lite 1.x. Flash Lite 1.x ActionScript is based on the version of ActionScript that was used in Flash 4. To use examples in a script, copy the code example from this manual, and paste it into the Script pane or into an external script file. The manual lists all ActionScript elements—operators, keywords, statements, commands, properties, functions, classes, and methods.

Sample entry for most ActionScript elements

The following sample entry explains the conventions used for all ActionScript elements.

Entry title

Entries are listed alphabetically within a chapter. The alphabetization ignores capitalization, leading underscores, and so on.

Availability

Unless otherwise noted, the Availability section tells which versions of Flash Lite support the element.

Usage

This section provides correct syntax for using the ActionScript element in your code. The required portion of the syntax is in `code font`. Both the code that you provide and data type information are in *`italicized code font`*. Data types can be distinguished from code that you provide by the colon (:) that always precedes data type information. Brackets (`[]`) indicate optional parameters.

Operands

This section describes any parameters listed in the syntax.

Description

This section identifies the type of element (for example, operator, function, and so on), what values if any that the element returns, and then describes how to use the element.

Example

This section provides a code sample demonstrating how to use the element.

See also

This section lists related ActionScript dictionary entries.

Samples folder

A set of sample files can be found in the /Samples and Tutorials/Samples/FlashLite/ directory within the Flash 8 installation directory.

Typical paths to this folder are as follows:

- Windows: /Program Files/Macromedia/Flash 8/Samples and Tutorials/Samples/FlashLite/
- Macintosh: HD/Applications/Macromedia/Flash 8/Samples and Tutorials/Samples/FlashLite/

The FlashLite folder contains a set of FLA files that are complete Flash Lite projects that have working ActionScript code.

Typographical conventions

The following typographical conventions are used in this book:

- *Italic font* indicates a value that should be replaced (for example, in a folder path).
- `Code font` indicates ActionScript code.
- *`Code font italic`* indicates an ActionScript parameter.
- **Bold font** indicates a verbatim entry.
- Double quotation marks (`" "`) in code examples indicate delimited strings. However, programmers can also use single quotation marks.

CHAPTER 1

Flash Lite Global Functions

1

This section describes the syntax and use of the Macromedia Flash Lite 1.1 ActionScript global functions. It includes the following functions:

Function	Description
call()	Executes the script in the called frame without moving the playhead to that frame.
chr()	Converts ASCII code numbers to characters.
duplicateMovieClip()	Creates an instance of a movie clip while the SWF file plays.
eval ()	Accesses variables, properties, objects, or movie clips by name.
getProperty()	Returns the value of the specified property for the specified movie clip.
getTimer()	Returns the number of milliseconds that elapsed since the SWF file started playing.
getURL()	Loads a document from a specific URL into a window or passes variables to another application at a defined URL.
gotoAndPlay()	Sends the playhead to the specified frame in a scene and begins playing from that frame. If no scene is specified, the playhead moves to the specified frame in the current scene.
gotoAndStop()	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.
ifFrameLoaded()	Checks whether the contents of a specific frame are available locally.
int()	Truncates a decimal number to an integer value.
length()	Returns the number of characters of the specified string or variable name.
loadMovie()	Loads a SWF file into Flash Lite while the original SWF file plays.
loadMovieNum()	Loads a SWF file into a level in Flash Lite while the originally loaded SWF file plays.

Function	Description
`loadVariables()`	Reads data from an external file, such as a text file or text generated by a ColdFusion, CGI ASP, PHP, or Perl script, and sets the values for variables in a Flash Lite level. This function can also update variables in the active SWF file with new values.
`loadVariablesNum()`	Reads data from an external file, such as a text file or text generated by a ColdFusion, CGI, ASP, PHP, or Perl script, and sets the values for variables in a Flash Lite level. This function can also update variables in the active SWF file with new values.
`mbchr()`	Converts an ASCII code number to a multibyte character.
`mblength()`	Returns the length of the multibyte character string.
`mbord()`	Converts the specified character to a multibyte number.
`mbsubstring()`	Extracts a new multibyte character string from a multibyte character string.
`nextFrame()`	Sends the playhead to the next frame and stops it.
`nextScene()`	Sends the playhead to Frame 1 of the next scene and stops it.
`Number()`	Converts an expression to a number and returns a value.
`on()`	Specifies the user event or keypress that triggers an event.
`ord()`	Converts characters to ASCII code numbers.
`play()`	Moves the playhead forward in the timeline.
`prevFrame()`	Sends the playhead to the previous frame and stops it. If the current frame is Frame 1, the playhead does not move.
`prevScene()`	Sends the playhead to Frame 1 of the previous scene and stops it.
`removeMovieClip()`	Deletes the specified movie clip that was originally created using `duplicateMovieClip()`.
`set()`	Assigns a value to a variable.
`setProperty()`	Changes a property value of a movie clip as the movie plays.
`stop()`	Stops the SWF file that is currently playing.
`stopAllSounds()`	Stops all sounds currently playing in a SWF file without stopping the playhead.
`String()`	Returns a string representation of the specified parameter.
`substring()`	Extracts part of a string.
`tellTarget()`	Applies the instructions specified in the *`statement(s)`* parameter to the timeline specified in the *`target`* parameter.

Function	Description
toggleHighQuality()	Turns anti-aliasing on and off in Flash Lite. Anti-aliasing smooths the edges of objects but slows down SWF file playback.
trace()	Evaluates the expression and shows the result in the Output panel in test mode.
unloadMovie()	Removes a movie clip from Flash Lite that was loaded using loadMovie() loadMovieNum(), or duplicateMovieClip().
unloadMovieNum()	Removes a movie clip that was loaded using loadMovie() loadMovieNum(), or duplicateMovieClip() from a level in Flash Lite.

call()

Availability

Flash Lite 1.0.

Usage

```
call(frame)

call(movieClipInstance:frame)
```

Operands

frame The label or number of a frame in the timeline.

movieClipInstance The instance name of a movie clip.

Description

Function; executes the script in the called frame without moving the playhead to that frame. Local variables do not exist after the script executes. The call() function can take two possible forms:

- The default form executes the script on the specified frame of the same timeline where the call() function was executed, without moving the playhead to that frame.
- The specified clip instance form executes the script on the specified frame of the movie clip instance, without moving the playhead to that frame.

NOTE

The call() function operates synchronously; any ActionScript that follows a call() function does not execute until all of the ActionScript at the specified frame has completed.

Example

The following examples execute the script in the `myScript` frame:

```
// to execute functions in frame with label "myScript"
thisFrame = "myScript";
trace ("Calling the script in frame: " add thisFrame);

// to execute functions in any other frame on the same timeline
call("myScript");
```

chr()

Availability

Flash Lite 1.0.

Usage

`chr(`*`number`*`)`

Operands

`number` An ASCII code number.

Description

String function; converts ASCII code numbers to characters.

Example

The following example converts the number 65 to the letter *A* and assigns it to the variable `myVar`:

```
myVar = chr(65);
trace (myVar);// Output: A
```

duplicateMovieClip()

Availability

Flash Lite 1.0.

Usage

`duplicateMovieClip(`*`target`*`, `*`newname`*`, `*`depth`*`)`

Operands

target The target path of the movie clip to duplicate.

newname A unique identifier for the duplicated movie clip.

depth A unique depth level for the duplicated movie clip. The depth level indicates a stacking order for duplicated movie clips. This stacking order is much like the stacking order of layers in the timeline; movie clips with a lower depth level are hidden under clips that have a higher depth level. You must assign to each duplicated movie clip a unique depth level so that it does not overwrite existing movie clips on occupied depth levels.

Description

Function; creates an instance of a movie clip while the SWF file plays and returns a reference to the duplicated movie clip. The playhead in a duplicate movie clip always starts at Frame 1, regardless of where the playhead is in the original (parent) movie clip. Variables in the parent movie clip are not copied into the duplicate movie clip. If the parent movie clip is deleted, the duplicate movie clip is also deleted. Use the `removeMovieClip()` function or method to delete a movie clip instance created with `duplicateMovieClip()`.

Example

The following example duplicates a movie clip named `originalClip` to create a new clip named `newClip`, at a depth level of 10. The new clip's *x* position is set to 100 pixels.

```
duplicateMovieClip("originalClip", "newClip", 10);
setProperty("newClip", _x, 100);
```

The following example uses `duplicateMovieClip()` in a `for` loop to create several new movie clips at once. An index variable keeps track of the highest occupied stacking depth. Each duplicate movie clip's name contains a numeric suffix that corresponds to its stacking depth (`clip1`, `clip2`, `clip3`).

```
for (i = 1; i <= 3; i++) {
  newName = "clip" add i;
  duplicateMovieClip("originalClip", newName); }
```

See also

`removeMovieClip()`

eval ()

Availability

Flash Lite 1.0.

Usage

`eval(`*`expression`*`)`

Operands

`expression` A string containing the name of a variable, property, object, or movie clip to retrieve.

Description

Function; accesses variables, properties, objects, or movie clips by name. If *`expression`* is a variable or a property, the value of the variable or property is returned. If *`expression`* is an object or movie clip, a reference to the object or movie clip is returned. If the element named in *`expression`* cannot be found, `undefined` is returned.

You can use `eval()` to simulate arrays, or to dynamically set and retrieve the value of a variable.

Example

The following example uses `eval()` to determine the value of the expression `"piece" + x`. Because the result is a variable name, `piece3`, `eval()` returns the value of the variable and assigns it to `y`:

```
piece3 = "dangerous";
x = 3;
y = eval("piece" add x);
trace(y);// Output: dangerous.
```

The following example demonstrates how an array could be simulated:

```
name1 = "mike";
name2 = "debbie";
name3 = "logan";
for(i = 1; i <= 3; i++) {
  trace (eval("name" add i));// Output: mike, debbie, logan
}
```

getProperty()

Availability

Flash Lite 1.0.

Usage

```
getProperty(my_mc, property)
```

Operands

my_mc The instance name of a movie clip for which the property is being retrieved.

property A property of a movie clip.

Description

Function; returns the value of the specified property for the *my_mc* movie clip.

Example

The following example retrieves the horizontal axis coordinate (_x) for the my_mc movie clip in the root movie timeline:

```
xPos = getProperty("person_mc", _x);
trace (xPos);   // output: -75
```

See also

setProperty()

getTimer()

Availability

Flash Lite 1.0.

Usage

```
getTimer()
```

Operands

None.

Description

Function; returns the number of milliseconds that elapsed since the SWF file started playing.

Example

The following example sets the `timeElapsed` variable to the number of milliseconds that elapsed since the SWF file started playing:

```
timeElapsed = getTimer();
trace (timeElapsed);// Output: milliseconds of time movie has been playing
```

getURL()

Availability

Flash Lite 1.0.

Usage

```
getURL(url [ , window [, "variables"]])
```

Operands

url The URL from which to obtain the document.

window An optional parameter that specifies the window or HTML frame that the document should load into.

NOTE	The *window* parameter is not specified for Flash Lite applications, because browsers on cell phones do not support multiple windows.

You can enter an empty string, or the name of a specific window, or choose from the following reserved target names:

- `_self` specifies the current frame in the current window.
- `_blank` specifies a new window.
- `_parent` specifies the parent of the current frame.
- `_top` specifies the top-level frame in the current window.

variables A `GET` or `POST` method for sending variables. If there are no variables, omit this parameter. The `GET` method appends the variables to the end of the URL and is used for small numbers of variables. The `POST` method sends the variables in a separate HTTP header and is used for sending long strings of variables.

Description

Function; loads a document from a specific URL into a window or passes variables to another application at a defined URL. To test this function, make sure the file you want to load is in the specified location. To use an absolute URL (for example, http://www.myserver.com), you need a network connection.

Flash Lite 1.0 recognizes only the HTTP, HTTPS, mailto, and tel protocols. Flash Lite 1.1 recognizes these protocols, and in addition, the file, SMS (short message service), and MMS (multimedia message service) protocols.

Flash Lite passes the call to the operating system, and the operating system handles the call with the registered default application for the specified protocol.

Only one `getURL()` function is processed per frame or per event handler.

Certain handsets restrict the `getURL()` function to key events only, in which case the `getURL()` call is processed only if it is triggered in a key event handler. Even under such circumstances, only one `getURL()` function is processed per event handler.

Example

In the following ActionScript, the Flash Lite player opens mobile.macromedia.com in the default browser:

```
myURL = "http://mobile.macromedia.com";
 on(keyPress "1") {
  getURL(myURL);
}
```

You can also use `GET` or `POST` for sending variables from the current timeline. The following example uses the `GET` method to append variables to a URL:

```
firstName = "Gus";
lastName = "Richardson";
age = 92;
getURL("http://www.macromedia.com", "_blank", "GET");
```

The following ActionScript uses `POST` to send variables in an HTTP header:

```
firstName = "Gus";
lastName = "Richardson";
age = 92;
getURL("http://www.macromedia.com", "POST");
```

You can assign a button function to open an e-mail composition window with the `address`, `subject`, and `body` text fields already populated. Use one of the following methods to assign a button function: Method 1 for either Shift-JIS or English character encoding; Method 2 only for English character encoding.

Method 1: Set variables for each of the desired parameters, as in this example:

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

Method 2: Define each parameter within the `getURL()` function, as in this 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 example, the strings that result from using Method 1 are as follows:

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

In contrast, Method 2 results in the following strings:

```
email subject
email body
```

The following example uses the tel protocol:

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

In the following example, `getURL()` is used to send an SMS message:

```
mySMS = "sms:4156095555?body=sample sms message";
on(keyPress "5") {
  getURL(mySMS);
}
```

In the following example, `getURL()` is used to send an MMS message:

```
// mms example
myMMS = "mms:4156095555?body=sample mms message";
on(keyPress "6") {
  getURL(myMMS);
}
```

In the following example, `getURL()` is used to open a text file stored on the local file system:

```
// file protocol example
filePath = "file://c:/documents/flash/myApp/myvariables.txt";
on(keyPress "7") {
  getURL(filePath);
}
```

gotoAndPlay()

Availability

Flash Lite 1.0.

Usage

```
gotoAndPlay([scene,] frame)
```

Operands

scene An optional string specifying the name of the scene to which the playhead is sent.

frame A number representing the frame number, or a string representing the label of the frame, to which the playhead is sent.

Description

Function; sends the playhead to the specified frame in a scene and begins playing from that frame. If no scene is specified, the playhead moves to the specified frame in the current scene.

You can use the *scene* parameter only on the root timeline, not within timelines for movie clips or other objects in the document.

Example

In the following example, when the user clicks a button to which `gotoAndPlay()` is assigned, the playhead moves to Frame 16 in the current scene and starts to play the SWF file:

```
on(keyPress "7") {
  gotoAndPlay(16);
}
```

gotoAndStop()

Availability

Flash 1.0.

Usage

```
gotoAndStop([scene,] frame)
```

Operands

scene An optional string specifying the name of the scene to which the playhead is sent.

frame A number representing the frame number, or a string representing the label of the frame, to which the playhead is sent.

Description

Function; 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.

You can use the *scene* parameter only on the root timeline, not within timelines for movie clips or other objects in the document.

Example

In the following example, when the user clicks a button to which `gotoAndStop()` is assigned, the playhead is sent to Frame 5 in the current scene, and the SWF file stops playing:

```
on(keyPress "8") {
  gotoAndStop(5);
}
```

ifFrameLoaded()

Availability

Flash Lite 1.0.

Usage

```
ifFrameLoaded([scene,] frame) {
  statement(s);
}
```

Operands

scene An optional string specifying the name of the scene to be loaded.

frame The frame number or frame label to be loaded before the next statement can execute.

statement(s) The instructions to execute if the specified frame, or scene and frame, are loaded.

Description

Function; checks whether the contents of a specific frame are available locally. Use the `ifFrameLoaded` function to start playing a simple animation while the rest of the SWF file downloads to the local computer. You can also use the `_framesloaded` property to check the download progress of an external SWF file. The difference between using `_framesloaded` and `ifFrameLoaded` is that `_framesloaded` lets you add custom `if` or `else` statements.

Example

The following example uses the `ifFrameLoaded` function to check if Frame 10 of the SWF file is loaded. If the frame is loaded, the `trace()` command prints "frame number 10 is loaded" to the Output panel. The output variable is also defined with a variable of `frame loaded: 10`.

```
ifFrameLoaded(10) {
  trace ("frame number 10 is loaded");
  output = "frame loaded: 10";
 }
```

See also

`_framesloaded`

int()

Availability

Flash Lite 1.0.

Usage

`int(`*`value`*`)`

Operands

`value` A number or string to be truncated to an integer.

Description

Function; truncates a decimal number to an integer value.

Example

The following example truncates the numbers in the `distance` and `myDistance` variables:

```
distance = 6.04 - 3.96;
//trace ("distance = " add distance add " and rounded to:" add
  int(distance));
// Output: distance = 2.08 and rounded to: 2
myDistance = "3.8";
//trace ("myDistance = " add int(myDistance));
// Output: 3
```

length()

Availability

Flash Lite 1.0.

Usage

```
length(expression)

length(variable)
```

Operands

expression A string.

variable The name of a variable.

Description

String function; returns the number of characters of the specified string or variable name.

Example

The following example returns the length of the string `"Hello"`:

```
length("Hello");
```

The result is 5.

The following example validates an e-mail address by checking that it contains at least six characters:

```
email = "someone@macromedia.com";
if (length(email) > 6) {
  //trace ("email appears to have enough characters to be valid");
}
```

loadMovie()

Availability

Flash Lite 1.1.

Usage

```
loadMovie(url, target [, method])
```

Operands

url A string specifying the absolute or relative URL of the SWF file to load. A relative path must be relative to the SWF file at level 0. Absolute URLs must include the protocol reference, such as http:// or file:///.

target A reference to a movie clip or a string representing the path to a target movie clip. The target movie clip is replaced by the loaded SWF file.

method An optional string parameter specifying an HTTP method for sending variables. The parameter must be the string `GET` or `POST`. If there are no variables to be sent, omit this parameter. The `GET` method appends the variables to the end of the URL and is used for small numbers of variables. The `POST` method sends the variables in a separate HTTP header and is used for long strings of variables.

Description

Function; loads a SWF file into Flash Lite while the original SWF file plays.

To load a SWF file into a specific level, use the `loadMovieNum()` function instead of `loadMovie()`.

When a SWF file is loaded into a target movie clip, you can use the target path of that movie clip to target the loaded SWF file. A SWF file loaded into a target inherits the position, rotation, and scale properties of the targeted movie clip. The upper-left corner of the loaded image or SWF file aligns with the registration point of the targeted movie clip. However, if the target is the root timeline, the upper-left corner of the image or SWF file aligns with the upper-left corner of the Stage.

Use the `unloadMovie()` function to remove SWF files that were loaded with `loadMovie()`.

Example

The following example loads the SWF file circle.swf from the same directory and replaces a movie clip called `mySquare` that already exists on the Stage:

```
loadMovie("circle.swf", "mySquare");
// Equivalent statement: loadMovie("circle.swf", _level0.mySquare);
```

See also

`_level`, `loadMovieNum()`, `unloadMovie()`, `unloadMovieNum()`

loadMovieNum()

Availability

Flash Lite 1.1.

Usage

`loadMovieNum(url, level [, method])`

Operands

`url` A string specifying the absolute or relative URL of the SWF file to be loaded. A relative path must be relative to the SWF file at level 0. For use in the stand-alone Flash Lite player or for use in test mode in the Flash authoring application, all SWF files must be stored in the same folder and the filenames cannot include folder or drive specifications.

`level` An integer specifying the level in Flash Lite where the SWF file loads.

`method` An optional string parameter specifying an HTTP method for sending variables. It must have the value `GET` or `POST`. If there are no variables to be sent, omit this parameter. The `GET` method appends the variables to the end of the URL and is used for small numbers of variables. The `POST` method sends the variables in a separate HTTP header and is used for long strings of variables.

Description

Function; loads a SWF file into a level in Flash Lite while the originally loaded SWF file plays.

Normally, Flash Lite displays a single SWF file and then closes. The `loadMovieNum()` function lets you display several SWF files at once and switch among SWF files without loading another HTML document.

To specify a target instead of a level, use the `loadMovie()` function instead of `loadMovieNum()`.

Flash Lite has a stacking order of levels starting with level 0. These levels are like layers of acetate; they are transparent except for the objects on each level. When you use `loadMovieNum()`, you must specify a level in Flash Lite where the SWF file will load. When a SWF file is loaded into a level, you can use the syntax `_level`*`N`*, where *`N`* is the level number, to target the SWF file.

When you load a SWF file, you can specify any level number. You can load SWF files into a level that already has a SWF file loaded into it, and the new SWF file replaces the existing file. If you load a SWF file into level 0, every level in Flash Lite is unloaded, and level 0 is replaced with the new file. The SWF file in level 0 sets the frame rate, background color, and frame size for all other loaded SWF files.

Use `unloadMovieNum()` to remove SWF files or images that were loaded with `loadMovieNum()`.

Example

The following example loads the SWF file into level 2:

```
loadMovieNum("http://www.someserver.com/flash/circle.swf", 2);
```

See also

`_level`, `loadMovie()`, `unloadMovieNum()`

loadVariables()

Availability

Flash Lite 1.1.

Usage

```
loadVariables(url, target [, variables])
```

Operands

`url` A string representing an absolute or relative URL where the variables are located. If the SWF file issuing this call is running in a web browser, *`url`* must be in the same domain as the SWF file.

`target` The target path to a movie clip that receives the loaded variables.

`variables` An optional string parameter specifying an HTTP method for sending variables. The parameter must be the string `GET` or `POST`. If there are no variables to be sent, omit this parameter. The `GET` method appends the variables to the end of the URL and is used for small numbers of variables. The `POST` method sends the variables in a separate HTTP header and is used for long strings of variables.

Description

Function; reads data from an external file, such as a text file or text generated by a ColdFusion, CGI, Active Server Pages (ASP), PHP, or Perl script, and sets the values for variables in a target movie clip. This function can also update variables in the active SWF file with new values.

The text at the specified URL must be in the standard MIME format *application/x-www-form-urlencoded* (a standard format used by CGI scripts). Any number of variables can be specified. For example, the following phrase defines several variables:

```
company=Macromedia&address=600+Townsend&city=San+Francisco&zip=94103
```

To load variables into a specific level, use the `loadVariablesNum()` function instead of the `loadVariables()` function.

Example

The following examples load variables from a text file and from a server:

```
// load variables from text file on local file system (Symbian Series 60)
on(release, keyPress "1") {
  filePath = "file://c:/documents/flash/myApp/myvariables.txt";
  loadVariables(filePath, _root);
}

// load variables (from server) into a movieclip
urlPath = "http://www.someserver.com/myvariables.txt";
loadVariables(urlPath, "myClip_mc");
```

See also

`loadMovieNum()`, `loadVariablesNum()`, `unloadMovie()`

loadVariablesNum()

Availability

Flash Lite 1.1.

Usage

`loadVariablesNum(`*`url`*`, `*`level`*` [, `*`variables`*`])`

Operands

`url` A string representing an absolute or relative URL where the variables to be loaded are located. If the SWF file issuing this call is running in a web browser, *`url`* must be in the same domain as the SWF file; for more information, see the following Description section.

`level` An integer that specifies the level in Flash Lite to receive the variables.

`variables` An optional string parameter specifying an HTTP method for sending variables. The parameter must be the string `GET` or `POST`. If there are no variables to be sent, omit this parameter. The `GET` method appends the variables to the end of the URL and is used for small numbers of variables. The `POST` method sends the variables in a separate HTTP header and is used for long strings of variables.

Description

Function; reads data from an external file, such as a text file or text generated by a ColdFusion, CGI, ASP, PHP, or Perl script, and sets the values for variables in a Flash Lite level. This function can also update variables in the active SWF file with new values.

The text at the specified URL must be in the standard MIME format *application/x-www-form-urlencoded* (a standard format used by CGI scripts). Any number of variables can be specified. The following example phrase defines several variables:

```
company=Macromedia&address=600+Townsend&city=San+Francisco&zip=94103
```

Normally, Flash Lite displays a single SWF file, and then closes. The `loadVariablesNum()` function lets you display several SWF files at once and switch among SWF files without loading another HTML document.

To load variables into a target movie clip, use the `loadVariables()` function instead of the `loadVariablesNum()` function.

See also

`getURL()`, `loadMovie()`, `loadMovieNum()`, `loadVariables()`

mbchr()

Availability

Flash Lite 1.0.

Usage

`mbchr(`*`number`*`)`

Operands

`number` The number to convert to a multibyte character.

Description

String function; converts an ASCII code number to a multibyte character.

Example

The following example converts ASCII code numbers to their mulitibyte character equivalents:

```
trace (mbchr(65));// Output: A
trace (mbchr(97));// Output: a
trace (mbchr(36));// Output: $

myString = mbchr(51) - mbchr(49);
trace ("result = " add myString);// Output: result = 2
```

See also

mblength(), mbsubstring()

mblength()

Availability

Flash Lite 1.0.

Usage

```
mblength(string)
```

Operands

string A string.

Description

String function; returns the length of the multibyte character string.

Example

The following example displays the length of the string in the `myString` variable:

```
myString = mbchr(36) add mbchr(50);
trace ("string length = " add mblength(myString));
// Output: string length = 2
```

See also

mbchr(), mbsubstring()

mbord()

Availability

Flash Lite 1.0.

Usage

`mbord(character)`

Operands

character The character to convert to a multibyte number.

Description

String function; converts the specified character to a multibyte number.

Example

The following examples convert the characters in the `myString` variable to multibyte numbers:

```
myString = "A";
trace ("ord = " add mbord(myString));// Output: 65

myString = "$120";
for (i=1; i<=length(myString); i++)
  char = substring(myString, i, 1);
  trace ("char ord = " add mbord(char));// Output: 36, 49, 50, 48
}
```

See also

`mbchr()`, `mbsubstring()`

mbsubstring()

Availability

Flash Lite 1.0.

Usage

`mbsubstring(value, index, count)`

Operands

value The multibyte string from which to extract a new multibyte string.

index The number of the first character to extract.

count The number of characters to include in the extracted string, not including the index character.

Description

String function; extracts a new multibyte character string from a multibyte character string.

Example

The following example extracts a new multibyte character string from the string contained in the `myString` variable:

```
myString = mbchr(36) add mbchr(49) add mbchr(50) add mbchr(48);
trace (mbsubstring(myString, 0, 2));// Output: $1
```

See also

`mbchr()`

nextFrame()

Availability

Flash Lite 1.0.

Usage

```
nextFrame()
```

Operands

None.

Description

Function; sends the playhead to the next frame and stops it.

Example

In the following example, when the user clicks the button, the playhead moves to the next frame and stops:

```
on (release) {
  nextFrame();
}
```

See also

prevFrame()

nextScene()

Availability

Flash Lite 1.0.

Usage

```
nextScene()
```

Operands

None.

Description

Function; sends the playhead to Frame 1 of the next scene and stops it.

Example

In the following example, when a user releases the button, the playhead moves to Frame 1 of the next scene:

```
on(release) {
  nextScene();
}
```

See also

prevScene()

Number()

Availability

Flash Lite 1.0.

Usage

`Number(`*`expression`*`)`

Operands

`expression` An expression to convert to a number.

Description

Function; converts the parameter *`expression`* to a number and returns a value as described in the following list:

- If *`expression`* is a number, the return value is *`expression`*.
- If *`expression`* is a Boolean value, the return value is 1 if *`expression`* is `true`; 0 if *`expression`* is `false`.
- If *`expression`* is a string, the function attempts to parse *`expression`* as a decimal number with an optional trailing exponent (that is, 1.57505e-3).
- If *`expression`* is `undefined`, the return value is -1.

Example

The following example converts the string in the `myString` variable to a number, stores the number in the `myNumber` variable, adds 5 to the number, and stores the result in the variable `myResult`. The final line shows the result when you call `Number()` on a Boolean value.

```
myString = "55";
myNumber = Number(myString);
myResult = myNumber + 5;

trace (myResult); // Output: 60

trace (Number(true));// Output: 1
```

on()

Availability

Flash Lite 1.0.

Usage

```
on(event) {
  // statement(s)
}
```

Operands

statement(s) The instructions to execute when *event* occurs.

event This trigger is called an *event*. When a user event occurs, the statements following it within curly braces ({}) execute. Any of the following values can be specified for the *event* parameter:

- press The button is pressed while the pointer is over the button.
- release The button is released while the pointer is over the button.
- rollOut The pointer rolls outside the button area.
- rollOver The pointer rolls over the button.
- keyPress "*key*" The specified key is pressed. For the key portion of the parameter, specify a key code or key constant.

Description

Event handler; specifies the user event or keypress that triggers a function. Not all events are supported.

Example

The following code, which scrolls the myText field down one line when the user presses the 8 key, tests against maxscroll before scrolling:

```
on (keyPress "8") {
  if (myText.scroll < myText.maxscroll) {
    myText.scroll++;
  }
}
```

ord()

Availability

Flash Lite 1.0.

Usage

```
ord(character)
```

Operands

character The character to convert to an ASCII code number.

Description

String function; converts characters to ASCII code numbers.

Example

The following example uses the `ord()` function to display the ASCII code for the character *A*:

```
trace ("multibyte number = " add ord("A"));// Output: multibyte number = 65
```

play()

Availability

Flash Lite 1.0.

Usage

```
play()
```

Operands

None.

Description

Function; moves the playhead forward in the timeline.

Example

The following example uses an `if` statement to check the value of a name that the user enters. If the user enters `Steve`, the `play()` function is called, and the playhead moves forward in the timeline. If the user enters anything other than `Steve`, the SWF file does not play, and a text field with the variable name `alert` appears.

```
stop();
if (name == "Steve") {
  play();
} else {
  alert="You are not Steve!";
}
```

prevFrame()

Availability

Flash Lite 1.0.

Usage

```
prevFrame()
```

Operands

None.

Description

Function; sends the playhead to the previous frame and stops it. If the current frame is Frame 1, the playhead does not move.

Example

When the user clicks a button that has the following handler attached to it, the playhead is sent to the previous frame:

```
on(release) {
  prevFrame();
}
```

See also

`nextFrame()`

prevScene()

Availability

Flash Lite 1.0.

Usage

```
prevScene()
```

Operands

None.

Description

Function; sends the playhead to Frame 1 of the previous scene and stops it.

Example

In this example, when the user clicks a button that has the following handler attached to it, the playhead is sent to the previous scene:

```
on(release) {
  prevScene();
}
```

See also

`nextScene()`

random()

Availability

Flash Lite 1.0.

Usage

```
random(value)
```

Operands

value An integer.

Description

Function; returns a random integer between 0 and one less than the integer specified in the *value* parameter.

Example

The following examples generate a number based on an integer specifying the range:

```
//pick random number between 0 and 5
myNumber = random(5);
trace (myNumber);// Output: could be 0,1,2,3,4

//pick random number between 5 and 10
myNumber = random(5) + 5;
trace (myNumber);// Output: could be 5,6,7,8,9
```

The following examples generate a number, and then concatenate it onto the end of a string being evaluated as a variable name. This is an example of how Flash Lite 1.1 syntax can be used to simulate arrays.

```
// select random name from list
myNames1 = "Mike";
myNames2 = "Debbie";
myNames3 = "Logan";

ran = random(3) + 1;
ranName = "myNames" add ran;
trace (eval(ranName));// Output: will be mike, debbie, or logan
```

removeMovieClip()

Availability

Flash Lite 1.0.

Usage

`removeMovieClip(`*`target`*`)`

Operands

`target` The target path of a movie clip instance created with `duplicateMovieClip()`.

Description

Function; deletes the specified movie clip that was originally created using `duplicateMovieClip()`.

Example

The following example deletes the duplicate movie clip named `second_mc`:

```
duplicateMovieClip("person_mc", "second_mc", 1);
second_mc:_x = 55;
second_mc:_y = 85;
removeMovieClip("second_mc");
```

set()

Availability

Flash Lite 1.0.

Usage

`set(variable, expression)`

Operands

`variable` An identifier to hold the value of the *`expression`* parameter.

`expression` A value assigned to the variable.

Description

Statement; assigns a value to a variable. A *variable* is a container that holds data. The container is always the same, but the contents can change. By changing the value of a variable as the SWF file plays, you can record and save information about what the user has done, record values that change as the SWF file plays, or evaluate whether a condition is `true` or `false`.

Variables can hold any data type (for example, String, Number, Boolean, or MovieClip). The timeline of each SWF file and movie clip has its own set of variables, and each variable has its own value that is independent of variables on other timelines.

Example

The following example sets a variable called `orig_x_pos`, which stores the original *x* axis position of the `ship` movie clip to reset the ship to its starting location later in the SWF file:

```
on(release) {
  set("orig_x_pos", getProperty("ship", _x));
}
```

The preceding code gives the same result as the following code:

```
on(release) {
  orig_x_pos = ship._x;
}
```

setProperty()

Availability

Flash Lite 1.0.

Usage

```
setProperty(target, property, value/expression)
```

Operands

target The path to the instance name of the movie clip whose property is to be set.

property The property to be set.

value The new literal value of the property.

expression An equation that evaluates to the new value of the property.

Description

Function; changes a property value of a movie clip as the movie plays.

Example

The following statement sets the `_alpha` property of the `star` movie clip to 30 percent when the user clicks the button associated with this event handler:

```
on(release) {
  setProperty("star", _alpha, "30");
}
```

See also

`getProperty()`

stop()

Availability

Flash Lite 1.0.

Usage

```
stop()
```

Operands

None.

Description

Function; stops the SWF file that is currently playing. The most common use of this function is to control movie clips with buttons.

Example

The following statement calls the `stop()` function when the user clicks the button associated with this event handler:

```
on(release) {
  stop();
}
```

stopAllSounds()

Availability

Flash Lite 1.0.

Usage

```
stopAllSounds()
```

Operands

None.

Description

Function; stops all sounds currently playing in a SWF file without stopping the playhead. Sounds set to stream will resume playing as the playhead moves over the frames that contain them.

Example

The following code could be applied to a button that when clicked, stops all sounds in the SWF file:

```
on(release) {
  stopAllSounds();
}
```

String()

Availability

Flash Lite 1.0.

Usage

`String(expression)`

Operands

expression An expression to convert to a string.

Description

Function; returns a string representation of the specified parameter as described in the following list:

- If *expression* is a number, the return string is a text representation of the number.
- If *expression* is a string, the return string is *expression*.
- If *expression* is a Boolean value, the return string is `true` or `false`.
- If *expression* is a movie clip, the return value is the target path of the movie clip in slash (/) notation.

Example

The following example sets `birthYearNum` to 1976, converts it to a string using the `String()` function, and then compares it to the string "1976" by using the `eq` operator.

```
birthYearNum = 1976;
birthYearStr = String(birthYearNum);
if (birthYearStr eq "1976") {
  trace ("birthYears match");
}
```

substring()

Availability

Flash Lite 1.0.

Usage

`substring(string, index, count)`

Operands

string The string from which to extract the new string.

index The number of the first character to extract.

count The number of characters to include in the extracted string, not including the index character.

Description

Function; extracts part of a string. This function is one-based, whereas the String class methods are zero-based.

Example

The following example extracts the first five characters from the string "Hello World":

```
origString = "Hello World!";
newString = substring(origString, 0, 5);
trace (newString);// Output: Hello
```

tellTarget()

Availability

Flash Lite 1.0.

Usage

```
tellTarget(target) {
  statement(s);
}
```

Operands

target A string that specifies the target path of the timeline to control.

statement(s) The instructions to execute if the condition evaluates to `true`.

Description

Function; applies the instructions specified in the *statement(s)* parameter to the timeline specified in the *target* parameter. The `tellTarget()` function is useful for navigation controls. Assign `tellTarget()` to buttons that stop or start movie clips elsewhere on the Stage. You can also make movie clips go to a particular frame in that clip. For example, you might assign `tellTarget()` to buttons that stop or start movie clips on the Stage or prompt movie clips to move to a particular frame.

Example

In the following example, `tellTarget()` controls the `ball` movie clip instance on the main timeline. Frame 1 of the `ball` instance is blank and has a `stop()` function so that it isn't visible on the Stage. When the user presses the 5 key, `tellTarget()` tells the playhead in `ball` to go to Frame 2 where the animation starts.

```
on(keyPress "5") {
  tellTarget("ball") {
    gotoAndPlay(2);
  }
}
```

toggleHighQuality()

Availability

Flash Lite 1.0.

Usage

```
toggleHighQuality()
```

Operands

None.

Description

Function; turns anti-aliasing on and off in Flash Lite. Anti-aliasing smooths the edges of objects but slows down SWF file playback. This function affects all SWF files in Flash Lite.

Example

The following code could be applied to a button that when clicked, would toggle anti-aliasing on and off:

```
on(release) {
  toggleHighQuality();
}
```

trace()

Availability

Flash Lite 1.0.

Usage

```
trace(expression)
```

Operands

expression An expression to evaluate. When a SWF file opens in the Flash authoring tool (by means of the Test Movie command), the value of the *expression* parameter appears in the Output panel.

Description

Function; evaluates the expression and shows the result in the Output panel in test mode.

Use this function to record programming notes or to display messages in the Output panel while testing a SWF file. Use the *expression* parameter to check if a condition exists, or to display values in the Output panel. The `trace()` function is similar to the `alert` function in JavaScript.

You can use the Omit Trace Actions command in publish settings to remove `trace()` functions from the exported SWF file.

Example

The following example uses the `trace()` function to observe the behavior of a `while` loop:

```
i = 0;
while (i++ < 5){
  trace("this is execution " add i);
}
```

unloadMovie()

Availability

Flash Lite 1.0.

Usage

```
unloadMovie(target)
```

Operands

target The target path of a movie clip.

Description

Function; removes a movie clip from Flash Lite that was loaded by means of `loadMovie()`, `loadMovieNum()`, or "duplicateMovieClip()".

Example

When the user presses the 3 key, the following code responds by unloading the `draggable_mc` movie clip on the main timeline and loading movie.swf into level 4 of the document stack:

```
on (keypress "3") {
  unloadMovie ("/draggable_mc");
  loadMovieNum("movie.swf", 4);
}
```

When the user presses the 3 key, the following example unloads the movie that was loaded into level 4:

```
on (keypress "3") {
    unloadMovieNum(4);
}
```

See also

`loadMovie()`

unloadMovieNum()

Availability

Flash Lite 1.0.

Usage

`unloadMovieNum(`*`level`*`)`

Operands

`level` The level (`_level`*`N`*) of a loaded movie.

Description

Function; removes a movie clip from Flash Lite that was loaded by means of `loadMovie()`, `loadMovieNum()`, or "duplicateMovieClip()".

Normally, Flash Lite displays a single SWF file, and then closes. The `unloadMovieNum()` function lets you affect several SWF files at once and switch among SWF files without loading another HTML document.

See also

`loadMovieNum()`

CHAPTER 2

Flash Lite Properties

2

This section describes the properties that Macromedia Flash Lite 1.x recognizes. The entries are listed alphabetically, ignoring any leading underscores. The properties are summarized in the following table:

Property	Description
`/ (Forward slash)`	Specifies or returns a reference to the main movie timeline.
`_alpha`	Returns the alpha transparency value of a movie clip.
`_currentframe`	Returns the number of the frame in which the playhead is located in the timeline.
`_focusrect`	Specifies whether a yellow rectangle appears around the button or text field that has the current focus.
`_framesloaded`	Returns the number of frames that have been loaded from a dynamically loaded SWF file.
`_height`	Specifies the height of the movie clip, in pixels.
`_highquality`	Specifies the level of anti-aliasing applied to the current SWF file.
`_level`	Returns a reference to the root timeline of `_levelN`. You must use the `loadMovieNum()` function to load SWF files into the Flash Lite player before you use the `_level` property to target them. You can also use `_levelN` to target a loaded SWF file at the level assigned by `N`.
`maxscroll`	Indicates the line number of the first visible line of text in a scrollable text field when the last line in the field is also visible.
`_name`	Returns the instance name of a movie clip. It applies only to movie clips and not to the main timeline.
`_rotation`	Returns the rotation of the movie clip, in degrees, from its original orientation.
`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.

Property	Description
_target	Returns the target path of the movie clip instance.
_totalframes	Returns the total number of frames in a movie clip.
_visible	Indicates whether a movie clip is visible.
_width	Returns the width of the movie clip, in pixels.
_x	Contains an integer that sets the *x* coordinate of a movie clip.
_xscale	Sets the horizontal scale (*percentage*) of the movie clip, as applied from the registration point of the movie clip.
_y	Contains an integer that sets the *y* coordinate of a movie clip, relative to the local coordinates of the parent movie clip.
_yscale	Sets the vertical scale (*percentage*) of the movie clip, as applied from the registration point of the movie clip.

/ (Forward slash)

Availability

Flash Lite 1.0

Usage

```
/

/targetPath

/:varName
```

Description

Identifier; specifies or returns a reference to the main movie timeline. The functionality provided by this property is similar to that provided by the `_root` property in Flash 5.

Example

To specify a variable on a timeline, use slash syntax (/) combined with the colon (:).

Example 1: The `car` variable on the main Timeline:

```
/:car
```

Example 2: The `car` variable in the movie clip instance mc1 that resides on the main Timeline:

```
/mc1/:car
```

Example 3: The `car` variable in the movie clip instance mc2 nested in the movie clip instance mc1 that resides on the main Timeline:

```
/mc1/mc2/:car
```

Example 4: The `car` variable in the movie clip instance mc2 that resides on the current Timeline:

```
mc2/:car
```

_alpha

Availability

Flash Lite 1.0.

Usage

```
my_mc:_alpha
```

Property; the alpha transparency value of the movie clip specified by the *my_mc* variable. Valid values are 0 (fully transparent) to 100 (fully opaque), which is the default value. Objects in a movie clip with `_alpha` set to 0 are active, even though they are invisible. For example, you can click a button in a movie clip whose `_alpha` property is set to 0.

Example

The following code for a button event handler sets the `_alpha` property of the `my_mc` movie clip to 30% when the user clicks the button:

```
on(release) {
  tellTarget("my_mc") {
    _alpha = 30;
  }
}
```

_currentframe

Availability

Flash Lite 1.0.

Usage

```
my_mc:_currentframe
```

Description

Property (read-only); returns the number of the frame in which the playhead is located in the timeline that the *my_mc* variable specifies.

Example

The following example uses the `_currentframe` property and the `gotoAndStop()` function to direct the playhead of the `my_mc` movie clip to advance five frames ahead of its current location:

```
tellTarget("my_mc") {
  gotoAndStop(_currentframe + 5);
}
```

See also

`gotoAndStop()`

_focusrect

Availability

Flash Lite 1.0.

Usage

```
_focusrect = Boolean;
```

Description

Property (global); specifies whether a yellow rectangle appears around the button or text field that has the current focus. The default value, `true`, displays a yellow rectangle around the currently focused button or text field as the user presses the Up or Down Arrow keys on their phone or mobile device to navigate through objects in a SWF file. Specify `false` if you do not want the yellow rectangle to appear.

Example

The following example disables the yellow focus rectangle from appearing in the application:

```
_focusrect = false;
```

_framesloaded

Availability

Flash Lite 1.0.

Usage

my_mc:_framesloaded

Description

Property (read-only); the number of frames that have been loaded from a dynamically loaded SWF file. 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 the browser. It is also useful as a monitor while large SWF files download. For example, you might want to display a message to users indicating that the SWF file is loading until a specified frame in the SWF file finishes loading.

Example

The following example uses the `_framesloaded` property to start a SWF file when all the frames are loaded. If all the frames aren't loaded, the `_xscale` property of the movie clip instance `loader` is increased proportionally to create a progress bar.

```
if (_framesloaded >= _totalframes) {
  gotoAndPlay ("Scene 1", "start");
} else {
  tellTarget(“loader”) {
    _xscale = (_framesloaded/_totalframes)*100;
  }
}
```

_height

Availability

Flash Lite 1.0.

Usage

```
my_mc:_height
```

Description

Property (read-only); the height of the movie clip, in pixels.

Example

The following example of event handler code sets the height of a movie clip when the user clicks the mouse button:

```
on(release) {
  tellTarget("my_mc") {
    _height = 200;
  }
}
```

_highquality

Availability

Flash Lite 1.0.

Usage

```
_highquality
```

Description

Property (global); specifies the level of anti-aliasing applied to the current SWF file. Specify 2 for best quality anti-aliasing. Specify 1 for high quality anti-aliasing. Specify 0 to prevent anti aliasing.

Example

The following statement applies high quality anti-aliasing to the current SWF file:

```
_highquality = 1;
```

See also

`toggleHighQuality()`

_level

Availability

Flash Lite 1.0.

Usage

`_levelN`

Description

Identifier; a reference to the root timeline of `_levelN`. You must use the `loadMovieNum()` function to load SWF files into the Flash Lite player before you use the `_level` property to target them. You can also use `_levelN` to target a loaded SWF file at the level assigned by *N*.

The initial SWF file that loads into an instance of the Flash Lite player automatically loads into `_level0`. The SWF file in `_level0` 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 `_level0`.

You must assign a level to each SWF file that you load into the Flash Lite player by using the `loadMovieNum()` function. You can assign levels in any order. If you assign a level that already contains a SWF file (including `_level0`), the SWF file at that level is unloaded and replaced by the new SWF file.

Example

The following example loads a SWF file into Level 1, and then stops the playhead of the loaded SWF file on Frame 6:

```
loadMovieNum("mySWF.swf", 1);

// at least 1 frame later
tellTarget(_level1) {
  gotoAndStop(6);
}
```

See also

`loadMovie()`

maxscroll

Availability

Flash Lite 1.1

Usage

variable_name:maxscroll

Description

Property (read-only); indicates the line number of the first visible line of text in a scrollable text field when the last line in the field is also visible. The `maxscroll` property works with the `scroll` property to control how information appears in a text field. This property can be retrieved but not modified.

Example

The following code, which scrolls the `myText` text field down one line when the user presses the 8 key, tests against `maxscroll` before scrolling:

```
on(keyPress "8") {
  if (myText:scroll < myText:maxscroll) {
    myText:scroll++;
  }
}
```

See also

scroll

_name

Availability

Flash Lite 1.0.

Usage

my_mc:_name

Description

Property; the instance name of the movie clip that *my_mc* specifies. It applies only to movie clips and not to the main timeline.

Example

The following example displays the name of the `bigRose` movie clip in the Output panel as a string:

```
trace(bigRose:_name);
```

_rotation

Availability

Flash Lite 1.0.

Usage

my_mc`:_rotation`

Description

Property; the rotation of the movie clip, in degrees, from its original orientation. Values from 0 to 180 represent clockwise rotation; values from 0 to -180 represent counterclockwise rotation. Values outside this range are added to or subtracted from 360 to obtain a value within the range. For example, the statement `my_mc:_rotation = 450` is the same as `my_mc:_rotation = 90`.

Example

The following example rotates the `my_mc` movie clip 15 degrees clockwise when the user presses the 2 key:

```
on (keyPress "2") {
  my_mc:_rotation += 15;
}
```

scroll

Availability

Flash Lite 1.1.

Usage

textFieldVariableName:scroll

Description

Property; 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. You can use the `scroll` property to create a scrolling text field or to direct a user to a specific paragraph in a long passage.

Example

The following code scrolls the `myText` text field up one line each time the user clicks the 2 key:

```
on(keyPress "2") {
  if (myText:scroll > 1) {
    myText:scroll--;
  }
}
```

See also

maxscroll

_target

Availability

Flash Lite 1.0.

Usage

my_mc:_target

Description

Property (read-only); returns the target path of the movie clip instance that *my_mc* specifies.

_totalframes

Availability

Flash Lite 1.0.

Usage

my_mc:_totalframes

Description

Property (read-only); returns the total number of frames in the *my_mc* movie clip.

Example

The following code loads mySWF.swf into Level 1, and then 25 frames later, checks to see whether it is loaded:

```
loadMovieNum("mySWF.swf", 1);

// 25 frames later in the main timeline
if (_level1._framesloaded >= _level1._totalframes) {
  tellTarget("_level1/") {
    gotoAndStop("myLabel");
  }
} else {
  // loop...
}
```

_visible

Availability

Flash Lite 1.0.

Usage

my_mc:_visible

Description

Property; a Boolean value that indicates whether the movie clip that *my_mc* specifies is visible. Movie clips that are not visible (_visible property set to false) are disabled. For example, a button in a movie clip with _visible set to false cannot be clicked. Movie clips are visible unless explicitly made invisible in this manner.

Example

The following code disables the `my_mc` movie clip when the user presses the 3 key, and enables it when the user presses the 4 key:

```
on(keyPress "3") {
  my_mc:_visible = 0;
 }

 on(keyPress "4") {
  my_mc:_visible = 1;
 }
```

_width

Availability

Flash Lite 1.0.

Usage

my_mc:_width

Description

Property; the width of the movie clip, in pixels.

Example

The following example sets the width properties of a movie clip when the user presses the 5 key:

```
on(keyPress "5") {
  my_mc:_width = 10;
}
```

_X

Availability

Flash Lite 1.0.

Usage

my_mc:_x

Description

Property; an integer that sets the *x* coordinate of a movie clip (represented here by *my_mc*), relative to the local coordinates of the parent movie clip. If a movie clip is in the main timeline, its coordinate system refers to the upper-left corner of the Stage as (0, 0).

If the movie clip is inside another movie clip that has transformations, the movie clip is in the local coordinate system of the enclosing movie clip. For example, if a movie clip is rotated 90 degrees counterclockwise, the child movie clips inherit a coordinate system that is rotated 90 degrees counterclockwise. The movie clip's coordinates refer to the registration point position.

Example

The following example changes the horizontal position of the `my_mc` movie clip when the user presses the 6 key:

```
on(keyPress "6") {
  my_mc:_x = 10;
}
```

See also

`_xscale`, `_y`, `_yscale`

_xscale

Availability

Flash Lite 1.0.

Usage

my_mc`:_xscale`

Description

Property; sets the horizontal scale (*percentage*) of the movie clip, as applied from the registration point of the movie clip. The default registration point is (0, 0).

Scaling the local coordinate system affects the `_x` and `_y` property settings, which are defined in pixels. For example, if the parent movie clip is scaled to 50%, setting the `_x` property moves an object in the movie clip by half of the number of pixels that it would if the movie were set at 100%.

Example

The following example changes the horizontal scale of the `my_mc` movie clip when the user presses the 7 key:

```
on(keyPress "7") {
  my_mc:_xscale = 10;
}
```

See also

_x, _y, _yscale

_y

Availability

Flash Lite 1.0.

Usage

my_mc:_y

Description

Property; an integer that sets the *y* coordinate of a movie clip (represented here by *my_mc*), relative to the local coordinates of the parent movie clip. If a movie clip is in the main Timeline, its coordinate system refers to the upper-left corner of the Stage as (0, 0).

If the move clip is inside another movie clip that has transformations, the movie clip is in the local coordinate system of the enclosing movie clip. For example, if a movie clip is rotated 90 degrees counterclockwise, the child movie clips inherit a coordinate system that is rotated 90 degrees counterclockwise. The movie clip's coordinates refer to the registration point position.

Example

The following code sets the *y* coordinates of the `my_mc` movie clip 10 pixels below the (0, 0) coordinate of is parent clip when the user presses the 1 key:

```
on(keyPress "1") {
  my_mc:_y = 10;
}
```

See also

_x, _xscale, _yscale

_yscale

Availability

Flash Lite 1.0.

Usage

my_mc:_yscale

Description

Property; sets the vertical scale (*percentage*) of the movie clip, as applied from the registration point of the movie clip. The default registration point is (0, 0).

Scaling the local coordinate system affects the _x and _y property settings, which are defined in pixels. For example, if the parent movie clip is scaled to 50%, setting the _y property moves an object in the movie clip by half the number of pixels as it would if the movie were set at 100%.

Example

The following example changes the vertical scale of the my_mc movie clip to 10% when the user presses the 1 key:

```
on(keyPress "1") {
  my_mc:_yscale = 10;
}
```

See also

_x, _xscale, _y

CHAPTER 3

Flash Lite Statements

3

This section describes the syntax and use of the Macromedia Flash Lite 1.x ActionScript statements, which are language elements that perform or specify an action. The statements are summarized in the following table:

Statement	Description
break	Instructs Flash Lite to skip the rest of the loop body, stop the looping action, and execute the statement following the loop statement.
case	Defines a condition for the `switch` statement. The statements in the *`statements`* parameter execute if the *`expression`* parameter that follows the `case` keyword equals the `expression` parameter of the `switch` statement.
continue	Jumps past all remaining statements in the innermost loop and starts the next iteration of the loop as if control had passed normally through to the end of the loop.
do..while	Executes the statements, and then evaluates the condition in a loop for as long as the condition is `true`.
else	Specifies the statements to run if the condition in the `if` statement evaluates to `false`.
else if	Evaluates a condition and specifies the statements to run if the condition in the initial `if` statement returns a `false` value.
for	Evaluates the *`init`* (initialize) expression once, and then begins a looping sequence by which, as long as the *`condition`* evaluates to `true`, *`statement`* is executed, and the next expression is evaluated.
if	Evaluates a condition to determine the next action in a SWF file. If the condition is `true`, Flash Lite runs the statements that follow the condition inside curly braces (`{}`). If the condition is `false`, Flash Lite skips the statements inside the curly braces and runs the statements following the braces.

Statement	Description
switch	Similar to the `if` statement, the `switch` statement tests a condition and executes statements if the condition evaluates to `true`.
while	Tests an expression and runs a statement or series of statements repeatedly in a loop as long as the expression is `true`.

break

Availability

Flash Lite 1.0.

Usage

```
break
```

Parameters

None.

Description

Statement; appears within a loop (`for`, `do..while` or `while`) or within a block of statements associated with a particular case within a `switch` statement. The `break` statement instructs Flash Lite to skip the rest of the loop body, stop the looping action, and execute the statement following the loop statement. When using the `break` statement, the ActionScript interpreter skips the rest of the statements in that `case` block and jumps to the first statement following the enclosing `switch` statement. Use this statement to break out of a series of nested loops.

Example

The following example uses the `break` statement to exit an otherwise infinite loop:

```
i = 0;
while (true) {
  if (i >= 100) {
    break;
  }
  i++;
}
```

See also

`case`, `do..while`, `for`, `switch`, `while`

case

Availability

Flash Lite 1.0.

Usage

```
case expression: statements
```

Parameters

expression Any expression.

statements Any statements.

Description

Statement; defines a condition for the `switch` statement. The statements in the *statements* parameter execute if the *expression* parameter that follows the `case` keyword equals the *expression* parameter of the `switch` statement.

If you use the `case` statement outside a `switch` statement, it produces an error and the code doesn't compile.

Example

In the following example, if the `myNum` parameter evaluates to 1, the `trace()` statement that follows `case 1` executes; if the `myNum` parameter evaluates to 2, the `trace()` statement that follows `case 2` executes; and so on. If no `case` expression matches the `number` parameter, the `trace()` statement that follows the `default` keyword executes.

```
switch (myNum) {
  case 1:
    trace ("case 1 tested true");
    break;
  case 2:
    trace ("case 2 tested true");
    break;
  case 3:
    trace ("case 3 tested true");
    break;
  default:
    trace ("no case tested true")
}
```

In the following example, no break occurs in the first case group, so if the number is 1, both `A` and `B` appear in the Output panel:

```
switch (myNum) {
  case 1:
    trace ("A");
  case 2:
    trace ("B");
    break;
  default:
    trace ("D")
}
```

See also

`switch`

continue

Availability

Flash Lite 1.0.

Usage

```
continue
```

Parameters

None.

Description

Statement; jumps past all remaining statements in the innermost loop and starts the next iteration of the loop as if control had passed through to the end of the loop normally. It has no effect outside a loop.

- In a `while` loop, `continue` causes the Flash interpreter to skip the rest of the loop body and jump to the top of the loop, where the condition is tested.
- In a `do..while` loop, `continue` causes the Flash interpreter to skip the rest of the loop body and jump to the bottom of the loop, where the condition is tested.
- In a `for` loop, `continue` causes the Flash interpreter to skip the rest of the loop body and jump to the evaluation of the `for` loop's post-expression.

Example

In the following `while` loop, `continue` causes Flash Lite to skip the rest of the loop body and jump to the top of the loop, where the condition is tested:

```
i = 0;
while (i < 10) {
  if (i % 3 == 0) {
    i++;
    continue;
  }
  trace(i);
  i++;
}
```

In the following `do..while` loop, `continue` causes Flash Lite to skip the rest of the loop body and jump to the bottom of the loop, where the condition is tested:

```
i = 0;
do {
  if (i % 3 == 0) {
    i++;
    continue;
  }
  trace(i);
  i++;
} while (i < 10);
```

In a `for` loop, `continue` causes Flash Lite to skip the rest of the loop body. In the following example, if `i` modulo 3 equals 0, the `trace(i)` statement is skipped:

```
for (i = 0; i < 10; i++) {
  if (i % 3 == 0) {
    continue;
  }
  trace(i);
}
```

See also

`do..while`, `for`, `while`

do..while

Availability

Flash Lite 1.0.

Usage

```
do {
  statement(s)
} while (condition)
```

Parameters

statement(s) The statement(s) to execute as long as the *condition* parameter evaluates to `true`.

condition The condition to evaluate.

Description

Statement; executes the statements, and then evaluates the condition in a loop for as long as the condition is `true`.

Example

The following example increments the index variable as long as the variable's value is less than 10:

```
i = 0;
 do {
  //trace (i);    // output: 0,1,2,3,4,5,6,7,8,9
  i++;
 } while (i<10);
```

See also

`break`, `continue`, `for`, `while`

else

Availability

Flash Lite 1.0.

Usage

```
if (condition){
   t-statement(s);
} else {
   f-statement(s);
}
```

Parameters

condition An expression that evaluates to `true` or `false`.

t-statement(s) The instructions to execute if the condition evaluates to `true`.

f-statement(s) An alternative series of instructions to execute if the condition evaluates to `false`.

Description

Statement; specifies the statements to run if the condition in the `if` statement evaluates to `false`.

Example

The following example shows the use of the `else` statement with a condition. An actual example would include code to take some action based on the condition.

```
currentHighestDepth = 1;
 if (currentHighestDepth == 2) {
  //trace ("currentHighestDepth is 2");
 } else {
  //trace ("currentHightestDepth is not 2");
 }
```

See also

`if`

else if

Availability

Flash Lite 1.0.

Usage

```
if (condition){
  statement(s);
} else if (condition){
  statement(s);
}
```

Parameters

condition An expression that evaluates to `true` or `false`.

statement(s) A series of statements to run if the condition specified in the `if` statement is `false`.

Description

Statement; evaluates a condition and specifies the statements to run if the condition in the initial `if` statement returns a `false` value. If the `else if` condition returns a `true` value, the Flash interpreter runs the statements that follow the `else if` condition inside curly braces ({}). If the `else if` condition is `false`, Flash skips the statements inside the curly braces and runs the statements following the curly braces. Use the `else if` statement to create branching logic in your scripts.

Example

The following example uses `else if` statements to check whether each side of an object is within a specific boundary:

```
person_mc.xPos = 100;
leftBound = 0;
rightBound = 100;

if (person_mc.xPos <= leftBound) {
  //trace ("Clip is to the far left");
} else if (person_mc.xPos >= rightBound) {
  //trace ("Clip is to the far right");
} else {
  //trace ("Your clip is somewhere in between");
}
```

See also

`if`

for

Availability

Flash Lite 1.0.

Usage

```
for (init; condition; next) {
  statement(s);
}
```

Parameters

init An expression to evaluate before beginning the looping sequence, typically an assignment expression.

condition An expression that evaluates to `true` or `false`. The condition is evaluated before each loop iteration; the loop exits when the condition evaluates to `false`.

next An expression to evaluate after each loop iteration; usually an assignment expression using the increment (++) or decrement (--) operator.

statement(s) One or more instructions to execute in the loop.

Description

Statement; a loop construct that evaluates the *init* (initialize) expression once and then begins a looping sequence by which, as long as the *condition* evaluates to `true`, *statement* is executed, and the next expression is evaluated.

Some properties cannot be enumerated by the `for` or `for..in` statements. For example, movie clip properties, such as `_x` and `_y`, are not enumerated.

Example

The following example uses the `for` loop to sum the numbers from 1 to 100:

```
sum = 0;
  for (i = 1; i <= 100; i++) {
    sum = sum + i;
  }
```

See also

++ (increment), -- (decrement), do..while, while

if

Availability

Flash Lite 1.0.

Usage

```
if (condition) {
  statement(s);
}
```

Parameters

condition An expression that evaluates to `true` or `false`.

statement(s) The instructions to execute if the condition evaluates to `true`.

Description

Statement; evaluates a condition to determine the next action in a SWF file. If the condition is `true`, Flash Lite runs the statements that follow the condition inside curly braces (`{}`). If the condition is `false`, Flash Lite skips the statements inside the curly braces and runs the statements following the braces. Use the `if` statement to create branching logic in your scripts.

Example

In the following example, the condition inside the parentheses evaluates the variable `name` to see if it has the literal value `"Erica"`. If it does, the `play()` function runs.

```
if(name eq "Erica"){
  play();
}
```

switch

Availability

Flash Lite 1.0.

Usage

```
switch (expression){
  caseClause:
  [defaultClause:]
}
```

Parameters

expression Any numeric expression.

caseClause A `case` keyword followed by an expression, a colon, and a group of statements to execute if the expression matches the switch *expression* parameter.

defaultClause An optional `default` keyword followed by statements to execute if none of the case expressions match the switch *expression* parameter.

Description

Statement; creates a branching structure for ActionScript statements. Similar to the `if` statement, the `switch` statement tests a condition and executes statements if the condition evaluates to `true`.

Switch statements contain a fallback option called default. If no other statements are true, the default statement is executed.

Example

In the following example, if the `myNum` parameter evaluates to 1, the `trace()` statement that follows `case 1` executes; if the `myNum` parameter evaluates to 2, the `trace()` statement that follows `case 2` executes; and so on. If no `case` expression matches the `number` parameter, the `trace()` statement that follows the `default` keyword executes.

```
switch (myNum) {
  case 1:
    trace ("case 1 tested true");
    break;
  case 2:
    trace ("case 2 tested true");
    break;
  case 3:
    trace ("case 3 tested true");
    break;
  default:
    trace ("no case tested true")
}
```

In the following example, the first case group doesn't contain a break, so if the number is 1, both A and B appear in the Output panel:

```
switch (myNum) {
  case 1:
    trace ("A");
  case 2:
    trace ("B");
    break;
  default:
    trace ("D")
}
```

See also

`case`

while

Availability

Flash Lite 1.0.

Usage

```
while(condition) {
  statement(s);
}
```

Parameters

condition The expression that is evaluated each time the while statement executes.

statement(s) The instructions to execute when the condition evaluates to true.

Description

Statement; tests an expression and runs a statement or series of statements repeatedly in a loop as long as the expression is true.

Before the statement block is run, the condition is tested; if the test returns true, the statement block is run. If the condition is false, the statement block is skipped and the first statement after the while statement's statement block is executed.

Looping is commonly used to perform an action when a counter variable is less than a specified value. At the end of each loop, the counter is incremented until the specified value is reached. At that point, the condition is no longer true, and the loop ends.

The while statement performs the following series of steps. Each repetition of steps 1 through 4 is called an *iteration* of the loop. The condition is tested at the beginning of each iteration:

1. The expression *condition* is evaluated.
2. If *condition* evaluates to true or a value that converts to the Boolean value true, such as a nonzero number, go to step 3.

   Otherwise, the while statement completes and execution resumes at the next statement after the while loop.
3. Run the statement block *statement(s)*.
4. Go to step 1.

Example

The following example executes a loop as long as the value of the index variable i is less than 10:

```
i = 0;
while(i < 10) {
  trace ("i = " add ++i);// Output: 1,2,3,4,5,6,7,8,9
}
```

See also

continue, do..while, for

CHAPTER 4

Flash Lite Operators

4

This section describes the syntax and use of the Macromedia Flash Lite 1.x ActionScript operators. All entries are listed alphabetically. However, some operators are symbols and are alphabetized by their text descriptions.

The operators in this section are summarized in the following table:

Operator	Description
add (string concatenation)	Concatenates (combines) two or more strings.
+= (addition assignment)	Assigns *expression1* the value of *expression1* + *expression2*.
and	Performs a logical AND operation.
= (assignment)	Assigns the value of *expression2* (the operand on the right) to the variable or property in *expression1*.
/* (block comment)	Indicates one or more lines of script comments. Any characters that appear between the opening comment tag (/*) and the closing comment tag (*/) are interpreted as a comment and ignored by the ActionScript interpreter.
, (comma)	Evaluates *expression1*, then *expression2*, and returns the value of *expression2*.
// (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 and ignored by the ActionScript interpreter.
?: (conditional)	Instructs Flash Lite to evaluate *expression1*, and if the value of *expression1* is `true`, the operator returns the value of *expression2*; otherwise, it returns the value of *expression3*.

Operator	Description
-- (decrement)	Subtracts 1 from *expression*. The pre-decrement form of the operator (*--expression*) subtracts 1 from *expression* and returns the result as a number. The post-decrement form of the operator (*expression--*) subtracts 1 from *expression* and returns the initial value of *expression* (the value before the subtraction).
/ (divide)	Divides *expression1* by *expression2*.
/= (division assignment)	Assigns *expression1* the value of *expression1 / expression2*.
. (dot)	Used to navigate movie clip hierarchies to access nested (child) movie clips, variables, or properties.
++ (increment)	Adds 1 to *expression*. The expression can be a variable, element in an array, or property of an object. The pre-increment form of the operator (*++expression*) adds 1 to *expression* and returns the result as a number. The post-increment form of the operator (*expression*++) adds 1 to *expression* and returns the initial value of *expression* (the value before the addition).
&& (logical AND)	Evaluates *expression1* (the expression on the left side of the operator) and returns `false` if the expression evaluates to `false`. If *expression1* evaluates to `true`, *expression2* (the expression on the right side of the operator) is evaluated. If *expression2* evaluates to `true`, the final result is `true`; otherwise, it is `false`.
! (logical NOT)	Inverts the Boolean value of a variable or expression. If *expression* is a variable with the absolute or converted value of `true`, the value of !*expression* is `false`. If the expression `x && y` evaluates to `false`, the expression `!(x && y)` evaluates to `true`.
|| (logical OR)	Evaluates *expression1* and *expression2*. The result is `true` if either or both expressions evaluate to `true`; the result is `false` only if both expressions evaluate to `false`. You can use the logical OR operator with any number of operands; if any operand evaluates to `true`, the result is `true`.
% (modulo)	Calculates the remainder of *expression1* divided by *expression2*. If an *expression* operand is non-numeric, the modulo operator attempts to convert it to a number.
%= (modulo assignment)	Assigns *expression1* the value of *expression1 % expression2*.
*= (multiplication assignment)	Assigns *expression1* the value of *expression1 * expression2*.
* (multiply)	Multiples two numeric expressions.
+ (numeric add)	Adds numeric expressions.

Operator	Description
== (numeric equality)	Tests for equality; if *expression1* is equal to *expression2*, the result is `true`.
> (numeric greater than)	Compares two expressions and determines whether *expression1* is greater than *expression2*; if it is, the operator returns *true*. If *expression1* is less than or equal to *expression2*, the operator returns `false`.
>= (numeric greater than or equal to)	Compares two expressions and determines whether *expression1* is greater than or equal to *expression2* `(true)` or whether *expression1* is less than *expression2* `(false)`.
<> (numeric inequality)	Tests for inequality; if *expression1* is equal to *expression2*, the result is `false`.
< (numeric less than)	Compares two expressions and determines whether *expression1* is less than *expression2*; if so, the operator returns `true`. If *expression1* is greater than or equal to *expression2*, the operator returns `false`.
<= (numeric less than or equal to)	Compares two expressions and determines whether *expression1* is less than or equal to *expression2*. If it is, the operator returns `true`; otherwise, it returns `false`.
() (parentheses)	Groups one or more parameters, performs sequential evaluation of expressions, or surrounds one or more parameters and passes them as parameters to a function outside the parentheses.
" " (string delimiter)	When used before and after a sequence of zero or more characters, quotation marks indicate that the characters have a literal value and are considered a *string*; they are not a variable, numeric value, or other ActionScript element.
eq (string equality)	Compares two expressions for equality and returns `true` if the string representation of *expression1* is equal to the string representation of *expression2*; otherwise, the operation returns `false`.
gt (string greater than)	Compares the string representation of *expression1* to the string representation of *expression2* and returns `true` if *expression1* is greater than *expression2*; otherwise, it returns `false`.
ge (string greater than or equal to)	Compares the string representation of *expression1* to the string representation of *expression2* and returns a `true` value if *expression1* is greater than or equal to *expression2*; otherwise, it returns `false`.
ne (string inequality)	Compares the string representations of *expression1* to *expression2* and returns `true` if *expression1* is not equal to *expression2*; otherwise, it returns `false`.

Operator	Description
lt (string less than)	Compares the string representation of *expression1* to the string representation of *expression2* and returns a `true` value if *expression1* is less than *expression2*; otherwise, it returns `false`.
le (string less than or equal to)	Compares the string representation of *expression1* to the string representation of *expression2* and returns a `true` value if *expression1* is less than or equal to *expression2*; otherwise, it returns `false`.
- (subtract)	Used for negating or subtracting.
-= (subtraction assignment)	Assigns *expression1* the value of *expression1 - expression2*.

add (string concatenation)

Availability

Flash Lite 1.0.

Usage

string1 add *string2*

Operands

string1, string2 Strings.

Description

Operator; concatenates (combines) two or more strings.

Example

The following example combines two string values to produce the string *catalog*.

```
conStr = "cat" add "alog";
trace (conStr);// output: catalog
```

See also

+ (numeric add)

+= (addition assignment)

Availability

Flash Lite 1.0.

Usage

expression1 += *expression2*

Operands

expression1, expression2 Numbers or strings.

Description

Operator (arithmetic compound assignment); assigns *expression1* the value of *expression1* + *expression2*. For example, the following two statements have the same result:

```
x += y;
x = x + y;
```

All the rules of the addition (+) operator apply to the addition assignment (+=) operator.

Example

The following example uses the addition assignment (+=) operator to increase the value of `x` by the value of `y`:

```
x = 5;
y = 10;
x += y;
trace(x);// output: 15
```

See also

+ (numeric add)

and

Availability

Flash Lite 1.0.

Usage

condition1 and *condition2*

Operands

condition1, condition2 Conditions or expressions that evaluate to `true` or `false`.

Description

Operator; performs a logical AND operation.

Example

The following example uses the `and` operator to test whether a player has won the game. The `turns` variable and the `score` variable are updated when a player takes a turn or scores points during the game. The following script shows "You Win the Game!" in the Output panel when the player's score reaches 75 or higher in three turns or less.

```
turns = 2;
score = 77;
winner = (turns <= 3) and (score >= 75);
if (winner) {
  trace("You Win the Game!");
} else {
  trace("Try Again!");
}
// output: You Win the Game!
```

See also

`&& (logical AND)`

= (assignment)

Availability

Flash Lite 1.0.

Usage

`expression1 = expression2`

Operands

`expression1` A variable or a property.

`expression2` A value.

Description

Operator; assigns the value of *`expression2`* (the operand on the right) to the variable or property in *`expression1`*.

Example

The following example uses the assignment (=) operator to assign a numeric value to the variable `weight`:

```
weight = 5;
```

The following example uses the assignment (=) operator to assign a string value to the variable `greeting`:

```
greeting = "Hello, " and personName;
```

/* (block comment)

Availability

Flash Lite 1.0

Usage

```
/* comment */
/* comment
comment */
```

Operands

comment Any characters.

Description

Comment delimiter; indicates one or more lines of script comments. Any characters that appear between the opening comment tag (/*) and the closing comment tag (*/) are interpreted as a comment and ignored by the ActionScript interpreter.

Use the // (comment delimiter) to identify single-line comments. Use the /* comment delimiter to identify comments on multiple successive lines. Leaving off the closing tag (*/) when using this form of comment delimiter returns an error message. Attempting to nest comments also returns an error message.

After you use an opening comment tag (/*), the first closing comment tag (*/) will end the comment, regardless of the number of opening comment tags (/*) placed between them.

See also

`// (comment)`

, (comma)

Availability

Flash Lite 1.0.

Usage

expression1, expression2

Operands

expression1, expression2 Numbers or expressions that evaluate to numbers.

Description

Operator; evaluates *expression1*, then *expression2*, and returns the value of *expression2*.

Example

The following example uses the comma (,) operator without the parentheses `()` operator and illustrates that the comma operator returns only the value of the first expression without the parentheses `()` operator:

```
v = 0;
v = 4, 5, 6;
trace(v); // output: 4
```

The following example uses the comma (,) operator with the parentheses `()` operator and illustrates that the comma operator returns the value of the last expression when used with the parentheses `()` operator:

```
v = 0;
v = (4, 5, 6);
trace(v); // output: 6
```

The following example uses the comma (,) operator without the parentheses `()` operator and illustrates that the comma operator sequentially evaluates all of the expressions but returns the value of the first expression. The second expression, `z++`, is evaluated and `z` is incremented by 1.

```
v = 0;
z = 0;
v = v + 4 , z++, v + 6;
trace(v); // output: 4
trace(z); // output: 1
```

The following example is identical to the previous example except for the addition of the parentheses `()` operator and illustrates once again that when used with the parentheses `()` operator, the comma (`,`) operator returns the value of the last expression in the series:

```
v = 0;
z = 0;
v = (v + 4, z++, v + 6);
trace(v); // output: 6
trace(z); // output: 1
```

See also

`for`, `() (parentheses)`

// (comment)

Availability

Flash Lite 1.0

Usage

```
// comment
```

Operands

comment Any characters.

Description

Comment delimiter; 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 and ignored by the ActionScript interpreter.

Example

The following example uses comment delimiters to identify the first, third, fifth, and seventh lines as comments:

```
// Record the X position of the ball movie clip.
ballX = ball._x;
// Record the Y position of the ball movie clip.
ballY = ball._y;
// Record the X position of the bat movie clip.
batX = bat._x;
// Record the Y position of the bat movie clip.
batY = bat._y;
```

See also

`/* (block comment)`

?: (conditional)

Availability

Flash Lite 1.0.

Usage

expression1 ? *expression2* : *expression3*

Operands

expression1 An expression that evaluates to a Boolean value, usually a comparison expression, such as `x < 5`.

expression2, *expression3* Values of any type.

Description

Operator; instructs Flash Lite to evaluate *expression1*, and if the value of *expression1* is `true`, it returns the value of *expression2*; otherwise, it returns the value of *expression3*.

Example

The following example assigns the value of variable `x` to variable `z` because *expression1* evaluates to `true`:

```
x = 5;
y = 10;
z = (x < 6) ? x: y;
trace (z);// output: 5
```

— (decrement)

Availability

Flash Lite 1.0.

Usage

--expression

expression--

Operands

None.

Description

Operator (arithmetic); a pre-decrement and post-decrement unary operator that subtracts 1 from *expression*. The pre-decrement form of the operator (*--expression*) subtracts 1 from *expression* and returns the result as a number. The post-decrement form of the operator (*expression--*) subtracts 1 from *expression* and returns the initial value of *expression* (the value before the subtraction).

Example

The following example shows the pre-decrement form of the operator, decrementing `aWidth` to 2 (`aWidth - 1 = 2`) and returning the result as `bWidth`:

```
aWidth = 3;
bWidth = --aWidth;
// The bWidth value is equal to 2.
```

The next example shows the post-decrement form of the operator decrementing `aWidth` to 2 (`aWidth - 1 = 2`) and returning the original value of `aWidth` as the result `bWidth`:

```
aWidth = 3;
bWidth = aWidth--;
// The bWidth value is equal to 3.
```

/ (divide)

Availability

Flash Lite 1.0.

Usage

expression1 / expression2

Operands

expression1, expression2 Numbers or expressions that evaluate to numbers.

Description

Operator (arithmetic); divides *expression1* by *expression2*. The result of the division operation is a double-precision floating-point number.

Example

The following statement divides the floating-point number 22.0 by 7.0 and then shows the result in the Output panel:

```
trace(22.0 / 7.0);
```

The result is 3.1429, which is a floating-point number.

/= (division assignment)

Availability

Flash Lite 1.0.

Usage

expression1 /= expression2

Operands

expression1, expression2 Numbers or expressions that evaluate to numbers.

Description

Operator (arithmetic compound assignment); assigns *expression1* the value of *expression1 / expression2*. For example, the following two statements are equivalent:

```
x /= y
x = x / y
```

Example

The following example uses the /= operator with variables and numbers:

```
x = 10;
y = 2;
x /= y;
// The expression x now contains the value 5.
```

. (dot)

Availability

Flash Lite 1.0.

Usage

instancename.variable

instancename.childinstance.variable

Operands

instancename The instance name of a movie clip.

childinstance A movie clip instance that is a child of, or nested in, another movie clip.

variable A variable on the timeline of the specified movie clip instance name.

Description

Operator; used to navigate movie clip hierarchies to access nested (child) movie clips, variables, or properties.

Example

The following example identifies the current value of the variable `hairColor` in the movie clip `person_mc`:

```
person_mc.hairColor
```

This is equivalent to the following slash notation syntax:

```
/person_mc:hairColor
```

See also

/ (Forward slash)

++ (increment)

Availability

Flash Lite 1.0.

Usage

```
++expression

expression++
```

Operands

None.

Description

Operator (arithmetic); a pre-increment and post-increment unary operator that adds 1 to *expression*. The expression can be a variable, element in an array, or property of an object. The pre-increment form of the operator (++*expression*) adds 1 to *expression* and returns the result as a number. The post-increment form of the operator (*expression*++) adds 1 to *expression* and returns the initial value of *expression* (the value before the addition).

Example

The following example uses ++ as a post-increment operator to make a `while` loop run five times:

```
i = 0;
while (i++ < 5){
  trace("this is execution " + i);
}
```

The following example uses ++ as a pre-increment operator:

```
a = "";
i = 0;
while (i < 10) {
  a = a add (++i) add ",";
}
trace(a);// output: 1,2,3,4,5,6,7,8,9,10,
```

This script shows the following result in the Output panel:

```
1,2,3,4,5,6,7,8,9,10,
```

The following example uses ++ as a post-increment operator:

```
a = "";
i = 0;
while (i < 10) {
  a = a add (i++) add ",";
}
trace(a);// output: 0,1,2,3,4,5,6,7,8,9,
```

This script shows the following result in the Output panel:

```
0,1,2,3,4,5,6,7,8,9,
```

&& (logical AND)

Availability

Flash Lite 1.0.

Usage

expression1 && *expression2*

Operands

expression1, expression2 Boolean values or expressions that convert to Boolean values.

Description

Operator (logical); performs a Boolean operation on the values of one or both of the expressions. The operator evaluates *expression1* (the expression on the left side of the operator) and returns `false` if the expression evaluates to `false`. If *expression1* evaluates to `true`, *expression2* (the expression on the right side of the operator) is evaluated. If *expression2* evaluates to `true`, the final result is `true`; otherwise, it is `false`.

Example

The following example uses the `&&` operator to perform a test to determine if a player has won the game. The `turns` variable and the `score` variable are updated when a player takes a turn or scores points during the game. The following script shows "You Win the Game!" in the Output panel when the player's score reaches 75 or higher in three turns or less.

```
turns = 2;
score = 77;
winner = (turns <= 3) && (score >= 75);
if (winner) {
  trace("You Win the Game!");
} else {
  trace("Try Again!");
}
```

The following example demonstrates testing to see if an imaginary `x` position is in between a range:

```
xPos = 50;
if (xPos >= 20 && xPos <= 80) {
  trace ("the xPos is in between 20 and 80");
}
```

! (logical NOT)

Availability

Flash Lite 1.0.

Usage

`!expression`

Operands

None.

Description

Operator (logical); inverts the Boolean value of a variable or expression. If *expression* is a variable with the absolute or converted value of `true`, the value of `!expression` is `false`. If the expression `x && y` evaluates to `false`, the expression `!(x && y)` evaluates to `true`.

The following expressions show the result of using the `!` operator:

`!true` returns `false`

`!false` returns `true`

Example

In the following example, the variable `happy` is set to `false`. The `if` condition evaluates the condition `!happy`, and if the condition is `true`, the `trace()` function sends a string to the Output panel.

```
happy = false;
if (!happy) {
  trace("don't worry, be happy");
}
```

|| (logical OR)

Availability

Flash Lite 1.0.

Usage

`expression1 || expression2`

Operands

expression1, expression2 Boolean values or expressions that convert to Boolean values.

Description

Operator (logical); evaluates *expression1* and *expression2*. The result is `true` if either or both expressions evaluate to `true`; the result is `false` only if both expressions evaluate to `false`. You can use the logical OR operator with any number of operands; if any operand evaluates to `true`, the result is `true`.

With non-Boolean expressions, the logical OR operator causes Flash Lite to evaluate the expression on the left; if it can be converted to `true`, the result is `true`. Otherwise, it evaluates the expression on the right, and the result is the value of that expression.

Example

Usage 1: The following example uses the `||` operator in an `if` statement. The second expression evaluates to `true`, so the final result is `true`:

```
theMinimum = 10;
theMaximum = 250;
start = false;
if (theMinimum > 25 || theMaximum > 200 || start){
  trace("the logical OR test passed");
}
```

% (modulo)

Availability

Flash Lite 1.0.

Usage

expression1 `%` *expression2*

Operands

expression1, expression2 Numbers or expressions that evaluate to numbers.

Description

Operator (arithmetic); calculates the remainder of *expression1* divided by *expression2*. If an *expression* operand is non-numeric, the modulo operator attempts to convert it to a number. The expression can be a number or string that converts to a numeric value.

When targeting Flash Lite 1.0 or 1.1, the Flash compiler expands the `%` operator in the published SWF file by using the following formula:

expression1 `-` `int(`*expression1*`/`*expression2*`)` `*` *expression2*

The performance of this approximation might not be as fast or as accurate as versions of Flash Player that natively support the modulo operator.

Example

The following code shows a numeric example that uses the modulo (%) operator:

```
trace (12 % 5);// output: 2
trace (4.3 % 2.1);// output: 0.0999...
```

%= (modulo assignment)

Availability

Flash Lite 1.0.

Usage

expression1 %= expression2

Operands

expression1, expression2 Numbers or expressions that evaluate to numbers.

Description

Operator (arithmetic compound assignment); assigns *expression1* the value of *expression1 % expression2*. For example, the following two expressions are equivalent:

```
x %= y
x = x % y
```

Example

The following example assigns the value 4 to the variable x:

```
x = 14;
y = 5;
trace(x %= y);// output: 4
```

See also

% (modulo)

*= (multiplication assignment)

Availability

Flash Lite 1.0.

Usage

*expression1 *= expression2*

Operands

expression1, expression2 Numbers or expressions that evaluate to numbers.

Description

Operator (arithmetic compound assignment); assigns *expression1* the value of *expression1 * expression2*.

For example, the following two expressions are the same:

```
x *= y
x = x * y
```

Example

Usage 1: The following example assigns the value 50 to the variable `x`:

```
x = 5;
y = 10;
trace (x *= y);// output: 50
```

Usage 2: The second and third lines of the following example calculate the expressions on the right side of the equals sign (=) and assign the results to `x` and `y`:

```
i = 5;
x = 4 - 6;
y = i + 2;
trace(x *= y);// output: -14
```

* (multiply)

Availability

Flash Lite 1.0.

Usage

*expression1 * expression2*

Operands

expression1, expression2 Numeric expressions.

Description

Operator (arithmetic); multiplies two numeric expressions. If both expressions are integers, the product is an integer. If either or both expressions are floating-point numbers, the product is a floating-point number.

Example

Usage 1: The following statement multiplies the integers 2 and 3:

```
2 * 3
```

The result is 6, which is an integer.

Usage 2: The following statement multiplies the floating-point numbers 2.0 and 3.1416:

```
2.0 * 3.1416
```

The result is 6.2832, which is a floating-point number.

+ (numeric add)

Availability

Flash Lite 1.0.

Usage

expression1 + *expression2*

Operands

expression1, expression2 Numbers.

Description

Operator; adds numeric expressions. The + is a numeric operator only; it cannot be used for string concatenation.

If both expressions are integers, the sum is an integer; if either or both expressions are floating-point numbers, the sum is a floating-point number.

Example

The following example adds the integers 2 and 3; the resulting integer, 5, appears in the Output panel:

```
trace (2 + 3);
```

The following example adds the floating-point numbers 2.5 and 3.25; the result, 5.75, a floating-point number, appears in the Output panel:

```
trace (2.5 + 3.25);
```

See also

add (string concatenation)

== (numeric equality)

Availability

Flash Lite 1.0.

Usage

expression1 == *expression2*

Operands

expression1, expression2 Numbers, Boolean values, or variables.

Description

Operator (comparison); tests for equality; the exact opposite of the <> operator. If *expression1* is equal to *expression2*, the result is `true`. As with the <> operator, the definition of *equal* depends on the data types being compared:

- Numbers and Boolean values are compared by value.
- Variables are compared by reference.

Example

The following examples show `true` and `false` return values:

```
trees = 7;
bushes = "7";
shrubs = "seven";

trace (trees == "7");// output: 1(true)
trace (trees == bushes);// output: 1(true)
trace (trees == shrubs);// output: 0(false)
```

See also

eq (string equality)

> (numeric greater than)

Availability

Flash Lite 1.0.

Usage

expression1 > *expression2*

Operands

expression1, expression2 Numbers or expressions that evaluate to numbers.

Operator (comparison); compares two expressions and determines whether *expression1* is greater than *expression2*; if it is, the operator returns `true`. If *expression1* is less than or equal to *expression2*, the operator returns `false`.

Example

The following examples show `true` and `false` results for numeric comparisons:

```
trace(3.14 > 2);// output: 1(true)
trace(1 > 2);// output: 0(false)
```

See also

gt (string greater than)

>= (numeric greater than or equal to)

Availability

Flash Lite 1.0.

Usage

expression1 >= *expression2*

Operands

expression1, expression2 Integers or floating-point numbers.

Description

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`).

Example

The following examples show `true` and `false` results:

```
trace(3.14 >= 2);// output: 1(true)
trace(3.14 >= 4);// output: 0(false)
```

See also

`ge (string greater than or equal to)`

<> (numeric inequality)

Availability

Flash Lite 1.0.

Usage

`expression1 <> expression2`

Operands

`expression1, expression2` Numbers, Boolean values, or variables.

Description

Operator (comparison); tests for inequality; the exact opposite of the equality (==) operator. If *`expression1`* is equal to *`expression2`*, the result is `false`. As with the equality (==) operator, the definition of *equal* depends on the data types being compared:

- Numbers and Boolean values are compared by value.
- Variables are compared by reference.

Example

The following examples show `true` and `false` returns:

```
trees = 7;
B = "7";

trace(trees <> 3);// output: 1(true)
trace(trees <> B);// output: 0(false)
```

See also

`ne (string inequality)`

< (numeric less than)

Availability

Flash Lite 1.0.

Usage

expression1 < *expression2*

Operands

expression1, expression2 Numbers.

Description

Operator (comparison); compares two expressions and determines whether *expression1* is less than *expression2*; if so, the operator returns `true`. If *expression1* is greater than or equal to *expression2*, the operator returns `false`. The < (less than) operator is a numeric operator.

Example

The following examples show `true` and `false` results for both numeric and string comparisons:

```
trace (3 < 10);// output: 1(true)

trace (10 < 3);// output: 0(false)
```

See also

lt (string less than)

<= (numeric less than or equal to)

Flash Lite 1.0.

Usage

expression1 <= *expression2*

Operands

expression1, expression2 Numbers.

Description

Operator (comparison); compares two expressions and determines whether *expression1* is less than or equal to *expression2*. If it is, the operator returns `true`; otherwise, it returns `false`. This operator is for numeric comparison only.

Example

The following examples show `true` and `false` results for numeric comparisons:

```
trace(5 <= 10);// output: 1(true)
trace(2 <= 2);// output: 1(true)
trace (10 <= 3);// output: 0 (false)
```

See also

`le (string less than or equal to)`

() (parentheses)

Availability

Flash Lite 1.0.

Usage

```
(expression1 [, expression2])
(expression1, expression2)
```

`expression1, expression2` Numbers, strings, variables, or text.

`parameter1,..., parameterN` A series of parameters to execute before the results are passed as parameters to the function outside the parentheses.

Description

Operator; groups one or more parameters, performs sequential evaluation of expressions, or surrounds one or more parameters and passes them as parameters to a function outside the parentheses.

Usage 1: Controls the order in which the operators execute in the expression. Parentheses override the normal precedence order and cause the expressions within the parentheses to be evaluated first. When parentheses are nested, the contents of the innermost parentheses are evaluated before the contents of the outer ones.

Usage 2: Evaluates a series of expressions, separated by commas, in sequence, and returns the result of the final expression.

Example

Usage 1: The following statements show the use of parentheses to control the order in which expressions are executed (the value of each expression appears in the Output panel):

```
trace((2 + 3) * (4 + 5)); // displays 45
trace(2 + (3 * (4 + 5))); // // displays 29
trace(2 + (3 * 4) + 5); // displays 19
```

Usage 1: The following statements show the use of parentheses to control the order in which expressions are executed (the value of each expression is written to the log file):

```
trace((2 + 3) * (4 + 5)); // writes 45
trace(2 + (3 * (4 + 5))); // writes 29
trace(2 + (3 * 4) + 5); // writes 19
```

" " (string delimiter)

Availability

Flash Lite 1.0.

Usage

"*text*"

Operands

text Zero or more characters.

Description

String delimiter; when used before and after a sequence of zero or more characters, quotation marks indicate that the characters have a literal value and are considered a *string*; they are not a variable, numeric value, or other ActionScript element.

Example

This example uses quotation marks to indicate that the value of the variable `yourGuess` is the literal string `"Prince Edward Island"` and not the name of a variable. The value of `province` is a variable, not a literal; to determine the value of `province`, the value of `yourGuess` must be located.

```
yourGuess = "Prince Edward Island";

on(release){
  province = yourGuess;
  trace(province);// output: Prince Edward Island
}
```

eq (string equality)

Availability

Flash Lite 1.0.

Usage

expression1 eq *expression2*

Operands

expression1, expression2 Numbers, strings, or variables.

Description

Comparison operator; compares two expressions for equality and returns `true` if the string representation of *expression1* is equal to the string representation of *expression2*; otherwise, the operation returns `false`.

Example

The following examples show `true` and `false` results:

```
word = "persons";
figure = "55";

trace("persons" eq "people");// output: 0(false)
trace("persons" eq word);// output: 1(true)
trace(figure eq 50 + 5);// output: 1(true)
trace(55.0 eq 55);// output: 1(true)
```

See also

== (numeric equality)

gt (string greater than)

Availability

Flash Lite 1.0.

Usage

expression1 gt *expression2*

Operands

expression1, expression2 Numbers, strings, or variables.

Description

Operator (comparison); compares the string representation of *expression1* to the string representation of *expression2* and returns a `true` value if *expression1* is greater than *expression2*; otherwise, it returns a `false` value. Strings are compared using alphabetical order; digits precede all letters, and all capital letters precede lowercase letters.

Example

The following examples show `true` and `false` results:

```
animals = "cats";
breeds = 7;

trace ("persons" gt "people");// output: 1(true)
trace ("cats" gt "cattle");// output: 0(false)
trace (animals gt "cats");// output: 0(false)
trace (animals gt "Cats");// output: 1(true)
trace (breeds gt "5");// output: 1(true)
trace (breeds gt 7);// output: 0(false)
```

See also

> (numeric greater than)

ge (string greater than or equal to)

Availability

Flash Lite 1.0.

Usage

expression1 `ge` *expression2*

Operands

expression1, *expression2* Numbers, strings, or variables.

Description

Operator (comparison); compares the string representation of *expression1* to the string representation of *expression2* and returns a `true` value if *expression1* is greater than or equal to *expression2*; otherwise, it returns a `false` value. Strings are compared using alphabetical order; digits precede all letters, and all capital letters precede lowercase letters.

Example

The following examples show `true` and `false` results:

```
animals = "cats";
breeds = 7;

trace ("cats" ge "cattle");// output: 0(false)
trace (animals ge "cats");// output: 1(true)
trace ("persons" ge "people");// output: 1(true)
trace (animals ge "Cats");// output: 1(true)
trace (breeds ge "5");// output: 1(true)
trace (breeds ge 7);// output: 1(true)
```

See also

`>= (numeric greater than or equal to)`

ne (string inequality)

Availability

Flash Lite 1.0.

Usage

`expression1` `ne` *`expression2`*

Operands

`expression1, expression2` Numbers, strings, or variables.

Description

Operator (comparison); compares the string representations of *`expression1`* to *`expression2`* and returns `true` if *`expression1`* is not equal to *`expression2`*; otherwise, it returns `false`.

Example

The following examples show `true` and `false` results:

```
word = "persons";
figure = "55";

trace ("persons" ne "people");// output: 1(true)
trace ("persons" ne word);// output: 0(false)
trace (figure ne 50 + 5);// output: 0(false)
trace (55.0 ne 55); // output: 0(false)
```

See also

`<> (numeric inequality)`

lt (string less than)

Availability

Flash Lite 1.0.

Usage

expression1 `lt` *expression2*

Operands

expression1, *expression2* Numbers, strings, or variables.

Description

Operator (comparison); compares the string representation of *expression1* to the string representation of *expression2* and returns a `true` value if *expression1* is less than *expression2*; otherwise, it returns a `false` value. Strings are compared using alphabetical order; digits precede all letters, and all capital letters precede lowercase letters.

Example

The following examples show the output of various string comparisons. In the last line, notice that `lt` does not return an error when you compare a string to an integer because ActionScript 1.0 syntax tries to convert the integer data type to a string and returns `false`.

```
animals = "cats";
breeds = 7;

trace ("persons" lt "people");// output: 0(false)
trace ("cats" lt "cattle");// output: 1(true)
trace (animals lt "cats");// output: 0(false)
trace (animals lt "Cats");// output: 0(false)
trace (breeds lt "5");// output: 0(false)
trace (breeds lt 7);// output: 0(false)
```

See also

`< (numeric less than)`

le (string less than or equal to)

Availability

Flash Lite 1.0.

Usage

expression1 `le` *expression2*

Operands

expression1, expression2 Numbers, strings, or variables.

Description

Operator (comparison); compares the string representation of *expression1* to the string representation of *expression2* and returns a `true` value if *expression1* is less than or equal to *expression2*; otherwise, it returns a `false` value. Strings are compared using alphabetical order; digits precede all letters, and all capital letters precede lowercase letters.

Example

The following examples show the output of various string comparisons:

```
animals = "cats";
breeds = 7;

trace ("persons" le "people");// output: 0(false)
trace ("cats" le "cattle");// output: 1(true)
trace (animals le "cats");// output: 1(true)
trace (animals le "Cats");// output: 0(false)
trace (breeds le "5");// output: 0(false)
trace (breeds le 7);// output: 1(true)
```

See also

<= (numeric less than or equal to)

– (subtract)

Availability

Flash Lite 1.0.

Usage

(Negation) *-expression*

(Subtraction) *expression1 - expression2*

Operands

expression1, expression2 Numbers or expressions that evaluate to numbers.

Description

Operator (arithmetic); used for negating or subtracting.

Usage 1: When used for negating, it reverses the sign of the numeric expression.

Usage 2: When used for subtracting, it performs an arithmetic subtraction on two numeric expressions, subtracting *expression2* from *expression1*. When both expressions are integers, the difference is an integer. When either or both expressions are floating-point numbers, the difference is a floating-point number.

Example

Usage 1: The following statement reverses the sign of the expression 2 + 3:

```
trace(-(2 + 3));
// output: -5.
```

Usage 2: The following statement subtracts the integer 2 from the integer 5:

```
trace(5 - 2);
// output: 3.
```

The result, 3, is an integer.

Usage 3: The following statement subtracts the floating-point number 1.5 from the floating-point number 3.25:

```
trace(3.25 - 1.5);
// output: 1.75.
```

The result, 1.75, is a floating-point number.

-= (subtraction assignment)

Availability

Flash Lite 1.0.

Usage

expression1 -= expression2

Operands

expression1, expression2 Numbers or expressions that evaluate to numbers.

Description

Operator (arithmetic compound assignment); assigns *expression1* the value of *expression1 - expression2*. No value is returned.

For example, the following two statements are the same:

```
x -= y;
x = x - y;
```

String expressions must be converted to numbers; otherwise, -1 is returned.

Example

Usage 1: The following example uses the -= operator to subtract 10 from 5 and assign the result to the variable x:

```
x = 2;
y = 3;
x -= y
trace(x);// output: -1
```

Usage 2: The following example shows how strings are converted to numbers:

```
x = "2";
y = "5";
x -= y;
trace(x);// output: -3
```

CHAPTER 5

Flash Lite Specific Language Elements

5

This section describes both the platform capabilities and variables that Macromedia Flash Lite 1.1 recognizes, and the Flash Lite commands you can execute using the `fscommand()` and `fscommand2()` functions. The functionality described in this section is specific to Flash Lite.

The contents of this section are summarized in the following table:

Language element	Description
`_capCompoundSound`	Indicates whether Flash Lite can process compound sound.
`_capEmail`	Indicates whether the Flash Lite client can send e-mail messages using the `GetURL()` ActionScript command.
`_capLoadData`	Indicates whether the host application can dynamically load additional data through calls to the `loadMovie()`, `loadMovieNum()`, `loadVariables()`, and `loadVariablesNum()` functions.
`_capMFi`	Indicates whether the device can play sound data in the Melody Format for i-mode (MFi) audio format
`_capMIDI`	Indicates whether the device can play sound data in the Musical Instrument Digital Interface (MIDI) audio format.
`_capMMS`	Indicates whether Flash Lite can send Multimedia Messaging Service (MMS) messages by using the `GetURL()` ActionScript command.
`_capMP3`	Indicates whether the device can play sound data in the MPEG Audio Layer 3 (MP3) audio format.
`_capSMAF`	Indicates whether the device can play multimedia files in the Synthetic music Mobile Application Format (SMAF).
`_capSMS`	Indicates whether Flash Lite can send Short Message Service (SMS) messages by using the `GetURL()` ActionScript command.
`_capStreamSound`	Indicates whether the device can play streaming (synchronized) sound.

Language element	Description
`_cap4WayKeyAS`	Indicates whether Flash Lite executes ActionScript expressions attached to key event handlers associated with the Right, Left, Up, and Down Arrow keys.
`$version`	Contains the version number of Flash Lite.
`fscommand()`	A function used to execute the `Launch` command (see next entry).
`Launch`	(The only command supported for `fscommand()`) Allows the SWF file to communicate with either Flash Lite or the host environment, such as the phone's or device's operating system.
`fscommand2()`	A function used to execute the commands in this table, except for `fscommand()`.
`Escape`	Encodes an arbitrary string into a format that is safe for network transfer.
`FullScreen`	Sets the size of the display area to be used for rendering.
`GetBatteryLevel`	Returns the current battery level.
`GetDateDay`	Returns the day of the current date as a numeric value.
`GetDateMonth`	Returns the month of the current date as a numeric value.
`GetDateWeekday`	Returns the number of the day of the current date as a numeric value.
`GetDateYear`	Returns a four-digit numeric value that is the year of the current date.
`GetDevice`	Sets a parameter that identifies the device on which Flash Lite is running.
`GetDeviceID`	Sets a parameter that represents the unique identifier of the device; for example, the serial number.
`GetFreePlayerMemory`	Returns the amount of heap memory, in kilobytes, currently available to Flash Lite.
`GetLanguage`	Sets a parameter that identifies the language currently used by the device.
`GetLocaleLongDate`	Sets a parameter to a string that represents the current date, in long form, formatted according to the currently defined locale.
`GetLocaleShortDate`	Sets a parameter to a string that represents the current date, in abbreviated form, formatted according to the currently defined locale.
`GetLocaleTime`	Sets a parameter to a string that represents the current time, formatted according to the currently defined locale.

Language element	Description
GetMaxBatteryLevel	Returns the maximum battery level of the device.
GetMaxSignalLevel	Returns the maximum signal strength level.
GetMaxVolumeLevel	Returns the maximum volume level of the device as a numeric value.
GetNetworkConnectStatus	Returns a value that indicates the current network connection status.
GetNetworkName	Sets a parameter to the name of the current network.
GetNetworkRequestStatus	Returns a value that indicates the status of the most recent HTTP request.
GetNetworkStatus	Returns a value that indicates the network status of the phone (that is, whether there is a network registered and whether the phone is currently roaming).
GetPlatform	Sets a parameter that identifies the current platform, which broadly describes the class of device. For devices with open operating systems, this identifier is typically the name and version of the operating system.
GetPowerSource	Returns a value that indicates whether the power source is currently supplied from a battery or from an external power source.
GetSignalLevel	Returns the current signal strength as a numeric value.
GetTimeHours	Returns the hour of the current time of day, based on a 24-hour clock as a numeric value.
GetTimeMinutes	Returns the minute of the current time of day as a numeric value.
GetTimeSeconds	Returns the second of the current time of day as a numeric value.
GetTimeZoneOffset	Sets a parameter to the number of minutes between the local time zone and universal time (UTC).
GetTotalPlayerMemory	Returns the total amount of heap memory, in kilobytes, allocated to Flash Lite.
GetVolumeLevel	Returns the current volume level of the device as a numeric value.
Quit	Causes the Flash Lite player to stop playback and exit.
ResetSoftKeys	Resets the soft keys to their original settings.
SetInputTextType	Specifies the mode in which the input text field should be opened.
SetQuality	Sets the rendering quality of the animation.

Language element	Description
SetSoftKeys	Remaps the Left and Right soft keys of the device, provided that they can be accessed and remapped.
StartVibrate	Starts the phone's vibration feature.
StopVibrate	Stops the current vibration, if any.
Unescape	Decodes an arbitrary string that was encoded to be safe for network transfer into its normal, unencoded form.

Capabilities

This section describes the platform capabilities and variables that Macromedia Flash Lite 1.1 recognizes. The entries are listed alphabetially, ignoring any leading underscores.

_capCompoundSound

Availability

Flash Lite 1.1.

Usage

```
_capCompoundSound
```

Description

Numeric 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.

As an example, a single Flash file can contain the same sound represented in both MIDI and MFi formats. The player will then play back data in the appropriate format based on the format supported by the device. This variable defines whether the Flash Lite player supports this ability on the current handset.

In the following example, `useCompoundSound` is set to 1 in Flash Lite 1.1, but is undefined in Flash Lite 1.0:

```
useCompoundSound = _capCompoundSound;

if (useCompoundSound == 1) {
  gotoAndPlay("withSound");
} else {
  gotoAndPlay("withoutSound");
}
```

_capEmail

Availability

Flash Lite 1.1.

Usage

```
_capEmail
```

Description

Numeric variable; indicates whether the Flash Lite client can send e-mail 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

If the host application can send e-mail messages by using the `GetURL()` ActionScript command, the following example sets `canEmail` to 1:

```
canEmail = _capEmail;

if (canEmail == 1) {
  getURL("mailto:someone@somewhere.com?subject=foo&body=bar");
}
```

_capLoadData

Availability

Flash Lite 1.1.

Usage

```
_capLoadData
```

Description

Numeric variable; indicates whether the host application can dynamically load additional data through calls to the `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

If the host application can perform dynamic loading of movies and variables, the following example sets `iCanLoad` to 1:

```
canLoad = _capLoadData;

if (canLoad == 1) {
  loadVariables("http://www.somewhere.com/myVars.php", GET);
} else {
  trace ("client does not support loading dynamic data");
}
```

_capMFi

Availability

Flash Lite 1.1.

Usage

```
_capMFi
```

Description

Numeric variable; indicates whether the device can play sound data in the Melody Format for i-mode (MFi) audio format. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

Example

If the device can play MFi sound data, the following example sets `canMfi` to 1:

```
canMFi = _capMFi;

if (canMFi == 1) {
  // send movieclip buttons to frame with buttons that trigger events
  sounds
  tellTarget("buttons") {
    gotoAndPlay(2);
  }
}
```

_capMIDI

Availability

Flash Lite 1.1.

Usage

```
_capMIDI
```

Description

Numeric variable; indicates whether the device can play sound data in the Musical Instrument Digital Interface (MIDI) audio format. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

Example

If the device can play MIDI sound data, the following example sets `canMidi` to 1:

```
canMIDI = _capMIDI;

if (canMIDI == 1) {
  // send movieclip buttons to frame with buttons that trigger events
  sounds
  tellTarget("buttons") {
    gotoAndPlay(2);
  }
}
```

_capMMS

Availability

Flash Lite 1.1.

Usage

```
_capMMS
```

Description

Numeric variable; indicates whether Flash Lite can send Multimedia Messaging Service (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

The following example sets `canMMS` to 1 in Flash Lite 1.1, but leaves it undefined in Flash Lite 1.0 (however, not all Flash Lite 1.1 phones can send MMS messages, so this code is still dependent on the phone):

```
on(release) {
  canMMS = _capMMS;
  if (canMMS == 1) {
    // send an MMS
    myMessage = "mms:4156095555?body=sample mms message";
  } else {
    // send an SMS
    myMessage = "sms:4156095555?body=sample sms message";
  }
  getURL(myMessage);
}
```

_capMP3

Availability

Flash Lite 1.1.

Usage

```
_capMP3
```

Description

Numeric variable; indicates whether the device can play sound data in the MPEG Audio Layer 3 (MP3) audio format. If so, this variable is defined and has a value of 1; if not, this variable is undefined.

Example

If the device can play MP3 sound data, the following example sets `canMP3` to 1:

```
canMP3 = _capMP3;
if (canMP3 == 1) {
  tellTarget("soundClip") {
    gotoAndPlay(2);
  }
}
```

_capSMAF

Availability

Flash Lite 1.1.

Usage

```
_capSMAF
```

Description

Numeric variable; indicates whether the device can play multimedia files in the Synthetic music Mobile Application Format (SMAF). If so, this variable is defined and has a value of 1; if not, this variable is undefined.

Example

The following example sets `canSMAF` to 1 in Flash Lite 1.1, but leaves it undefined in Flash Lite 1.0 (however, not all Flash Lite 1.1 phones can send SMAF messages, so this code is still dependent on the phone):

```
canSMAF = _capSMAF;

if (canSMAF) {
  // send movieclip buttons to frame with buttons that trigger events
  sounds
  tellTarget("buttons") {
    gotoAndPlay(2);
  }
}
```

_capSMS

Availability

Flash Lite 1.1.

Usage

```
_capSMS
```

Description

Numeric variable; indicates whether Flash Lite can send *Short Message Service* (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

The following example sets `canSMS` to 1 in Flash Lite 1.1, but leaves it undefined in Flash Lite 1.0 Flash Lite 1.0 (however, not all Flash Lite 1.1 phones can send SMS messages, so this code is still dependent on the phone):

```
on(release) {
  canSMS = _capSMS;
  if (canSMS) {
    // send an SMS
    myMessage = "sms:4156095555?body=sample sms message";
    getURL(myMessage);
  }
}
```

_capStreamSound

Availability

Flash Lite 1.1.

Usage

```
_capStreamSound
```

Description

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

Example

The following example plays streaming sound if `canStreamSound` is enabled:

```
on(press) {
  canStreamSound = _capStreamSound;
  if (canStreamSound) {
    // play a streaming sound in a movieclip with this button
    tellTarget("music") {
      gotoAndPlay(2);
    }
  }
}
```

_cap4WayKeyAS

Availability

Flash Lite 1.1.

Usage

```
_cap4WayKeyAS
```

Description

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

When one of the four-way keys is pressed, if the value of this variable is 1, Flash Lite first looks for a handler for that key. If it finds none, Flash control navigation occurs. However, if an event handler is found, no navigation action occurs for that key. For example, if a key press handler for the Down Arrow key is found, the user cannot navigate.

Example

The following example sets `canUse4Way` to 1 in Flash Lite 1.1, but leaves it undefined in Flash Lite 1.0 (however, not all Flash Lite 1.1 phones support four-way keys, so this code is still dependent on the phone):

```
canUse4Way = _cap4WayKeyAS;
 if (canUse4Way == 1) {
  msg = "Use your directional joypad to navigate this application";
 } else {
  msg = "Please use the 2 key to scroll up, the 6 key to scroll right, the
  8 key to scroll down, and the 4 key to scroll left.";
 }
```

$version

Availability

Flash Lite 1.1.

Usage

```
$version
```

Description

String 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.

The major number reported for all Flash Lite 1.x products is 5. Flash Lite 1.0 has a minor number of 1; Flash Lite 1.1 has a minor number of 2.

Example

In the Flash Lite 1.1 player, the following code sets the value of `myVersion` to "5, 2, 12, 0":

```
myVersion = $version;
```

fscommand()

Availability

Flash Lite 1.1.

Usage

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

Parameters

`"Launch"` The command specifier. The `Launch` command is the only command that you use the `fscommand()` function to execute.

`"application-path, arg1, arg2,..., argn"` The name of the application being started and the parameters to it, separated by commas.

Description

Function; allows the SWF file to communicate with either Flash Lite or the host environment, such as the phone's or device's operating system.

See also

`fscommand2()`

Launch

Availability

Flash Lite 1.1.

Usage

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

Parameters

`"Launch"` The command specifier. In Flash Lite, you use the `fscommand()` function only to execute the `Launch` command.

`"application-path, arg1, arg2,..., argn"` The name of the application being started and the parameters to it, separated by commas.

Description

Command executed through the `fscommand()` function; launches another application on the device. The name of the application being launched and the parameters to it are passed in as a single argument.

NOTE	This feature is operating-system dependent.

This command is supported only when the Flash Lite player 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).

Example

The following example would open wap.yahoo.com on the services/Web browser on Series 60 phones:

```
on(keyPress "9") {
  status = fscommand("launch",
  "z:\\system\\apps\\browser\\browser.app,http://wap.yahoo.com");
}
```

See also

`fscommand2()`

fscommand2()

Availability

Flash Lite 1.1.

Usage

```
returnValue = fscommand2(command [, expression1 ... expressionN])
```

Parameters

command A string passed to the host application for any use or a command passed to Flash Lite.

parameter1...parameterN A comma-delimited list of strings passed as parameters to the command specified by *command*.

Description

Function; allows the SWF file to communicate with either Flash Lite or the host environment, such as the phone or device's operating system. The value that `fscommand2()` returns depends on the specific command.

The `fscommand2()` function is similar to the `fscommand()` function, with the following differences:

- The `fscommand2()` function can take an arbitrary number of arguments.
- Flash Lite executes `fscommand2()` 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.

The strings and expressions that you pass to the function as commands and parameters are described in the tables in this section.

The tables have the following three columns:

- The Command column shows the string literal parameter that identifies the command.
- The Parameters column explains what kinds of values to pass for the additional parameters, if any.
- The Value returned column explains the expected return values.

Example

Examples are provided with the specific commands that you execute using the `fscommand2()` function, which are described in the rest of this section.

See also

`fscommand()`

Escape

Availability

Flash Lite 1.1.

Description

Encodes an arbitrary string into a format that is safe for network transfer. Replaces each nonalphanumeric character with a hexadecimal escape sequence (`%xx`, or `%xx%xx` in the case of multibyte characters).

Command	Parameters	Value returned
`"Escape"`	*`original`* String to be encoded into a format safe for URLs. *`encoded`* Resulting encoded string. These parameters are either names of variables or constant string values (for example, `"Encoded_String"`).	0: Failure. 1: Success.

Example

The following example shows the conversion of a sample string to its encoded form:

```
original_string = "Hello, how are you?";
status = fscommand2("escape", original_string, "encoded_string");
trace (encoded_string);  // output: Hello%2C%20how%20are%20you%3F
```

See also

`Unescape`

FullScreen

Availability

Flash Lite 1.1.

Description

Sets the size of the display area to be used for rendering. The size can be full screen or less-than full screen.

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).

Command	Parameters	Value returned
`"FullScreen"`	*`size`* Either a defined variable or a constant string value, with one of these values: `true` (full screen) or `false` (less than full screen). Any other value is treated as the value `false`.	-1: Not supported. 0: Supported.

Example

The following example attempts to set the display area to full screen. If the returned value is other than 0, it sends the playback head to the frame labeled smallScreenMode:

```
status = fscommand2("FullScreen", true);
if(status != 0) {
  gotoAndPlay("smallScreenMode");
}
```

GetBatteryLevel

Availability

Flash Lite 1.1.

Description

Returns the current battery level. It is a numeric value that ranges from 0 to the maximum value returned by the `GetMaxBatteryLevel` variable.

Command	Parameters	Value returned
`"GetBatteryLevel"`	None.	-1: Not supported. Other numeric values: The current battery level.

Example

The following example sets the `battLevel` variable to the current level of the battery:

```
battLevel = fscommand2("GetBatteryLevel");
```

See also

`GetMaxBatteryLevel`

GetDateDay

Availability

Flash Lite 1.1.

Description

Returns the day of the current date. It is a numeric value (without a leading 0). Valid days are 1 through 31.

Command	Parameters	Value returned
`"GetDateDay"`	None.	-1: Not supported. 1 to 31: The day of the month.

Example

The following example collects the date information and constructs a complete date string:

```
today = fscommand2("GetDateDay");
weekday = fscommand2("GetDateWeekday");
thisMonth = fscommand2("GetDateMonth");
thisYear = fscommand2("GetDateYear");
when = weekday add ", " add ThisMonth add " " add today add ", " add
  thisYear;
```

See also

`GetDateMonth`, `GetDateWeekday`, `GetDateYear`

GetDateMonth

Availability

Flash Lite 1.1.

Description

Returns the month of the current date as a numeric value (without a leading 0).

Command	Parameters	Value returned
`"GetDateMonth"`	None.	-1: Not supported. 1 to 12: The number of the current month.

Example

The following example collects the date information and constructs a complete date string:

```
today = fscommand2("GetDateDay");
weekday = fscommand2("GetDateWeekday");
thisMonth = fscommand2("GetDateMonth");
thisYear = fscommand2("GetDateYear");
when = weekday add ", " add thisMonth add " " add today add ", " add
  thisYear;
```

See also

GetDateDay, GetDateWeekday, GetDateYear

GetDateWeekday

Availability

Flash Lite 1.1.

Description

Returns a numeric value that is the name of the day of the current date, represented as a numeric value.

Command	Parameters	Value returned
`"GetDateWeekday"`	None.	-1: Not supported. 0: Sunday. 1: Monday. 2: Tuesday. 3: Wednesday. 4: Thursday. 5: Friday. 6: Saturday.

Example

The following example collects the date information and constructs a complete date string:

```
today = fscommand2("GetDateDay");
weekday = fscommand2("GetDateWeekday");
thisMonth = fscommand2("GetDateMonth");
thisYear = fscommand2("GetDateYear");
when = weekday add ", " add thisMonth add " " add today add ", " add
  thisYear;
```

See also

GetDateDay, GetDateMonth, GetDateYear

GetDateYear

Returns a four-digit numeric value that is the year of the current date.

Command	Parameters	Value returned
`"GetDateYear"`	None.	-1: Not supported. 0 to 9999: The current year.

Availability

Flash Lite 1.1.

Example

The following example collects the date information and constructs a complete date string:

```
today = fscommand2("GetDateDay");
weekday = fscommand2("GetDateWeekday");
thisMonth = fscommand2("GetDateMonth");
thisYear = fscommand2("GetDateYear");
when = weekday add ", " add thisMonth add " " add today add ", " add
  thisYear;
```

See also

`GetDateDay`, `GetDateMonth`, `GetDateWeekday`

GetDevice

Sets a parameter that identifies the device on which Flash Lite is running. This identifier is typically the model name.

Command	Parameters	Value returned
`"GetDevice"`	*`device`* String to receive the identifier of the device. It can be either the name of a variable or a string value that contains the name of a variable.	-1: Not supported. 0: Supported.

Availability

Flash Lite 1.1.

Example

The following code example assigns the device identifier to the `statusdevice` variable, and then updates a text field with the generic device name.

These are some sample results and the devices they signify:

`D506i` A Mitsubishi 506i phone.

`DFOMA1` A Mitsubishi FOMA1 phone.

`F506i` A Fujitsu 506i phone.

`FFOMA1` A Fujitsu FOMA1 phone.

`N506i` An NEC 506i phone.

`NFOMA1` An NEC FOMA1 phone.

`Nokia3650` A Nokia 3650 phone.

`p506i` A Panasonic 506i phone.

`PFOMA1` A Panasonic FOMA1 phone.

`SH506i` A Sharp 506i phone.

`SHFOMA1` A Sharp FOMA1 phone.

`SO506i` A Sony 506iphone.

```
statusdevice = fscommand2("GetDevice", "devicename");
switch(devicename) {
  case "D506i":
    /:myText += "device: Mitsubishi 506i" add newline;
    break;
  case "DFOMA1":
    /:myText += "device: Mitsubishi FOMA1" add newline;
    break;
  case "F506i":
    /:myText += "device: Fujitsu 506i" add newline;
    break;
  case "FFOMA1":
    /:myText += "device: Fujitsu FOMA1" add newline;
    break;
  case "N506i":
    /:myText += "device: NEC 506i" add newline;
    break;
  case "NFOMA1":
    /:myText += "device: NEC FOMA1" add newline;
    break;
  case "Nokia 3650":
    /:myText += "device: Nokia 3650" add newline;
    break;
```

```
  case "P506i":
    /:myText += "device: Panasonic 506i" add newline;
    break;
  case "PFOMA1":
    /:myText += "device: Panasonic FOMA1" add newline;
    break;
  case "SH506i":
    /:myText += "device: Sharp 506i" add newline;
    break;
  case "SHFOMA1":
    /:myText += "device: Sharp FOMA1" add newline;
    break;
  case "SO506i":
    /:myText += "device: Sony 506i" add newline;
    break;
}
```

GetDeviceID

Sets a parameter that represents the unique identifier of the device (for example, the serial number).

Command	Parameters	Value returned
`"GetDeviceID"`	*id* A string to receive the unique identifier of the device. It can be either the name of a variable or a string value that contains the name of a variable.	-1: Not supported. 0: Supported.

Availability

Flash Lite 1.1.

Example

The following example assigns the unique identifier to the `deviceID` variable:

```
status = fscommand2("GetDeviceID", "deviceID");
```

GetFreePlayerMemory

Returns the amount of heap memory, in kilobytes, currently available to Flash Lite.

Command	Parameters	Value returned
`"GetFreePlayerMemory"`	None.	-1: Not supported. 0 or positive value: Available kilobytes of heap memory.

Availability

Flash Lite 1.1.

Example

The following example sets status equal to the amount of free memory:

```
status = fscommand2("GetFreePlayerMemory");
```

See also

GetTotalPlayerMemory

GetLanguage

Availability

Flash Lite 1.1.

Sets a parameter that identifies the language currently used by the device. The language is returned as a string in a variable that is passed in by name.

Command	Parameters	Value returned
"GetLanguage"	*language* String to receive the language code. It can be either the name of a variable or a string value that contains the name of a variable. The value returned is one of the following: cs: Czech. da: Danish. de: German. en-UK: UK or international English. en-US: US 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: an undetermined language. zh-CN: simplified Chinese. zh-TW: traditional Chinese.	-1: Not supported. 0: Supported.

NOTE

When Japanese phones are set to display English, en_US is returned for *language*.

Example

The following example assigns the language code to the `language` variable, and then updates a text field with the language recognized by the Flash Lite player:

```
statuslanguage = fscommand2("GetLanguage", "language");
switch(language) {
  case "cs":
    /:myText += "language is Czech" add newline;
    break;
  case "da":
    /:myText += "language is Danish" add newline;
    break;
  case "de":
    /:myText += "language is German" add newline;
    break;
  case "en-UK":
    /:myText += "language is UK" add newline;
    break;
  case "en-US":
    /:myText += "language is US" add newline;
    break;
  case "es":
    /:myText += "language is Spanish" add newline;
    break;
  case "fi":
    /:myText += "language is Finnish" add newline;
    break;
  case "fr":
    /:myText += "language is French" add newline;
    break;
  case "hu":
    /:myText += "language is Hungarian" add newline;
    break;
  case "it":
    /:myText += "language is Italian" add newline;
    break;
  case "jp":
    /:myText += "language is Japanese" add newline;
    break;
  case "ko":
    /:myText += "language is Korean" add newline;
    break;
  case "nl":
    /:myText += "language is Dutch" add newline;
    break;
  case "no":
    /:myText += "language is Norwegian" add newline;
    break;
```

```
  case "pl":
    /:myText += "language is Polish" add newline;
    break;
  case "pt":
    /:myText += "language is Portuguese" add newline;
    break;
  case "ru":
    /:myText += "language is Russian" add newline;
    break;
  case "sv":
    /:myText += "language is Swedish" add newline;
    break;
  case "tr":
    /:myText += "language is Turkish" add newline;
    break;
  case "xu":
    /:myText += "language is indeterminable" add newline;
    break;
  case "zh-CN":
    /:myText += "language is simplified Chinese" add newline;
    break;
  case "zh-TW":
    /:myText += "language is traditional Chinese" add newline;
    break;
}
```

GetLocaleLongDate

Availability

Flash Lite 1.1.

Description

Sets a parameter to a string that represents the current date, in long form, formatted according to the currently defined locale.

Command	Parameters	Value returned
`"GetLocaleLongDate"`	*`longdate`* String variable to receive the long form of the value of the current date, such as `"October 16, 2004"` or `"16 October 2004"`. It can be either the name of a variable or a string value that contains the name of a variable. The value returned in *`longdate`* is a multicharacter, variable-length string. The actual formatting depends on the device and the locale.	-1: Not supported. 0: Supported.

Example

The following example attempts to return the long form of the current date in the `longDate` variable. It also sets the value of `status` to report whether it was able to do so.

```
status = fscommand2("GetLocaleLongDate", "longdate");
trace (longdate);      // output: Tuesday, June 14, 2005
```

See also

`GetLocaleShortDate`, `GetLocaleTime`

GetLocaleShortDate

Availability

Flash Lite 1.1.

Description

Sets a parameter to a string that represents the current date, in abbreviated form, formatted according to the currently defined locale.

Command	Parameters	Value returned
`"GetLocaleShortDate"`	*`shortdate`* String variable to receive the long form of the value of the current date, such as `"10/16/2004"` or `"16-10-2004"`. It can be either the name of a variable or a string value that contains the name of a variable. The value returned in *`shortdate`* is a multicharacter, variable-length string. The actual formatting depends on the device and the locale.	-1: Not supported. 0: Supported.

Example

The following example attempts to get the short form of the current date into the `shortDate` variable. It also sets the value of `status` to report whether it was able to do so.

```
status = fscommand2("GetLocaleShortDate", "shortdate");
trace (shortdate);     // output: 06/14/05
```

See also

`GetLocaleLongDate`, `GetLocaleTime`

GetLocaleTime

Availability

Flash Lite 1.1.

Description

Sets a parameter to a string representing the current time, formatted according to the currently defined locale.

Command	Parameters	Value returned
`"GetLocaleTime"`	*time* String variable to receive the value of the current time, such as `"6:10:44 PM"` or `"18:10:44"`. It can be either the name of a variable or a string value that contains the name of a variable. The value returned in *time* is a multicharacter, variable-length string. The actual formatting depends on the device and the locale.	-1: Not supported. 0: Supported.

Example

The following example attempts to get the current local time into the `time` variable. It also sets the value of `status` to report whether it was able to do so.

```
status = fscommand2("GetLocaleTime", "time");
trace(time);                // output: 14:30:21
```

See also

`GetLocaleLongDate`, `GetLocaleShortDate`

GetMaxBatteryLevel

Availability

Flash Lite 1.1.

Description

Returns the maximum battery level of the device. It is a numeric value greater than 0.

Command	Parameters	Value returned
`"GetMaxBatteryLevel"`	None.	-1: Not supported. other values: The maximum battery level.

Example

The following example sets the `maxBatt` variable to the maximum battery level:

```
maxBatt = fscommand2("GetMaxBatteryLevel");
```

GetMaxSignalLevel

Availability

Flash Lite 1.1.

Description

Returns the maximum signal strength level. It is a numeric value greater than 0.

Command	Parameters	Value returned
`"GetMaxSignalLevel"`	None.	-1: Not supported. Other numeric values: The maximum signal level.

Example

The following example assigns the maximum signal strength to the `sigStrengthMax` variable:

```
sigStrengthMax = fscommand2("GetMaxSignalLevel");
```

GetMaxVolumeLevel

Availability

Flash Lite 1.1.

Description

Returns the maximum volume level of the device as a numeric value.

Command	Parameters	Value returned
`"GetMaxVolumeLevel"`	None.	-1: Not supported. Other values: The maximum volume level.

Example

The following example sets the `maxvolume` variable to the maximum volume level of the device:

```
maxvolume = fscommand2("GetMaxVolumeLevel");
trace (maxvolume);     // output: 80
```

See also

`GetVolumeLevel`

GetNetworkConnectStatus

Availability

Flash Lite 1.1.

Description

Returns a value that indicates the current network connection status.

Command	Parameters	Value returned
`"GetNetworkConnectStatus"`	None.	-1: Not supported. 0: There is currently an active network connection. 1: The device is attempting to connect to the network. 2: There is currently no active network connection. 3: The network connection is suspended. 4: The network connection cannot be determined.

Example

The following example assigns the network connection status to the `connectstatus` variable, and then uses a `switch` statement to update a text field with the status of the connection:

```
connectstatus = fscommand2("GetNetworkConnectStatus");
switch (connectstatus) {
  case -1 :
    /:myText += "connectstatus not supported" add newline;
    break;
  case 0 :
    /:myText += "connectstatus shows active connection" add newline;
    break;
  case 1 :
    /:myText += "connectstatus shows attempting connection" add newline;
    break;
  case 2 :
    /:myText += "connectstatus shows no connection" add newline;
    break;
  case 3 :
    /:myText += "connectstatus shows suspended connection" add newline;
    break;
  case 4 :
    /:myText += "connectstatus shows indeterminable state" add newline;
    break;
}
```

GetNetworkName

Availability

Flash Lite 1.1.

Description

Sets a parameter to the name of the current network.

Command	Parameters	Value returned
`"GetNetworkName"`	*`networkName`* String representing the network name. It can be either the name of a variable or a string value that contains the name of a variable. If the network is registered and its name can be determined, *`networkname`* is set to the network name; otherwise, it is set to the empty string.	-1: Not supported. 0: No network is registered. 1: Network is registered, but network name is not known. 2: Network is registered, and network name is known.

Example

The following example assigns the name of the current network to the `myNetName` variable and a status value to the `netNameStatus` variable:

```
netNameStatus = fscommand2("GetNetworkName", myNetName);
```

GetNetworkRequestStatus

Availability

Flash Lite 1.1.

Description

Returns a value indicating the status of the most recent HTTP request.

Command	Parameters	Value returned
`"GetNetworkRequestStatus"`	None.	-1: The command is not supported. 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 has 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.

Example

The following example assigns the status of the most recent HTTP request to the `requesttatus` variable, and then uses a `switch` statement to update a text field with the status:

```
requeststatus = fscommand2("GetNetworkRequestStatus");
switch (requeststatus) {
  case -1:
    /:myText += "requeststatus not supported" add newline;
    break;
  case 0:
    /:myText += "connection to server has been made" add newline;
    break;
  case 1:
    /:myText += "connection is being established" add newline;
    break;
  case 2:
    /:myText += "pending request, contacting network" add newline;
    break;
  case 3:
    /:myText += "pending request, resolving domain" add newline;
    break;
  case 4:
    /:myText += "failed, network error" add newline;
    break;
  case 5:
    /:myText += "failed, couldn't reach server" add newline;
    break;
  case 6:
    /:myText += "HTTP error" add newline;
    break;
  case 7:
    /:myText += "DNS failure" add newline;
    break;
  case 8:
    /:myText += "request has been fulfilled" add newline;
    break;
  case 9:
    /:myText += "request timedout" add newline;
    break;
  case 10:
    /:myText += "no HTTP request has been made" add newline;
    break;
}
```

GetNetworkStatus

Availability

Flash Lite 1.1.

Description

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).

Command	Parameters	Value returned
`"GetNetworkStatus"`	None.	-1: The command is not supported. 0: No network registered. 1: On home network. 2: On extended home network. 3: Roaming (away from home network).

Example

The following example assigns the status of the network connection to the `networkstatus` variable, and then uses a `switch` statement to update a text field with the status:

```
networkstatus = fscommand2("GetNetworkStatus");
switch(networkstatus) {
  case -1:
    /:myText += "network status not supported" add newline;
    break;
  case 0:
    /:myText += "no network registered" add newline;
    break;
  case 1:
    /:myText += "on home network" add newline;
    break;
  case 2:
    /:myText += "on extended home network" add newline;
    break;
  case 3:
    /:myText += "roaming" add newline;
    break;
}
```

GetPlatform

Availability

Flash Lite 1.1.

Description

Sets a parameter that identifies the current platform, which broadly describes the class of device. For devices with open operating systems, this identifier is typically the name and version of the operating system.

Command	Parameters	Value returned
`"GetPlatform"`	*`platform`* String to receive the identifier of the platform. It can be either the name of a variable or a string value that contains the name of a variable.	-1: Not supported. 0: Supported.

Example

The following code example assigns the platform identifier to the `statusplatform` variable, and then updates a text field with the generic platform name.

These are some sample results for `myPlatform` and the classes of device they signify:

`506i` A 506i phone.

`FOMA1` A FOMA1 phone.

`Symbian6.1_s60.1` A Symbian 6.1, Series 60 version 1 phone.

`Symbian7.0` A Symbian 7.0 phone

```
statusplatform = fscommand2("GetPlatform", "platform");
switch(platform){
  case "506i":
    /:myText += "platform: 506i" add newline;
    break;
  case "FOMA1":
    /:myText += "platform: FOMA1" add newline;
    break;
  case "Symbian6.1-Series60v1":
    /:myText += "platform: Symbian6.1, Series 60 version 1 phone" add
  newline;
    break;
  case "Symbian7.0":
    /:myText += "platform: Symbian 7.0" add newline;
    break;
}
```

GetPowerSource

Availability

Flash Lite 1.1.

Description

Returns a value that indicates whether the power source is currently supplied from a battery or from an external power source.

Command	Parameters	Value returned
`"GetPowerSource"`	None.	-1: Not supported. 0: Device is operating on battery power. 1: Device is operating on an external power source.

Example

The following example sets the `myPower` variable to indicate the power source, or to -1 if it was unable to do so:

```
myPower = fscommand2("GetPowerSource");
```

GetSignalLevel

Availability

Flash Lite 1.1.

Description

Returns the current signal strength as a numeric value.

Command	Parameters	Value returned
`"GetSignalLevel"`	None.	-1: Not supported. Other numeric values: The current signal level, ranging from 0 to the maximum value returned by `GetMaxSignalLevel`.

Example

The following example assigns the signal level value to the `sigLevel` variable:

```
sigLevel = fscommand2("GetSignalLevel");
```

GetTimeHours

Availability

Flash Lite 1.1.

Description

Returns the hour of the current time of day, based on a 24-hour clock. It is a numeric value (without a leading 0).

Command	Parameters	Value returned
`"GetTimeHours"`	None.	-1: Not supported. 0 to 23: The current hour.

Example

The following example sets the `hour` variable to the hour portion of the current time of day, or to -1:

```
hour = fscommand2("GetTimeHours");
trace (hour);            // output: 14
```

See also

`GetTimeMinutes`, `GetTimeSeconds`, `GetTimeZoneOffset`

GetTimeMinutes

Availability

Flash Lite 1.1.

Description

Returns the minute of the current time of day. It is a numeric value (without a leading 0).

Command	Parameters	Value returned
`"GetTimeMinutes"`	None.	-1: Not supported. 0 to 59: The current minute.

Example

The following example sets the `minutes` variable to the minute portion of the current time of day, or to -1:

```
minutes = fscommand2("GetTimeMinutes");
trace (minutes);         // output: 38
```

See also

GetTimeHours, GetTimeSeconds, GetTimeZoneOffset

GetTimeSeconds

Availability

Flash Lite 1.1.

Description

Returns the second of the current time of day. It is a numeric value (without a leading 0).

Command	Parameters	Value returned
`"GetTimeSeconds"`	None.	-1: Not supported. 0 to 59: The current second.

Example

The following example sets the `seconds` variable to the seconds portion of the current time of day, or to -1:

```
seconds = fscommand2("GetTimeSeconds");
trace (seconds);         // output: 41
```

See also

GetTimeHours, GetTimeMinutes, GetTimeZoneOffset

GetTimeZoneOffset

Availability

Flash Lite 1.1.

Description

Sets a parameter to the number of minutes between the local time zone and universal time (UTC).

Command	Parameters	Value returned
`"GetTimeZoneOffset"`	*`timezoneOffset`* Number of minutes between the local time zone and UTC. It can be either the name of a variable or a string value that contains the name of a variable. A positive or a negative numeric value is returned, such as the following: 540: Japan standard time -420: Pacific daylight saving time	-1: Not supported. 0: Supported.

Example

The following example either assigns the minutes of offset from UTC to the `timezoneoffset` variable and sets `status` to 0 or else sets `status` to -1:

```
status = fscommand2("GetTimeZoneOffset", "timezoneoffset");
trace (timezoneoffset);// output: 300
```

See also

`GetTimeHours`, `GetTimeMinutes`, `GetTimeSeconds`

GetTotalPlayerMemory

Availability

Flash Lite 1.1.

Description

Returns the total amount of heap memory, in kilobytes, allocated to Flash Lite.

Command	Parameters	Value returned
`"GetTotalPlayerMemory"`	None.	-1: Not supported. 0 or positive value: Total kilobytes of heap memory.

Example

The following example sets the `status` variable to the total amount of heap memory:

```
status = fscommand2("GetTotalPlayerMemory");
```

See also

`GetFreePlayerMemory`

GetVolumeLevel

Availability

Flash Lite 1.1.

Description

Returns the current volume level of the device as a numeric value.

Command	Parameters	Value returned
`"GetVolumeLevel"`	None.	-1: Not supported. Other numeric values: The current volume level, ranging from 0 to the value returned by `fscommand2("GetMaxVolumeLevel")`.

Example

The following example assigns the current volume level to the `volume` variable:

```
volume = fscommand2("GetVolumeLevel");
trace (volume);          // output: 50
```

See also

`GetVolumeLevel`

Quit

Availability

Flash Lite 1.1.

Description

Causes the Flash Lite player to stop playback and exit.

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).

Command	Parameters	Value returned
`"Quit"`	None.	-1: Not supported.

Example

The following example causes Flash Lite to stop playback and quit when running in stand-alone mode:

```
status = fscommand2("Quit");
```

ResetSoftKeys

Availability

Flash Lite 1.1.

Description

Resets the soft keys to their original settings.

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).

Command	Parameters	Value returned
`"ResetSoftKeys"`	None.	-1: Not supported. 0: Supported.

Example

The following statement resets the soft keys to their original settings:

```
status = fscommand2("ResetSoftKeys");
```

See also

SetSoftKeys

SetInputTextType

Availability

Flash Lite 1.1.

Description

Specifies the mode in which the input text field should be opened:

Command	Parameters	Value returned
"SetInputTextType"	*variableName* Name of the input text field. It can be either the name of a variable or a string value that contains the name of a variable. *type* One of the values Numeric, Alpha, Alphanumeric, Latin, NonLatin, or NoRestriction.	0: Failure. 1: Success.

Flash Lite supports input text functionality by asking the host application to start the generic device-specific text input interface, often referred to as the *front-end processor* (FEP). When the SetInputTextType command is not used, the FEP is opened in default mode.

The following table shows what effect each mode has, and what modes are substituted:

Mode specified	Sets the FEP to one of these mutually exclusive modes	If not supported on current device, opens the FEP in this mode
Numeric	Numbers only (0 to 9)	Alphanumeric
Alpha	Alphabetic characters only (A to Z, a to z)	Alphanumeric
Alphanumeric	Alphanumeric characters only (0 to 9, A to Z, a to z)	Latin
Latin	Latin characters only (alphanumeric and punctuation)	NoRestriction
NonLatin	Non-Latin characters only (for example, Kanji and Kana)	NoRestriction
NoRestriction	Default mode (sets no restriction on the FEP)	

NOTE

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

Example

The following line of code sets the input text type of the field associated with the `input1` variable to receive numeric data:

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

SetQuality

Availability

Flash Lite 1.1.

Description

Sets the quality of the rendering of the animation.

Command	Parameters	Value returned
`"SetQuality"`	*`quality`* The rendering quality; must be `"high"`, `"medium"`, or `"low"`.	-1: Not supported. 0: Supported.

Example

The following example sets the rendering quality to low:

```
status = fscommand2("SetQuality", "low");
```

SetSoftKeys

Availability

Flash Lite 1.1.

Description

Remaps the Left and Right soft keys of the device, provided that they can be accessed and remapped.

After this command 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 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).

Command	Parameters	Value returned
`"SetSoftKeys"`	*`left`* Text to be displayed for the Left soft key. *`right`* Text to be displayed for the Right soft key. These parameters are either names of variables or constant string values (for example, `"Previous"`).	-1: Not supported. 0: Supported.

Example

The following example directs that the Left soft key be labeled Previous and the Right soft key be labeled Next:

```
status = fscommand2("SetSoftKeys", "Previous", "Next");
```

See also

`ResetSoftKeys`

StartVibrate

Availability

Flash Lite 1.1.

Description

Starts the phone's vibration feature. If a vibration is already occurring, Flash Lite stops that vibration before starting the new one. Vibrations also stop when playback of the Flash application is stopped or paused, and when Flash Lite player quits.

Command	Parameters	Value returned
`"StartVibrate"`	*`time_on`* Amount of time, in milliseconds (to a maximum of 5 seconds), that the vibration is on. *`time_off`* Amount of time, in milliseconds (to a maximum of 5 seconds), that the vibration is off. *`repeat`* Number of times (to a maximum of 3) to repeat this vibration.	-1: Not supported. 0: Vibration was started. 1: An error occurred and vibration could not be started.

Example

The following example attempts to start a vibration sequence of 2.5 seconds on, 1 second off, repeated twice. It assigns a value to the `status` variable that indicates success or failure.

```
status = fscommand2("StartVibrate", 2500, 1000, 2);
```

See also

`StopVibrate`

StopVibrate

Availability

Flash Lite 1.1.

Description

Stops the current vibration, if any.

Command	Parameters	Value returned
`"StopVibrate"`	None.	-1: Not supported. 0: The vibration stopped.

Example

The following example calls `StopVibrate` and saves the result (not supported or vibration stopped) in the `status` variable:

```
status = fscommand2("StopVibrate");
```

See also

`StartVibrate`

Unescape

Availability

Flash Lite 1.1.

Description

Decodes an arbitrary string that was encoded to be safe for network transfer into its normal, unencoded form. All characters that are in hexadecimal format, that is, a percent character (%) followed by two hexadecimal digits, are converted into their decoded form.

Command	Parameters	Value returned
`"Unescape"`	*`original`* String to be decoded from a format safe for URLs to a normal form. *`decoded`* Resulting decoded string. (This parameter can be either the name of a variable or a string value that contains the name of a variable.)	0: Failure. 1: Success.

Example

The following example shows the decoding of an encoded string:

```
encoded_string = "Hello%2C%20how%20are%20you%3F";
status = fscommand2("unescape", encoded_string, "normal_string");
trace (normal_string);   // output: Hello, how are you?
```

See also

`Escape`

Index

Symbols

A

B

H

I

L

M

N

O

P

R

S

T

U

V

W

X

Y