

Chapter 7: Reference

Introduction

This chapter lists all syntax (keywords, statements, operators, objects, methods, properties, and global functions) recognized by the LiveMotion scripting engine. Table 7.1 lists and describes all keywords and statements. Table 7.2 shows the precedence and associativity for all operators. Table 7.3 lists and describes all operators. The remainder of the chapter is an alphabetical listing of all built-in objects, global properties and functions.

Table 7.1 *Keywords and Statement Syntax*

Keyword/Statement	Description
<code>break</code>	Standard JavaScript construct. Exit the currently executing loop.
<code>continue</code>	Standard JavaScript construct. Cease execution of the current loop iteration.
<code>do - while</code>	Standard JavaScript construct. Similar to the <code>while</code> loop, except loop condition evaluation occurs at the end of the loop.
<code>false</code>	Literal representing Boolean false.
<code>for</code>	Standard JavaScript loop construct.
<code>for - in</code>	Standard JavaScript construct. Provides a way to easily loop through the properties of an object.
<code>function</code>	Used to define a function.
<code>if/if - else</code>	Standard JavaScript conditional constructs.
<code>#include</code>	Standard JavaScript directive used to import files located elsewhere.
<code>null</code>	Assigned to a variable, array element, or object property to indicate that it does not contain a legal value.
<code>return</code>	Standard JavaScript way of returning a value from a function or exiting a function.
<code>set</code>	Used to assign a value to a dynamically created variable.
<code>switch</code>	Standard JavaScript way of evaluating an expression and attempting to match the expression's value to a case label.
<code>this</code>	Standard JavaScript method of indicating the current object.

Keyword/Statement	Description
true	Literal representing Boolean true.
undefined	Indicates that the variable, array element, or object property has not yet been assigned a value.
var	Standard JavaScript syntax used to declare a local variable.
while	Standard JavaScript construct. Similar to the do - while loop, except loop condition evaluation occurs at the beginning of the loop.
with	Standard JavaScript construct used to specify an object to use in ensuing statements.

Table 7.2 *Operator Precedence*

Operators (Listed from highest precedence —top row—to lowest)	Associativity
[], (), .	left to right
new, delete, -(unary negation), ~, !, typeof, void, ++, --	right to left
*, /, %	left to right
+, -(subtraction)	left to right
<<, >>, >>>	left to right
<, <=, >, >=	left to right
==, !=	left to right
&	left to right
^	left to right
	left to right
&&	left to right
	left to right
?:	right to left
=, /=, %=, <<=, >>=, >>>=, &=, ^=, =, +=, -=, *=	right to left
,	left to right

Table 7.3 Description of Operators

Operators	Description
new	Allocate object
delete	Deallocate object
typeof	Data type
void	Returns undefined value
.	Structure member
[]	Array element
()	Function call
++	Pre- or post-increment
--	Pre- or post-decrement
-	Unary negation or subtraction
~	Bitwise NOT
!	Logical NOT
*	Multiply
/	Divide
%	Modulo division
+	Add
<<	Bitwise left shift
>>	Bitwise right shift
>>>	Unsigned bitwise right shift
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal

Operators	Description
!=	Not equal
&	Bitwise AND
^	Bitwise XOR
	Bitwise OR
&&	Logical AND
	Logical OR
?:	Conditional (ternary)
=	Assignment
+=	Assignment with add operation
-=	Assignment with subtract operation
*=	Assignment with multiply operation
/=	Assignment with divide operation
%=	Assignment with modulo operation
<<=	Assignment with bitwise left shift operation
>>=	Assignment with bitwise right shift operation
>>>=	Assignment with bitwise right shift unsigned operation
&=	Assignment with bitwise AND operation
^=	Assignment with bitwise XOR operation
=	Assignment with bitwise OR operation
,	Multiple evaluation

Arguments Object

Description

The Arguments object provides two types of information about an executing function:

- the name of the function itself, and
- the arguments that were passed to the function.

The Arguments object is a static object—to use the object, do not create an instance using a constructor.

Properties

callee	See “Arguments.callee Property” on page 113	The name of the currently executing function.
length	See “Arguments.length Property” on page 113	The number of parameters passed to the currently executing function. This value can be used to access the individual parameters themselves.

Arguments.callee Property

arguments.callee

Description

The callee property holds a reference to the currently executing function. This property can only be read.

Example

```
function selfReferenceTest()
{
  if (arguments.callee == selfReferenceTest)
    trace("true");
  else
    trace("false");
};

selfReferenceTest();//prints "true"
```

Arguments.length Property

arguments.length

Description

The `length` property stores an integer specifying the number of parameters passed to the currently executing function. The property can then be used to access the names of the individual arguments themselves, using the `arguments` object as an array. The `length` object, however, is not zero-based, so will always have a value of one greater than the largest index into the array. This property can only be read.

Example

```
function baseball(glove, bat)
{
  trace(arguments.length);
  trace(arguments[0]);
  trace(arguments[1]);
};

baseball("catchers", "wooden");
//prints
//2
//catchers
//wooden
```

Array Object

Description

The `Array` object provides the ability to create and manipulate arrays of data. If the `Array` constructor is invoked with a single integer value, the value sets the array length. If two or more values are used, they become the initial values of the array elements, and the array length is determined by the number of values provided. Similarly, a single non-numeric value can be used to initialize the array with a single element with that value.

To call the `Array` object's methods, you must create a new object using the constructor. Alternatively, you may use the bracket syntax (e.g., `var x = [a, b]` populates the first two elements of the array with the values `a` and `b`). If the `Array` constructor is invoked without passing arguments to `Array`, then an empty array is created with 0 elements.

Constructor

```
new Array()
```

```
new Array(length)  
new Array(element0, ...elementn)
```

Constructor Parameters

<i>length</i>	An non-negative integer indicating the number of elements in the array.
<i>element0</i> , ... <i>elementn</i>	One or more values that are assigned as array elements.

Properties

<i>length</i>	See “Array.length Property” on page 118	The number of elements in the array.
---------------	---	--------------------------------------

Methods

<i>concat()</i>	See “Array.concat() Method” on page 116	Concatenate elements to an existing array to create a new array.
<i>join()</i>	See “Array.join() Method” on page 117	Join all elements of the array into a string.
<i>pop()</i>	See “Array.pop() Method” on page 118	Pop the last element in the array (return the value and remove from the array).
<i>push()</i>	See “Array.push() Method” on page 119	Push an array element onto the end of the array (add an element).
<i>reverse()</i>	See “Array.reverse() Method” on page 119	Reverse the order of the elements in the array (last element becomes first; first element becomes last).
<i>shift()</i>	See “Array.shift() Method” on page 120	Same as <i>pop()</i> except the <i>first</i> element is returned and removed from the array.
<i>slice()</i>	See “Array.slice() Method” on page 121	Copy a subset of an existing array to create a new array consisting of just those elements.
<i>sort()</i>	See “Array.sort() Method” on page 122	Sort the elements of the array in place.
<i>splice()</i>	See “Array.splice() Method” on page 124	Add or delete array elements.

<code>toString()</code>	See “Array.toString() Method” on page 125	Convert an array to a string of comma-delimited values (can also be achieved using <code>join()</code> without a parameter).
<code>unshift()</code>	See “Array.unshift() Method” on page 126	Add one or more elements to the beginning of the array and return the new length of the array.

Array.concat() Method

`array.concat(value1, ...valuen)`

Description

The `concat()` method concatenates elements to an existing array to create a new array. The original array is left unmodified. If an array is provided as a parameter to `concat()`, each of its elements are appended as separate array elements at the end of the new array.

Parameters

value1, ...valuen Any number of values to be added to the end of the array. Can also be arrays to be concatenated to the current array.

Returns

A new array formed by the concatenation of the specified values or arrays to the current array.

Example

```
var a=[1,2,3];
b = a.concat(4,5);
c = b.concat([5,6]);
d = c.concat([7,8],[9,10]);
e = 0;
for(i=0; i<d.length;i++)
{
    e = e + d[i];
};
trace(e);//prints 60
```

See Also

“Array.push() Method” on page 119, “Array.pop() Method” on page 118, “Array.shift() Method” on page 120, “Array.unshift() Method” on page 126

Array.join() Method

```
array.join()  
array.join(delimiter)
```

Description

The `join()` method joins all elements of the array into a string; each element is separated by `delimiter`.

Parameters

`delimiter` (*Optional*) Specifies a string to separate each element of the array. If omitted, the array elements are separated with a comma. If omitted, results are the same as those achieved with `array.toString()`.

Returns

The string containing the joined elements and delimiters.

Example

```
baseball = new Array("bat","ball");  
baseballString = baseball.join();  
trace(baseballString);// prints "bat,ball"  
newString = baseball.join(" + ");  
trace(newString);// prints "bat + ball"
```

See Also

“Array.toString() Method” on page 125, “Array.reverse() Method” on page 119, “Array.sort() Method” on page 122

Array.length Property

`array.length`

Description

The `length` property is a positive integer that represents the length of the array. Since array indices start with 0 (zero-based), `length` is one greater than the last index value of the array. `length` is initialized when the array is created.

Example

```
baseball = new Array();  
trace(baseball.length); // prints 0  
moreBaseball = new Array("bat", "ball");  
trace(moreBaseball.length); // prints 2  
moreBaseball[2] = "glove";  
trace(moreBaseball.length); // prints 3
```

Array.pop() Method

`array.pop()`

Description

The `pop` method pops the last element of the array, returns the value of the element, removes the element from the array, and decreases `length` by 1.

Returns

The value of the deleted array element.

Example

```
var stack = [1,2,3];  
trace(stack.pop()); // stack is now [1,2] and pop prints 3
```

See Also

“`Array.push()` Method” on page 119, “`Array.shift()` Method” on page 120, “`Array.unshift()` Method” on page 126, “`Array.concat()` Method” on page 116

Array.push() Method

`array.push(value1, ...valuen)`

Description

The push method appends one or more values onto the end of the array and increases length by *n*.

Parameters

value1, ...valuen Any number of values to be pushed onto the end of the array.

Returns

The new length of the array.

Example

```
var stack = [1,2,3];
trace(stack.push(4,5)); //stack is now [1,2,3,4,5] and push() prints 5
for(i=0; i<stack.length;i++)
{
    trace(stack[i]);
};
//prints
//1
//2
//3
//4
//5
```

See Also

“Array.pop() Method” on page 118, “Array.shift() Method” on page 120, “Array.unshift() Method” on page 126, “Array.concat() Method” on page 116

Array.reverse() Method

`array.reverse()`

Description

The `reverse` method reverses the order of the elements in the array (last element becomes first; first element becomes last).

Example

```
var baseball = ["bat", "ball", "glove", "base"];
for(i=0; (i != 4); ++i)
{
  trace(baseball[i]);
};
//prints
//bat
//ball
//glove
//base
baseball.reverse();
for(i=0; (i != 4); ++i)
{
  trace(baseball[i]);
};
//prints
//base
//glove
//bat
//ball
```

See Also

“`Array.join()` Method” on page 117, “`Array.sort()` Method” on page 122

Array.shift() Method

`array.shift()`

Description

The `shift` method is the same as `pop()` except the *first* element is returned and removed from the array. As a result, the array length is reduced by 1.

Returns

The value of the deleted array element.

Example

```
fish = ["shark", "guppy", "red fish", "blue fish"];
trace(fish.shift()); //prints "shark"
i=0;
while (fish[i] != "blue fish")
{
  trace(fish[i]);
  ++i;
};
trace(fish[i]);
//prints
//guppy
//red fish
//blue fish
```

See Also

“Array.push() Method” on page 119, “Array.pop() Method” on page 118, “Array.unshift() Method” on page 126, “Array.concat() Method” on page 116

Array.slice() Method

```
array.slice(start)
array.slice(start, end)
```

Description

The slice method copies a subset of an existing array to create a new array consisting of just those elements. The new array is a subset of the existing array. *start* and *end* are indices into the array (zero-based). The slice begins with *start* and continues up to, but not including, *end*. If *start* or *end* are negative numbers, the index is equal to the total number of elements in the array minus the number.

Parameters

<i>start</i>	The array index at which to begin the slice. Can also be a negative number.
<i>end</i>	(Optional) The array index at which to end the slice. The slice does not include this element. If this argument is not present, the slice extends all the way to the end of the array. Can also be a negative number.

Returns

A new array that begins with array element `start` and contains all array elements between `start` up to, but not including, array element `end` of the original array.

Example

```
function printArray(arrayId)
{
    for(i=0; i<arrayId.length; i++)
    {
        trace(arrayId[i]);
    }
};
var a = [1,2,3,4,5];
b = a.slice(0,3);
printArray(b);//prints 1,2,3
b = a.slice(3);
printArray(b);//prints 4,5
b = a.slice(1,-1);
printArray(b);//prints 2,3,4
b = a.slice(-3,-2);
printArray(b);//prints 3
```

See Also

“`Array.splice()` Method” on page 124

Array.sort() Method

```
array.sort()
array.sort(userFunction)
```

Description

The `sort` method sorts the elements of *array* in place. If no argument is provided, the elements are sorted in alphabetical order. To sort the array in any other order, you have to supply a function that compares two array elements and returns a value indicating how they should be sorted. For *userFunction(a, b)*, if the return value is:

- less than 0, then *b* is sorted to a lower index than *a*;

- 0, then *a* and *b* are left unchanged with respect to each other, but are sorted with respect to all different elements;
- greater than 0, then *b* is sorted to a higher index than *a*.

Parameters

userFunction (Optional) A user-supplied function that dictates sort order. If omitted, the array is sorted lexicographically (in dictionary order) according to the string conversion of each element.

Example

```
fish = new Array("shark", "guppy", "red fish", "blue fish");
fish.sort();
for(i=0; i != fish.length; ++i)
{
  trace(fish[i]);
};
//prints
//blue fish
//guppy
//red fish
//shark

function numberOrder(a,b) { return a - b; }
a = new Array(33,4,1111,222);
a.sort();
for (i=0;i<a.length;i++) {
  trace(a[i]);
}
a.sort(numberOrder);
for (i=0;i<a.length;i++) {
  trace(a[i]);
}
//prints
//1111
//222
//33
//4
//4
//33
//222
//1111
```

See Also

“Array.join() Method” on page 117, “Array.reverse() Method” on page 119, “Array.join() Method” on page 117

Array.splice() Method

`array.splice(start, num, [val1, ...valn])`

Description

The `splice()` method removes `num` elements from an array. `splice()` optionally inserts new elements starting at zero-based index `start`. To ensure element contiguity, `splice` moves elements up to fill in any gaps.

Parameters

<code>start</code>	The (zero-based) index of first array element to remove. If <code>start</code> is a negative value, <code>start</code> is relative to the end of the array (the index is the number of elements in the array minus the value).
<code>num</code>	(Optional) Number of array elements to remove, including <code>start</code> . If 0, no elements are removed. If <code>num</code> is omitted, all elements from array index <code>start</code> to the end of the array are removed.
<code>[val1,...valn]</code>	(Optional) List of one or more values to be added to the array starting at index <code>start</code> .

Returns

An array consisting of any elements that were spliced from the array.

Example

```
fishAndNumbers = new Array(1,2, "shark", 3, "guppy");
fishAndNumbers.splice(2,2,6,"red fish");
for(i=0; i != fishAndNumbers.length; ++i)
{
  trace(fishAndNumbers[i]);
};
//prints
//1
```

```
//2
//6
//redfish
//guppy

fishAndNumbers = new Array(1,2, "shark", 3, "guppy");
fishAndNumbers.splice(-3,2,6,"red fish");//negative start index
for(i=0; (i != fishAndNumbers.length); ++i)
{
  trace(fishAndNumbers[i]);
};
//prints
//1
//2
//6
//red fish
//guppy
```

See Also

“Array.slice() Method” on page 121

Array.toString() Method

`array.toString()`

Description

The `toString()` method converts an array to a string and returns the string. Yields the same result as the `array.join()` method (when that method is used without a parameter).

Parameters

None

Returns

A comma-separated list of all the elements of the array.

Example

```
fishAndNumbers = new Array(1,2, "shark", 3, "guppy");
trace(fishAndNumbers.toString());//prints "1,2,shark,3,guppy"
```

See Also

“Array.join() Method” on page 117, “Array.reverse() Method” on page 119, “Array.sort() Method” on page 122

Array.unshift() Method

array.unshift(val1, ...valn)

Description

The `unshift()` method adds elements to the beginning of the array.

Parameters

val1, ...valn The values of one or more elements to be added to the beginning of the array, starting at index 0.

Returns

The new array length.

Example

```
fishAndNumbers = new Array(1,2, "shark", 3, "guppy");
trace(fishAndNumbers.unshift(2,6,"red fish")); //prints return value of 8
for(i=0; (i != fishAndNumbers.length); ++i)
{
  trace(fishAndNumbers[i]);
};
//prints
//2
//6
//red fish
//1
//2
//shark
//3
//guppy
```

See Also

“Array.push() Method” on page 119, “Array.pop() Method” on page 118, “Array.shift() Method” on page 120, “Array.concat() Method” on page 116

Boolean() Global Function

Boolean(*value*)

Description

The Boolean() global function converts its parameter to a Boolean value and returns the value.

Parameters

value The value to convert to Boolean.

Returns

The Boolean value of *value* (true or false).

Example

```
var testFalse = 0;
var testTrue = true;
trace(Boolean(0)); //prints "false"
trace(Boolean(1)); //prints "true"
trace(Boolean(true)); //prints "true"
trace(Boolean(false)); //prints "false"
trace(Boolean(testFalse)); //prints "false"
trace(Boolean(testTrue)); //prints "true"
```

Boolean Object

Description

The `Boolean()` function provides support for Boolean values. The `Boolean()` constructor with the `new` operator converts its parameter to a Boolean value and returns a Boolean object wrapper containing the value. This allows the object to inherit the methods of the `Object` object (see “Object Class” on page 236).

Constructor

```
new Boolean()  
new Boolean(value)
```

Parameters

value (Optional) The value that is converted to a Boolean—can be a number, string, Boolean, or object. The values `0`, `NaN`, `null`, the empty string (`""`), and `undefined` all return `false`. All other values return `true`. If this parameter is omitted, the Boolean object is initialized with a value of `false`.

Methods

<code>toString()</code>	See “ <code>Boolean.toString()</code> Method” on page 128	Convert the value of the Boolean object to a string.
<code>valueOf()</code>	See “ <code>Boolean.valueOf()</code> Method” on page 129	Return the primitive Boolean value of the object.

Boolean.toString() Method

```
bool.toString()
```

Description

The `toString()` method returns the string representation of the value of `bool`. The method returns the string `true` if the primitive value of `bool` is `true`; otherwise it returns the string `false`.

Example

```
bool = new Boolean(1);  
trace(bool.toString()); // displays "true"
```

Boolean.valueOf() Method

bool.valueOf()

Description

The `valueOf()` method returns the primitive value of *bool*. The method returns `true` if the primitive value of *bool* is `true`; otherwise it returns `false`.

Example

```
bool = new Boolean("true");  
trace(bool.valueOf()); // displays "false" to the output window
```

Color Object**Description**

The `Color` object allows you to get and set the RGB color values and transformation information for a movie clip. You must create an instance of the `Color` object for a specific target before using any of the `Color` methods.

Constructor

`new Color(target)`

Parameters

<code>target</code>	The movie clip for which an instance of the <code>Color</code> object is created.
---------------------	---

Properties

None.

Methods

<code>getRGB()</code>	See “Color.getRGB() Method” on page 130.	Return the RGB values for the object as a decimal number.
<code>getTransform()</code>	See “Color.getTransform() Method” on page 131.	Return the current offset and percentage values as an object of type <code>Object</code> . For more information on the type <code>Object</code> , see “Object Class” on page 236.
<code>setRGB()</code>	See “Color.setRGB() Method” on page 132.	Set the RGB values for the object expressed as 6 hexadecimal digits.
<code>setTransform()</code>	See “Color.setTransform Method” on page 132.	Set the offset and/or percentage values using an object of type <code>Object</code> . For more information on the type <code>Object</code> , see “Object Class” on page 236.

Color.getRGB() Method

colorObject.getRGB()

Description

The `getRGB()` method returns the RGB values for the object as a decimal number. These are the values that were set by a call to `setRGB()` or when the object was created.

Parameters

None

Returns

A decimal number indicating the RGB value of *colorObject*.

Example

```
redBaseball = new Color(_root.baseball);
redBaseball.setRGB(0xFF0000);
trace(redBaseball.getRGB()); //prints "16711680"
```

See Also

“Color.setRGB() Method” on page 132.

Color.getTransform() Method

colorObject.getTransform()

Description

The getTransform() method returns an object of type Object whose properties are the transformation values set by a call to setTransform().

Parameters

None.

Returns

An object of type Object whose properties contain the transformation values of the movie clip *colorObject*.

Example

```
redFish= new Color(_root.fish);
fishChanger = new Object();
fishChanger.ra = 100;//Red percentage
fishChanger.rb = 200;//Red offset
fishChanger.ga = 0;//Green percentage
fishChanger.gb = 0;//Green offset
fishChanger.ba = 100;//Blue percentage
fishChanger.bb = 50;//Blue offset
fishChanger.aa = 40;//Alpha percentage
fishChanger.ab = -10;//Alpha offset
redFish.setTransform(fishChanger);
fishChanger = redFish.getTransform();
fishChanger.rb = 300;//set the Red offset
fishChanger.ga = 20;//set the Green transformation percentage
redFish.setTransform(fishChanger);//changes the transformation values
```

See Also

“Color.setTransform Method” on page 132, “Object Class” on page 236

Color.setRGB() Method

colorObject.setRGB(0xRRGGBB)

Description

The `setRGB()` method sets the RGB color values for the `Color` object.

Parameters

`0xRRGGBB` .A hexadecimal number (0x) indicating the offsets of each of the color components. It consists of two hexadecimal digits specifying the offset of the red (RR), green (GG), and blue (BB) components.

Example

```
redBaseball = new Color("_root.baseball");
redBaseball.setRGB(0xFF0000);
trace(redBaseball.getRGB()); //prints "16711680"
```

See Also

“`Color.getRGB()` Method” on page 130.

Color.setTransform Method

colorObj.setTransform(transformObj)

Description

The `setTransform()` method sets the color transform information for an object. To use `setTransform()`, you first must create an object of type `Object` (for more information on the type `Object`, see “Object Class” on page 236) with a series of properties, and pass the object as a parameter to `setTransform()`. `setTransform()` uses the values as the new offsets and percentages of `colorObj`. The properties are the following:

- `ra` is the red transformation percentage (-100 to 100)
- `rb` is the red offset (-255 to 255)
- `ga` is the green transformation percentage (-100 to 100)

- gb is the green offset (-255 to 255)
- ba is the blue transformation percentage (-100 to 100)
- bb is the blue offset (-255 to 255)
- aa is the alpha transformation percentage (-100 to 100)
- ab is the alpha offset (-255 to 255)

Parameters

`transformObj` .An object created using the constructor of the generic `Object` object that specifies color transformation values.

Example

```
redFish= new Color(_root.fish);
fishChanger = new Object();
fishChanger.ra = 100;//Red percentage
fishChanger.rb = 200;//Red offset
fishChanger.ga = 0;//Green percentage
fishChanger.gb = 0;//Green offset
fishChanger.ba = 100;//Blue percentage
fishChanger.bb = 50;//Blue offset
fishChanger.aa = 40;//Alpha percentage
fishChanger.ab = -10;//Alpha offset
redFish.setTransform(fishChanger);//sets the new transformation values
```

See Also

“`Color.getTransform()` Method” on page 131.

Date() Global Function

`Date()`

Description

The `Date()` global function returns a string containing the current date, the current time in the local time zone, and the offset in hours between Coordinated Universal Time (UTC—formerly called the Greenwich Mean Time, or GMT) and the local time. For example:

Mon Sep 10, 16:30:29 GMT-0700 2001

Example

```
var now = Date();  
trace(now); //prints string
```

Date Object

Description

The `Date` object allows you to get and set the local date and time or the Coordinated Universal Time (UTC—formerly called the Greenwich Mean Time, or GMT). To call the `Date` object's methods, you must create a new object using the constructor.

All dates and time input are based on (and are as accurate as) the settings of the operating system upon which the Flash player is running.

Constructor

```
new Date()  
new Date(ms)  
new Date(year, month, date, hour, min, sec, ms)
```

Description

You can create a `Date` object in three ways:

- With no arguments. This creates a new `Date` object holding the current date and time based on the local system clock. For example:

```
var now = new Date();  
trace(now.getDate()); //prints the day of the month
```

- With one argument representing milliseconds. This creates a `Date` object holding the number of milliseconds relative to midnight January 1, 1970. For example:

```
var now = new Date(999901885456);  
trace(now.getTime()); //prints 999901885456
```

- With three or more arguments. This creates a `Date` object indicating the year (required), month (required), day (required), hour, minute, second, and millisecond.

```
var now = new Date(99, 11, 31, 9, 52, 54, 999);
trace(now.getFullYear()); //prints 1999
trace(now.getMonth()); //prints 11
trace(now.getDate()); //prints 31
trace(now.getHours()); //prints 9
trace(now.getMinutes()); //prints 52
trace(now.getSeconds()); //prints 54
trace(now.getMilliseconds()); //prints 999
```

Parameters

ms	(Optional) An integer value representing the number of milliseconds since 1 January 1970 00:00:00.
year	The year expressed in four digits—for example, 2001. Alternatively, if you need to indicate a year from 1900 to 1999, specify a value from 0 to 99.
month	An integer value from 0 (Jan.) to 11 (Dec.).
date	An integer value from 1 to 31. If this argument is not supplied, its value is set to 0.
hour	(Optional) An integer value from 0 (midnight) to 23 (11 PM). If this argument is not supplied, its value is set to 0.
min	(Optional) An integer value from 0 to 59. If this argument is not supplied, its value is set to 0.
sec	(Optional) An integer value from 0 to 59. If this argument is not supplied, its value is set to 0.
ms	(Optional) An integer value from 0 to 999. If this argument is not supplied, its value is set to 0.

Methods

getDate()	See “Date.getDate() Method” on Return the day of the month. page 138
getDay()	See “Date.getDay() Method” on Return the day of the week. page 138
getFullYear()	See “Date.getFullYear() Method” Return the year expressed in four-digit format. on page 139
getHours()	See “Date.getHours() Method” Return the hour. on page 139

<code>getMilliseconds()</code>	See "Date.getMilliseconds() Method" on page 140	Return the milliseconds.
<code>getMinutes()</code>	See "Date.getMinutes() Method" on page 140	Return the minutes.
<code>getMonth()</code>	See "Date.getMonth() Method" on page 141	Return the month.
<code>getSeconds()</code>	See "Date.getSeconds() Method" on page 141	Return the seconds.
<code>getTime()</code>	See "Date.getTime() Method" on page 142	Return the number of milliseconds that have passed since January 1, 1970.
<code>getTimezoneOffset()</code>	See "Date.getTimezoneOffset() Method" on page 142	Return the number of minutes between UTC and local time.
<code>getUTCDate()</code>	See "Date.getUTCDate() Method" on page 143	Return the day of the month in UTC.
<code>getUTCDay()</code>	See "Date.getUTCDay() Method" on page 143	Return the day of the week in UTC.
<code>getUTCFullYear()</code>	See "Date.getUTCFullYear() Method" on page 144	Return the year as four-digits in UTC.
<code>getUTCHours()</code>	See "Date.getUTCHours() Method" on page 145	Return the hour in UTC.
<code>getUTCMilliseconds()</code>	See "Date.getUTCMilliseconds() Method" on page 145	Return the milliseconds in UTC.
<code>getUTCMinutes()</code>	See "Date.getUTCMinutes() Method" on page 146	Return the minutes in UTC.
<code>getUTCMonth()</code>	See "Date.getUTCMonth() Method" on page 146	Return the month in UTC.
<code>getUTCSeconds()</code>	See "Date.getUTCSeconds() Method" on page 147	Return the seconds in UTC.
<code>getYear()</code>	See "Date.getYear() Method" on page 147	Return the year relative to 1900.
<code>setDate()</code>	See "Date.setDate() Method" on page 148	Set the day of the month.

<code>setFullYear()</code>	See "Date.setFullYear() Method" Set the year in four-digit format. on page 148
<code>setHours()</code>	See "Date.setHours() Method" on Set the hour of the day. page 149
<code>setMilliseconds()</code>	See "Date.setMilliseconds() Method" Set the milliseconds. on page 150
<code>setMinutes()</code>	See "Date.setMinutes() Method" Set the minutes. on page 151
<code>setMonth()</code>	See "Date.setMonth() Method" Set the month. on page 151
<code>setSeconds()</code>	See "Date.setSeconds() Method" Set the seconds. on page 152
<code>setTime()</code>	See "Date.setTime() Method" on Set the date in number of millisec- page 153 onds that have passed since Janu- ary 1, 1970.
<code>setUTCDate()</code>	See "Date.setUTCDate() Method" Set the day of the month in UTC. on page 153
<code>setUTCFullYear()</code>	See "Date.setUTCFullYear() Method" Set the year in four-digit format in UTC. on page 154
<code>setUTCHours()</code>	See "Date.setUTCHours() Method" Set the hour in UTC. on page 155
<code>setUTCMilliseconds()</code>	See "Date.setUTCMilliseconds() Method" Set the milliseconds in UTC. on page 156
<code>setUTCMinutes()</code>	See "Date.setUTCMinutes() Method" Set the minutes in UTC. on page 157
<code>setUTCMonth()</code>	See "Date.setUTCMonth() Method" Set the month in UTC. on page 157
<code>setUTCSeconds()</code>	See "Date.setUTCSeconds() Method" Set the seconds in UTC. on page 158
<code>setYear()</code>	See "Date.setYear() Method" on Set the year in four-digit format. page 159
<code>toString()</code>	See "Date.toString() Method" on Return the date and time values as page 160 a string.

<code>UTC()</code>	See “ <code>Date.UTC()</code> Method” on page 160	Return the number of milliseconds between January 1, 1970 in UTC and the time specified.
<code>valueOf()</code>	See “ <code>Date.valueOf()</code> Method” on page 162	Return the number of milliseconds that have passed since midnight, January 1, 1970 UTC. Equivalent to <code>getTime()</code> .

Date.getDate() Method

`date.getDate()`

Description

The `getDate()` method returns the day of the month.

Returns

An integer value from 1 to 31.

Example

```
var now = new Date();  
trace(now.getDate()); //prints the day of the month
```

See Also

“`Date.getUTCDate()` Method” on page 143, “`Date.getUTCDay()` Method” on page 143, “`Date.setDate()` Method” on page 148

Date.getDay() Method

`date.getDay()`

Description

The `getDay()` method returns the day of the week.

Returns

An integer from 0 (Sunday) to 6 (Saturday).

Example

```
var now = new Date();  
trace(now.getDay()); //prints the day of the week as an integer
```

See Also

“Date.getUTCDay() Method” on page 143, “Date.setDate() Method” on page 148

Date.getFullYear() Method

date.getFullYear()

Description

The `getFullYear()` method returns the year expressed in four-digit format.

Returns

The year expressed in four digits—for example, 2001.

```
var now = new Date();  
trace(now.getFullYear()); //prints the year in four digits
```

See Also

“Date.getYear() Method” on page 147, “Date.getUTCFullYear() Method” on page 144,
“Date.setFullYear() Method” on page 148

Date.getHours() Method

date.getHours()

Description

The `getHours()` method returns the hour of the day.

Returns

An integer value in the range of 0 (midnight) to 23 (11 PM).

Example

```
var now = new Date();  
trace(now.getHours()); //prints the hour
```

See Also

“Date.getUTCHours() Method” on page 145, “Date.setHours() Method” on page 149

Date.getMilliseconds() Method

date.getMilliseconds()

Description

The `getMilliseconds()` method returns the milliseconds.

Returns

An integer from 0 to 999.

```
var now = new Date();  
trace(now.getMilliseconds()); //prints the milliseconds
```

See Also

“Date.getUTCMilliseconds() Method” on page 145, “Date.setMilliseconds() Method” on page 150

Date.getMinutes() Method

date.getMinutes()

Description

The `getMinutes()` method returns the minutes.

Returns

An integer value in the range 0 to 59.

Example

```
var now = new Date();  
trace(now.getMinutes()); //prints the minutes
```

See Also

“Date.getUTCMinutes() Method” on page 146, “Date.setMinutes() Method” on page 151

Date.getMonth() Method

date.getMonth()

Description

The `getMonth()` method returns the month.

Returns

An integer value from 0 (Jan.) to 11 (Dec.).

Example

```
var now = new Date();  
trace(now.getMonth()); //prints the month as an integer
```

See Also

“Date.getUTCMonth() Method” on page 146, “Date.setMonth() Method” on page 151

Date.getSeconds() Method

date.getSeconds()

Description

The `getSeconds()` method returns the seconds.

Returns

An integer value in the range of 0 to 59.

Example

```
var now = new Date();  
trace(now.getSeconds()); //prints the seconds
```

See Also

“Date.getUTCSeconds() Method” on page 147, “Date.setSeconds() Method” on page 152

Date.getTime() Method

date.getTime()

Description

The `getTime()` method returns the number of milliseconds that have passed since January 1, 1970.

Returns

An integer.

Example

```
var now = new Date();  
trace(now.getTime()); //prints a very large integer
```

See Also

“Date.getUTCHours() Method” on page 145, “Date.setTime() Method” on page 153

Date.getTimezoneOffset() Method

date.getTimezoneOffset()

Description

The `getTimezoneOffset()` method returns the number of minutes between UTC and local time. Accounts for daylight savings time.

Returns

An integer representing the number of minutes.

Example

```
var now = new Date();
trace(now.getTimezoneOffset());
// for California, prints 420 (7 hours) if daylight savings;
// if not daylight savings, prints 480
```

Date.getUTCDate() Method

`date.getUTCDate()`

Description

The `getUTCDate()` method returns the day of the month in UTC.

Returns

An integer value from 1 to 31.

Example

```
var now = new Date();
trace(now.getUTCDate()); //prints the day of the month
```

See Also

“`Date.getDate()` Method” on page 138, “`Date.getUTCDay()` Method” on page 143,
“`Date.setUTCDate()` Method” on page 153

Date.getUTCDay() Method

`date.getUTCDay()`

Description

The `getUTCDay()` method returns the day of the week in UTC.

Returns

An integer from 0 (Sunday) to 6 (Saturday).

Example

```
var now = new Date();  
trace(now.getUTCDay()); //prints the day of the week as an integer
```

See Also

“Date.getDay() Method” on page 138, “Date.getUTCDate() Method” on page 143,
“Date.setUTCDate() Method” on page 153

Date.getUTCFullYear() Method

`date.getUTCFullYear()`

Description

The `getUTCFullYear()` method returns the year as four-digits in UTC.

Returns

The year expressed in four digits—for example, 2001.

Example

```
var now = new Date();  
trace(now.getUTCFullYear()); //prints the year in four digits
```

See Also

“Date.getFullYear() Method” on page 139, “Date.setFullYear() Method” on page 148

Date.getUTCHours() Method

`date.getUTCHours()`

Description

The `getUTCHours()` method returns the hour in UTC.

Returns

An integer value in the range of 0 (midnight) to 23 (11 PM).

Example

```
var now = new Date();  
trace(now.getUTCHours()); //prints the hour
```

See Also

“Date.getHours() Method” on page 139, “Date.setUTCHours() Method” on page 155

Date.getUTCMilliseconds() Method

`date.getUTCMilliseconds()`

Description

The `getUTCMilliseconds()` method returns the milliseconds in UTC.

Returns

An integer from 0 to 999.

Example

```
var now = new Date();  
trace(now.getUTCMilliseconds()); //prints the milliseconds
```

See Also

“Date.getMilliseconds() Method” on page 140, “Date.setUTCMilliseconds() Method” on page 156

Date.getUTCMinutes() Method

date.getUTCMinutes()

Description

The `getUTCMinutes()` method returns the minutes in UTC.

Return

An integer value in the range of 0 to 59.

Example

```
var now = new Date();  
trace(now.getUTCMinutes()); //prints the minutes
```

See Also

“Date.getMinutes() Method” on page 140, “Date.setUTCMinutes() Method” on page 157

Date.getUTCMonth() Method

date.getUTCMonth()

Description

The `getUTCMonth()` method returns the month in UTC.

Returns

An integer value from 0 (Jan.) to 11 (Dec.).

Example

```
var now = new Date();  
trace(now.getUTCMonth()); //prints the month as an integer
```

See Also

“Date.getMonth() Method” on page 141, “Date.setUTCMonth() Method” on page 157

Date.getUTCSeconds() Method

date.getUTCSeconds()

Description

The `getUTCSeconds()` method returns the seconds in UTC.

Returns

An integer value in the range of 0 to 59.

Example

```
var now = new Date();  
trace(now.getUTCSeconds()); //prints the seconds
```

See Also

“Date.getSeconds() Method” on page 141, “Date.setUTCSeconds() Method” on page 158

Date.getYear() Method

date.getYear()

Description

The `getYear()` method returns the year relative to 1900. For example, 101 is returned for the year 2001.

Returns

An integer representing the number of years that have passed since 1900.

Example

```
var now = new Date();  
trace(now.getYear()); //prints current year minus 1900
```

See Also

“Date.getFullYear() Method” on page 139, “Date.getUTCFullYear() Method” on page 144, “Date.setYear() Method” on page 159

Date.setDate() Method

date.setDate(*date*)

Description

The setDate() method sets the day of the month of *date*. This does not affect the system clock or anything else.

Parameters

date An integer value from 1 to 31 indicating the day of the month to set.

Returns

The number of milliseconds between the date set and midnight, January 1, 1970.

Example

```
var now = new Date();
trace(now.setDate(6)); //prints a very large integer
trace(now.getDate()); //prints 6
```

See Also

“Date.getDate() Method” on page 138, “Date.setUTCDate() Method” on page 153

Date.setFullYear() Method

date.setFullYear(*year*, *month*, *date*)

Description

The setFullYear() method sets the year of *date*. The method also sets *month* and *day*, if these optional parameters are specified. This does not affect the system clock or anything else.

Parameters

year	A four-digit integer value indicating the year to set—for example, 2001.
month	<i>(Optional)</i> An integer value from 0 (Jan.) to 11 (Dec.) indicating the month of the year to set.
date	<i>(Optional)</i> An integer value from 1 to 31 indicating the day of the month to set.

Returns

The number of milliseconds between the date set and midnight, January 1, 1970.

Example

```
var now = new Date();
trace(now.setFullYear(2001)); //prints a very large integer
trace(now.getFullYear()); //prints 2001
trace(now.getMonth()); //prints month set by constructor
trace(now.getDate()); //prints day of the month set by constructor
```

See Also

“Date.getUTCFullYear() Method” on page 144, “Date.setUTCFullYear() Method” on page 154, “Date.setYear() Method” on page 159

Date.setHours() Method

date.setHours(hour)

Description

The `setHours()` method sets the hour. This does not affect the system clock or anything else.

Parameters

hour	An integer value from 0 (midnight) to 23 (11 PM) indicating the hour of the day to set.
------	---

Returns

The number of milliseconds between the date set and midnight, January 1, 1970.

Example

```
var now = new Date();  
trace(now.setHours(22)); //prints a very large integer  
trace(now.getHours()); //prints 22
```

See Also

“Date.getHours() Method” on page 139, “Date.setUTCHours() Method” on page 155

Date.setMilliseconds() Method

date.setMilliseconds(ms)

Description

The `setMilliseconds()` method sets the milliseconds. This does not affect the system clock or anything else.

Parameters

`ms` An integer value from 0 to 999 indicating the milliseconds to set.

Returns

The number of milliseconds between the date set and midnight, January 1, 1970.

Example

```
var now = new Date();  
trace(now.setMilliseconds(847)); //prints a very large integer  
trace(now.getMilliseconds()); //prints 847
```

See Also

“Date.getMilliseconds() Method” on page 140, “Date.setUTCMilliseconds() Method” on page 156

Date.setMinutes() Method

date.setMinutes(min)

Description

The `setMinutes()` method sets the minutes. This does not affect the system clock or anything else.

Parameters

min An integer value from 0 to 59 indicating the number of minutes to set.

Returns

The number of milliseconds between the date set and midnight, January 1, 1970.

Example

```
var now = new Date();  
trace(now.setMinutes(59)); //prints a very large integer  
trace(now.getMinutes()); //prints 59
```

See Also

“Date.getMinutes() Method” on page 140, “Date.setUTCMilliseconds() Method” on page 156

Date.setMonth() Method

date.setMonth(month, date)

Description

The `setMonth()` method sets the month. The method also sets *day*, if this optional parameters is specified. This does not affect the system clock or anything else.

Parameters

<i>month</i>	An integer value from 0 (Jan.) to 11 (Dec.) indicating the month to set.
<i>date</i>	(Optional) An integer value from 1 to 31 indicating the day of the month to set.

Returns

The number of milliseconds between the date set and midnight, January 1, 1970.

Example

```
var now = new Date();  
trace(now.setMonth(0, 22)); //prints a very large integer  
trace(now.getMonth()); //prints 0  
trace(now.getDate()); //prints 22
```

See Also

“Date.getMonth() Method” on page 141, “Date.setUTCMonth() Method” on page 157

Date.setSeconds() Method

date.setSeconds(sec)

Description

The `setSeconds()` method sets the seconds. This does not affect the system clock or anything else.

Parameters

<i>sec</i>	An integer value from 0 to 59 indicating the seconds to set.
------------	--

Returns

The number of milliseconds between the date set and midnight, January 1, 1970.

Example

```
var now = new Date();
```

```
trace(now.setSeconds(59)); //prints a very large integer  
trace(now.getSeconds()); //prints 59
```

See Also

“Date.getSeconds() Method” on page 141, “Date.setUTCSeconds() Method” on page 158

Date.setTime() Method

date.setTime(ms)

Description

The `setTime()` method sets the date in number of milliseconds that have passed since January 1, 1970. This does not affect the system clock or anything else.

Parameters

<code>ms</code>	An integer indicating the number of milliseconds between the date to be set and midnight, January 1, 1970.
-----------------	--

Returns

The number of milliseconds set.

Example

```
var now = new Date();  
trace(now.setTime(999930239559)); //prints a very large integer  
trace(now.getTime()); //prints 999930239559
```

See Also

“Date.getTime() Method” on page 142

Date.setUTCDate() Method

date.setUTCDate(date)

Description

The `setUTCDate()` method sets the date of the month in UTC. This does not affect the system clock or anything else.

Parameters

`date` An integer value from 1 to 31 indicating the day to be set.

Returns

The number of milliseconds between the date set and midnight, January 1, 1970, in UTC.

Example

```
var now = new Date();  
trace(now.setUTCDate(2)); //prints a very large integer  
trace(now.getUTCDate()); //prints 2
```

See Also

“Date.getUTCDate() Method” on page 143, “Date.setDate() Method” on page 148

Date.setUTCFullYear() Method

`date.setUTCFullYear(year, month, date)`

Description

The `setUTCFullYear()` method sets the year in UTC, and optionally sets the month and day of the month. This does not affect the system clock or anything else.

Parameters

`year` The year expressed in four digits—for example, 2001.

`month` *(Optional)* An integer from 0 (Jan.) to 11 (Dec.).

`date` *(Optional)* An integer value from 1 to 31.

Returns

The number of milliseconds between the date set and midnight, January 1, 1970, in UTC.

Example

```
var now = new Date();  
trace(now.setUTCFullYear(2001,3,1));//prints a very large integer  
trace(now.getUTCFullYear());//prints 2001  
trace(now.getUTCMonth());//prints 3  
trace(now.getUTCDate());//prints 1
```

See Also

“Date.getUTCFullYear() Method” on page 144, “Date.setFullYear() Method” on page 148

Date.setUTCHours() Method

date.setUTCHours(hour, min, sec, ms)

Description

The `setUTCHours()` method sets the hour of the day in UTC. It also sets the minutes, seconds, and milliseconds if these parameters are supplied. This does not affect the system clock or anything else.

Parameters

<code>hour</code>	An integer value from 0 (midnight) to 23 (11 PM) indicating the hour to be set.
<code>min</code>	<i>(Optional)</i> An integer value from 0 to 59 indicating the number of minutes to set.
<code>sec</code>	<i>(Optional)</i> An integer value from 0 to 59 indicating the number of seconds to set.
<code>ms</code>	<i>(Optional)</i> An integer value from 0 to 999 indicating the milliseconds to set.

Returns

The number of milliseconds between the date set and midnight, January 1, 1970, in UTC.

Example

```
var now = new Date();
trace(now.setUTCHours(22, 45, 46, 888)); //prints a very large integer
trace(now.getUTCHours()); //prints 22
trace(now.getUTCMinutes()); //prints 45
trace(now.getUTCSeconds()); //prints 46
trace(now.getUTCMilliseconds()); //prints 888
```

See Also

“Date.getUTCHours() Method” on page 145, “Date.setHours() Method” on page 149

Date.setUTCMilliseconds() Method

date.setUTCMilliseconds(ms)

Description

The `setUTCMilliseconds()` method sets the milliseconds in UTC. This does not affect the system clock or anything else.

Parameters

ms An integer value in the range of 0 to 999 indicating the number of milliseconds to set.

Returns

The number of milliseconds between the date set and midnight, January 1, 1970, in UTC.

Example

```
var now = new Date();
trace(now.setUTCMilliseconds(220)); //prints a very large integer
trace(now.getUTCMilliseconds()); //prints 220
```

See Also

“Date.getUTCMilliseconds() Method” on page 145, “Date.setMilliseconds() Method” on page 150

Date.setUTCMinutes() Method

date.setUTCMinutes(min, sec, ms)

Description

The `setUTCMinutes()` method sets the minutes in UTC and optionally sets the seconds and milliseconds. This does not affect the system clock or anything else.

Parameters

<i>min</i>	An integer value in the range 0 to 59 indicating the number of minutes to be set.
<i>sec</i>	(Optional) An integer value from 0 to 59 indicating the number of seconds to set.
<i>ms</i>	(Optional) An integer value from 0 to 999 indicating the milliseconds to set.

Returns

The number of milliseconds between the date set and midnight, January 1, 1970, in UTC.

Example

```
var now = new Date();
trace(now.setUTCMinutes(45, 47, 889)); //prints a very large integer
trace(now.getUTCMinutes()); //prints 45
trace(now.getUTCSeconds()); //prints 47
trace(now.getUTCMilliseconds()); //prints 889
```

See Also

“Date.getUTCMinutes() Method” on page 146, “Date.setMinutes() Method” on page 151

Date.setUTCMonth() Method

date.setUTCMonth(month, date)

Description

The `setUTCMonth()` method sets the month in UTC. It also optionally sets the day of the month. This does not affect the system clock or anything else.

Parameters

<code>month</code>	An integer value in the range 0 (Jan.) to 11 (Dec.) indicating the month to set.
<code>date</code>	(Optional) An integer value from 1 to 31 indicating the day of the month.

Returns

The number of milliseconds between the date set and midnight, January 1, 1970, in UTC.

Example

```
var now = new Date();
trace(now.setUTCMonth(11, 31)); //prints a very large integer
trace(now.getUTCMonth()); //prints 11
trace(now.getUTCDate()); //prints 31
```

See Also

“`Date.getUTCMonth()` Method” on page 146, “`Date.setMonth()` Method” on page 151

Date.setUTCSeconds() Method

`date.setUTCSeconds(sec, ms)`

Description

The `setUTCSeconds()` sets the seconds in UTC. It also optionally sets the milliseconds. This does not affect the system clock or anything else.

Parameters

<i>sec</i>	An integer value in the range 0 to 59 indicating the number of seconds to set.
<i>ms</i>	(Optional) An integer value from 0 to 999 indicating the milliseconds to set.

Returns

The number of milliseconds between the date set and midnight, January 1, 1970, in UTC.

Example

```
var now = new Date();  
trace(now.setUTCSeconds(44, 310)); //prints a very large integer  
trace(now.getUTCSeconds()); //prints 44  
trace(now.getUTCMilliseconds()); //prints 310
```

See Also

“Date.getUTCSeconds() Method” on page 147, “Date.setSeconds() Method” on page 152

Date.setYear() Method

date.setYear(year, month, date)

Description

The `setYear()` method sets the year, and optionally the month and day of the month. This does not affect the system clock or anything else.

Parameters

<i>year</i>	An integer value indicating the year to set. The method interprets a 1- or 2-digit value to mean the 1900s—for example, 13 is interpreted to mean 1913.
<i>month</i>	(Optional) An integer value in the range of 0 (Jan.) to 11 (Dec.) indicating the month to set. If this argument is not supplied, its value is set to 0.
<i>date</i>	(Optional) An integer value in the range of 1 to 31 indicating the day to be set for date. If this argument is not supplied, its value is set to 0.

Returns

The number of milliseconds between the date set and midnight, January 1, 1970.

Example

```
var now = new Date();  
trace(now.setYear(2001,3,1));//prints a very large integer  
trace(now.getFullYear());//prints 2001  
trace(now.getMonth());//prints 3  
trace(now.getDate());//prints 1
```

See Also

“Date.getYear() Method” on page 147, “Date.setFullYear() Method” on page 148,
“Date.setUTCFullYear() Method” on page 154

Date.toString() Method

date.toString()

Description

The `toString()` method returns the date and time values as a string.

Returns

The following string is an example of the format returned by this method:

Mon Aug 13, 10:54:21 GMT-0700 2001

Example

```
var now = new Date();  
trace(now.toString());//string with the date
```

Date.UTC() Method

Date.UTC(year, month, date, hour, min, sec, ms)

Description

The `Date.UTC()` method returns the date as the number of milliseconds between the time specified (passed in as the arguments to the method) and midnight, January 1, 1970, in UTC. The first three parameters are required. `Date.UTC()` and `Date()` accept the same arguments; the only difference between the two is that the new date object created using `Date.UTC()` assumes UTC while the new date object created using only `Date()` assumes local time. A new UTC date object is normally created like this:

```
now = new Date(Date.UTC(2001, 9, 30));
```

In addition, `Date.UTC()` is commonly used with the `setTime()` method.

Parameters

<i>year</i>	The year expressed in four digits— for example, 2001. To indicate for a year from 1900 to 1999, you can specify a value from 0 to 99.
<i>month</i>	An integer value from 0 (Jan.) to 11 (Dec.).
<i>date</i>	An integer value from 1 to 31.
<i>hour</i>	(Optional) An integer value in the range of 0 (midnight) to 23 (11 PM).
<i>min</i>	(Optional) An integer value in the range of 0 to 59.
<i>sec</i>	(Optional) An integer value in the range of 0 to 59.
<i>ms</i>	(Optional) An integer value in the range of 0 to 999.

Returns

The number of milliseconds between the date set and midnight, January 1, 1970, in UTC.

Example

```
var now = new Date(Date.UTC(96, 11, 29, 11, 58, 59, 345));
trace(now.getTime()); //prints milliseconds
trace(now.getUTCFullYear()); //prints 1996
trace(now.getMonth()); //prints 11
trace(now.getUTCDate()); //prints 29
trace(now.getUTCHours()); //prints 11
trace(now.getUTCMinutes()); //prints 58
trace(now.getUTCSeconds()); //prints 59
trace(now.getUTCMilliseconds()); //prints 345
```

Date.valueOf() Method

date.valueOf()

Description

The `valueOf()` method returns the number of milliseconds that have passed since midnight, January 1, 1970 UTC. Equivalent to `getTime()`.

Returns

An integer.

Example

```
var now = new Date();  
trace(now.valueOf()); //prints the number of milliseconds
```

See Also

“Date.getTime() Method” on page 142

DuplicateMovieClip() Global Function

`duplicateMovieClip(target, newName, depth)`

Description

The `duplicateMovieClip()` global function creates a duplicate of `target` while `target` is playing. The duplicated movie clip always starts at its frame 1 regardless of `target`'s frame at the time of duplication. The duplicated movie clip inherits shape transformations but not the timeline variables. The duplicated movie clip is placed in `target`'s parent's programmatic stack. A programmatic stack holds child movie clips; when you duplicate a movie clip it will have the same parent as the original, and thus “live” in the parent's programmatic stack.

The `removeMovieClip()` global function is used to delete duplicated movie clips. `MovieClip.removeMovieClip()` can also be used by duplicated movie clips to delete themselves. Duplicated movie clips can also be removed by placing another movie clip at the same depth in the programmatic stack.

Parameters

<i>target</i>	The movie clip that is duplicated.
<i>newName</i>	The name of the duplicate movie clip. This must be a unique name.
<i>depth</i>	The depth of the movie clip in <i>target</i> 's parent's programmatic stack.

Example

```
duplicateMovieClip (_root.baseball, "newBaseball", 1); //creates new baseball
_root.newBaseball._x += 25; //moves new baseball along x axis
_root.newBaseball._y += 25; //moves new baseball along y axis
```

See Also

“RemoveMovieClip() Global Function” on page 242, “MovieClip.duplicateMovieClip() Method” on page 207, “MovieClip.removeMovieClip() Method” on page 220

Escape() Global Function

`escape(string)`

Description

The `escape()` global function creates a URL-encoded string from *string*. In the new string, characters of *string* that require URL encoding are replaced with the format `%xx`, where *xx* is the hexadecimal value of the character. This format is used to transmit information appended to a URL during, for example, execution of the GET method. Use the `unescape()` global function to translate the string back into its original format.

Parameters

<i>string</i>	The string to be encoded.
---------------	---------------------------

Example

```
//prints Billy%20went%20fishing%21%24%23%21
trace(escape("Billy went fishing!$#!"));
```

See Also

“Unescape() Global Function” on page 271

Eval() Global Function

`eval(expression)`

Description

The `eval()` global function returns the value of or a reference to *expression*.

Note: This implementation of eval() is different from JavaScript's implementation.

Parameters

expression An expression that evaluates to a variable, property, object, or movie clip.

Returns

If *expression* is a variable or property, the value of the variable or property is returned. If *expression* is an object or movie clip, a reference to the movie clip or object is returned.

Example

```
x=4;
trace(eval(x)); //prints 4
str = "baseball";
hitBaseball = eval("_root."+str);
hitBaseball._x += 50; //moves movie clip 50 pixels
```

Focusrect Global Property

`_focusrect`

Description

The `_focusrect` global property is a Boolean that specifies whether the button or text field that currently has focus has a yellow rectangle that appears around it. As a Boolean, it can be assigned only one of two values: `true` or `false`. If assigned `true`, the yellow rectangle appears; if `false`, it does not. This property can be read or written.

Fscommand() Global Function

`fscommand(string command, string arguments)`

Description

The `fscommand()` global function is used only within the context of `getURL().fscommand()` communication is supported only on the Microsoft Windows operating system—primarily on the Internet Explorer web browser. With version 6 of the Netscape web browser, `fscommand()` is no longer supported.

Parameters

<i>string command</i>	The command to execute, in quotes.
<i>string arguments</i>	The arguments for the command, in quotes.

See Also

“GetURL Global Function” on page 166

GetTimer Global Function

`getTimer()`

Description

The `getTimer()` global function gets the number of milliseconds that have elapsed since the movie started playing.

Parameters

None.

Returns

The elapsed time in milliseconds.

GetURL Global Function

```
getURL(url)  
getURL(url, target)  
getURL(url, target, howToSendVariables)
```

Description

The `getURL()` global function gets a document from a specified URL and loads it into the browser at the specified target. It is also used to execute a script on a server and receive the results in a web browser window or frame. Additionally, it can be used to execute JavaScript (`"javascript:command"`) or VBScript (`"vbscript:command"`) in a web browser, and provides support for the `fscommand()` global function.

Note: This method is not supported in preview mode.

The `fscommand()` options are as follows:

- `getURL("fscommand: allowscale", value)`—Tells the stand-alone SWF player whether its contents should scale with the size of the player's window. *value* is the string `"true"` or `"false"`, indicating whether or not (respectively) the contents of the SWF player should scale.
- `getURL("fscommand: exec", applicationName)`—Tells the stand-alone SWF player to launch an external application. *applicationName* is a string showing an absolute path to the application.
- `getURL("fscommand: fullscreen", value)`—Tells the stand-alone SWF player whether to maximize, filling the entire screen. *value* is the string `"true"` or `"false"`, indicating whether or not (respectively) to maximize.
- `getURL("fscommand: quit")`—Tells the stand-alone SWF player to quit.
- `getURL("fscommand: showmenu", value)`—Tells the stand-alone SWF player whether to suppress the display of the controls in the context menu. *value* is the string `"true"` or `"false"`, indicating whether or not (respectively) to suppress.

- `getURL("fscommand: trapallkeys", value)`—Tells the stand-alone SWF player whether to send all keystrokes to the SWF file(s) executing in the player. *value* is the string "true" or "false", indicating whether or not (respectively) to send.

Parameters

<i>url</i>	A string specifying the URL to which to hyperlink (HTTP or FTP). This may be a relative or an absolute pathname. It can be the name of a document or it can be a script, and the <code>fscommand()</code> global function can be used here.
<i>target</i>	(Optional) A string specifying the target frame in the browser—e.g., <code>_self</code> (the default), <code>_parent</code> , <code>_top</code> , <code>_blank</code> . If omitted, <code>_self</code> is used. Custom names can also be used.
<i>howToSendVariables</i>	(Optional) Omit this parameter if you don't want to send the variables. This parameter is a string literal. Specify <code>GET</code> to send variables via <code>get</code> (i.e., tacked onto the end of the URL) or <code>POST</code> to send them with <code>post</code> (i.e., put into the body of the request). Both methods send them in application/x-www-form-urlencoded MIME format. All user-defined variables are sent, except for user-defined standard handlers.

Example

```
getURL("ftp://download.intel.com");  
getURL("http://www.adobe.com", "_parent");  
getURL("file:///C:/coolestFile.html");  
getURL("javascript: alert(\"Hi\");");
```

See Also

“`LoadVariables()` Global Function” on page 185, “`MovieClip.getURL()` Method” on page 211, “`MovieClip.loadVariables()` Method” on page 216, “`Fscommand()` Global Function” on page 165

GetVersion() Global Function

`getVersion()`

Description

The `getVersion()` global function returns, in string form, the version of Flash that the user currently has installed. The first number refers to the major version number of Flash; the second number gives the minor version; the third number is the build (revision); and the fourth number is the patch.

For example, from LiveMotion's preview mode:

```
LM 5,0,42,0
```

For example, from an exported SWF file (on a Windows machine):

```
WIN 5,0,30,0
```

Parameters

None

Returns

The version of Flash installed on the user's system.

GotoAndPlay() Global Function

```
gotoAndPlay(label)
```

Description

The `gotoAndPlay()` global function sends the main timeline's playhead to the specified *label* and continues playing from *label*.

Note: Frame numbers should not be passed to this global function. The use of labels is recommended.

Parameters

<i>label</i>	Destination of the playhead.
--------------	------------------------------

See Also

"GotoAndStop() Global Function" on page 169

GotoAndStop() Global Function

`gotoAndStop(label)`

Description

The `gotoAndStop()` global function sends the main timeline's playhead to the specified *label* and stops playing at *label*.

Note: Frame numbers should not be passed to this global function. The use of labels is recommended.

Parameters

<i>label</i>	Destination of the playhead.
--------------	------------------------------

See Also

"`MovieClip.gotoAndPlay()` Method" on page 213

Highquality Global Property

`_highquality`

Description

The `_highquality` global property determines the level of anti-aliasing present in the movie clip. This property can be read or written. One of three values is possible, as follows:

- 0 = Turns off anti-aliasing.
- 1 = Anti-aliasing applied. (Default)
- 2 = Highest level of anti-aliasing (bitmap smoothing).

See Also

"Quality Global Property" on page 242

Infinity Global Property

Infinity

Description

The `Infinity` global property is a predefined variable with the value for infinity. It is any value larger than `Number.MAX_VALUE`, which is the largest number that can be represented in JavaScript.

See Also

“-Infinity Global Property” on page 170, “`Number.POSITIVE_INFINITY` Property” on page 234, “`Number.MAX_VALUE` Property” on page 232

-Infinity Global Property

-Infinity

Description

The `-Infinity` global property is a predefined variable with the value of `-infinity`.

See Also

“Infinity Global Property” on page 170, “`Number.NEGATIVE_INFINITY` Property” on page 233

isFinite Global Function

`isFinite(expression)`

Description

The `isFinite()` global function evaluates an expression and returns `true` if the expression is a finite number. Otherwise, it returns `false`—the value is infinity or negative infinity

Parameters

expression Any valid JavaScript expression.

Returns

true if the expression is a finite number, false otherwise.

See Also

“Infinity Global Property” on page 170, “-Infinity Global Property” on page 170

isNaN() Global Function

isNaN(expression)

Description

The `isNaN()` global function returns true if the expression is Not-a-Number (NaN).

Parameters

expression Any valid JavaScript expression.

Returns

true if the expression is not a number (NaN), false otherwise.

See Also

“Number.NaN Property” on page 233

Key Object**Description**

The Key object used to retrieve the state of the keyboard. The Key object and its constants and methods are static—you do not create Key objects using a constructor.

Constants

BACKSPACE	See "Key.BACKSPACE Constant" on page 174	Passed to Key . isDown () to determine whether BACKSPACE key is pressed. Constant representing the value for BACKSPACE.
CAPSLCK	See "Key.CAPSLCK Constant" on page 174	Passed to Key . isDown () to determine whether CAPSLCK key is pressed. Constant representing the value for CAPSLCK.
CONTROL	See "Key.CONTROL Constant" on page 174	Passed to Key . isDown () to determine whether CONTROL key is pressed. Constant representing the value for the CONTROL key.
DELETEKEY	See "Key.DELETEKEY Constant" on page 175	Passed to Key . isDown () to determine whether DELETEKEY key is pressed. Constant representing the value for DELETEKEY.
DOWN	See "Key.DOWN Constant" on page 175	Passed to Key . isDown () to determine whether DOWN key is pressed. Constant representing the value for the DOWN key.
END	See "Key.END Constant" on page 175	Passed to Key . isDown () to determine whether END key is pressed. Constant representing the value for the END key.
ENTER	See "Key.ENTER Constant" on page 176	Passed to Key . isDown () to determine whether ENTER key is pressed. Constant representing the value for the ENTER key.
ESCAPE	See "Key.ESCAPE Constant" on page 176	Passed to Key . isDown () to determine whether ESCAPE key is pressed. Constant representing the value for the ESCAPE key.
HOME	See "Key.HOME Constant" on page 177	Passed to Key . isDown () to determine whether HOME key is pressed. Constant representing the value for HOME key.
INSERT	See "Key.INSERT Constant" on page 178	Passed to Key . isDown () to determine whether INSERT key is pressed. Constant representing the value for the INSERT key.
LEFT	See "Key.LEFT Constant" on page 180	Passed to Key . isDown () to determine whether LEFT key is pressed. Constant representing the value for the LEFT key.

PGDN	See "Key.PGDN Constant" on page 180	Passed to Key.isDown() to determine whether PGDN key is pressed. Constant representing the value for the PGDN key.
PGUP	See "Key.PGUP Constant" on page 180	Passed to Key.isDown() to determine whether PGUP key is pressed. Constant representing the value for the PGUP key.
RIGHT	See "Key.RIGHT Constant" on page 181	Passed to Key.isDown() to determine whether RIGHT key is pressed. Constant representing the value for the RIGHT key.
SHIFT	See "Key.SHIFT Constant" on page 181	Passed to Key.isDown() to determine whether SHIFT key is pressed. Constant representing the value for the SHIFT key.
SPACE	See "Key.SPACE Constant" on page 181	Passed to Key.isDown() to determine whether SPACE bar is pressed. Constant representing the value for the SPACE bar.
TAB	See "Key.TAB Constant" on page 182	Passed to Key.isDown() to determine whether TAB key is pressed. Constant representing the value for the TAB key.
UP	See "Key.UP Constant" on page 182	Passed to Key.isDown() to determine whether UP key is pressed. Constant representing the value for the UP key.

Methods

getAscii()	See "Key.getAscii() Method" on page 176	Get the ASCII code of the last key pressed.
getCode()	See "Key.getCode() Method" on page 177	Get the key code of the last key pressed.
isDown()	See "Key.isDown() Method" on page 178	Check whether the specified key is currently down.
isToggled()	See "Key.isToggled() Method" on page 179	Check whether the num lock or caps lock key is toggled on.

Key.BACKSPACE Constant

Key.BACKSPACE

Description

The Key.BACKSPACE constant is passed to Key.isDown() to determine whether the BACKSPACE key is pressed. It is returned by Key.getCode() if the BACKSPACE key was last key pressed.

See Also

“Key.getCode() Method” on page 177, “Key.isDown() Method” on page 178

Key.CAPSLOCK Constant

Key.CAPSLOCK

Description

The Key.CAPSLOCK is passed to Key.isToggled to determine whether the CAPSLOCK key is on. It is returned by Key.getCode() if CAPSLOCK key was last key pressed.

See Also

“Key.isToggled() Method” on page 179

Key.CONTROL Constant

Key.CONTROL

Description

The Key.CONTROL constant is passed to Key.isDown() to determine whether the CONTROL key is pressed. It is returned by Key.getCode() if CONTROL key was last key pressed.

See Also

“Key.getCode() Method” on page 177, “Key.isDown() Method” on page 178

Key.DELETEKEY Constant

`Key.DELETEKEY`

Description

The `Key.DELETEKEY` constant is passed to `Key.isDown()` to determine whether the `DELETEKEY` key is pressed. It is returned by `Key.getCode()` if the `DELETEKEY` key was last key pressed.

See Also

“`Key.getCode()` Method” on page 177, “`Key.isDown()` Method” on page 178

Key.DOWN Constant

`Key.DOWN`

Description

The `Key.DOWN` constant is passed to `Key.isDown()` to determine whether the `DOWN` key is pressed. It is returned by `Key.getCode()` if the `DOWN` key was last key pressed.

See Also

“`Key.getCode()` Method” on page 177, “`Key.isDown()` Method” on page 178

Key.END Constant

`Key.END`

Description

The `Key.END` constant is passed to `Key.isDown()` to determine whether the `END` key is pressed. It is returned by `Key.getCode()` if the `END` key was last key pressed.

See Also

“`Key.getCode()` Method” on page 177, “`Key.isDown()` Method” on page 178

Key.ENTER Constant

`Key. ENTER`

Description

The `Key. ENTER` constant is passed to `Key. isDown()` to determine whether the `ENTER` key is pressed. It is returned by `Key. getCode()` if the `ENTER` key was last key pressed.

See Also

“`Key.getCode()` Method” on page 177, “`Key.isDown()` Method” on page 178

Key.ESCAPE Constant

`Key. ESCAPE`

Description

The `Key. ESCAPE` constant is passed to `Key. isDown()` to determine whether the `ESCAPE` key is pressed. It is returned by `Key. getCode()` if the `ESCAPE` key was last key pressed.

See Also

“`Key.getCode()` Method” on page 177, “`Key.isDown()` Method” on page 178

Key.getAscii() Method

`Key.getAscii()`

Description

The `Key.getAscii()` method returns the ASCII code of the last key pressed.

Arguments

None

Example

In the `onKeyUp` or `onKeyDown` event:

```
var asciiVal = Key.getAscii();
if (asciiVal == 102)
{
    trace("Lower case 'f' has been pressed");
};
```

See Also

“`Key.getCode()` Method” on page 177

Key.getCode() Method

`Key.getCode()`

Description

The `Key.getCode()` method returns the key code of the last key pressed.

Parameters

None

Example

In the `onKeyUp` or `onKeyDown` event:

```
if (Key.getCode() == Key.ESCAPE)
{
    trace("Key.ESCAPE was pressed.");
};
```

See Also

“`Key.getAscii()` Method” on page 176

Key.HOME Constant

`Key.HOME`

Description

The `Key.HOME` constant is passed to `Key.isDown()` to determine whether the HOME key is pressed. It is returned by `Key.getCode()` if the HOME key was last key pressed.

See Also

“`Key.getCode()` Method” on page 177, “`Key.isDown()` Method” on page 178

Key.INSERT Constant

`Key.INSERT`

Description

The `Key.INSERT` constant is passed to `Key.isDown()` to determine whether INSERT key is pressed. It is returned by `Key.getCode()` if the INSERT key was last key pressed.

See Also

“`Key.getCode()` Method” on page 177, “`Key.isDown()` Method” on page 178

Key.isDown() Method

`Key.isDown(keycode)`

Description

The `Key.isDown()` method is used to check whether the specified key is currently down.

Parameters

keycode The key code to check for.

Returns

`true` if the key is pressed; `false` otherwise.

Example

In the `onKeyUp` or `onKeyDown` event:

```
if (Key.isDown(key.RIGHT))
{
    trace("Right arrow key was pressed.");
};
```

See Also

“`Key.isToggled()` Method” on page 179

Key.isToggled() Method

`Key.isToggled(keycode)`

Description

The `Key.isToggled()` method is used to see if the caps lock or num lock key is on.

Parameters

keycode

If this parameter is `Key.CAPSLOCK` or the integer 20, then the method checks for whether the caps lock key is toggled on. If the parameter is the integer 144, then the method checks for whether the num lock key is toggled on.

Returns

true if the num lock or caps lock key is toggled on; false otherwise.

Example

In the `onKeyUp` or `onKeyDown` event:

```
if (Key.isToggled(20))//detect whether caps lock key is toggled on
{
    trace("Caps lock key is on.");
};
```

See Also

“Key.isDown() Method” on page 178

Key.LEFT Constant

Key.LEFT

Description

The Key.LEFT is passed to Key.isDown() to determine whether the LEFT key is pressed. It is returned by Key.getCode() if the LEFT key was last key pressed.

See Also

“Key.getCode() Method” on page 177, “Key.isDown() Method” on page 178

Key.PGDN Constant

Key.PGDN

Description

The Key.PGDN is passed to Key.isDown() to determine whether the PGDN key is pressed. It is returned by Key.getCode() if the PGDN key was last key pressed.

See Also

“Key.getCode() Method” on page 177, “Key.isDown() Method” on page 178

Key.PGUP Constant

Key.PGUP

Description

The Key.PGUP constant is passed to Key.isDown() to determine whether the PGUP key is pressed. It is returned by Key.getCode() if the PGUP key was last key pressed.

See Also

“Key.getCode() Method” on page 177, “Key.isDown() Method” on page 178

Key.RIGHT Constant

Key.RIGHT

Description

The Key.RIGHT constant is passed to Key.isDown() to determine whether the RIGHT key is pressed. It is returned by Key.getCode() if the RIGHT key was last key pressed.

See Also

“Key.getCode() Method” on page 177, “Key.isDown() Method” on page 178

Key.SHIFT Constant

Key.SHIFT

Description

The Key.SHIFT constant is passed to Key.isDown() to determine whether the SHIFT key is pressed. It is returned by Key.getCode() if the SHIFT key was last key pressed.

See Also

“Key.getCode() Method” on page 177, “Key.isDown() Method” on page 178

Key.SPACE Constant

Key.SPACE

Description

The Key.SPACE constant is passed to Key.isDown() to determine whether the SPACE key is pressed. It is returned by Key.getCode() if the SPACE key was last key pressed.

See Also

“Key.getCode() Method” on page 177, “Key.isDown() Method” on page 178

Key.TAB Constant

Key.TAB

Description

The Key.TAB constant is passed to Key.isDown() to determine whether the TAB key is pressed. It is returned by Key.getCode() if the TAB key was last key pressed.

See Also

“Key.getCode() Method” on page 177, “Key.isDown() Method” on page 178

Key.UP Constant

Key.UP

Description

The Key.UP constant is passed to Key.isDown() to determine whether the UP key is pressed. It is returned by Key.getCode() if the UP key was last key pressed.

See Also

“Key.getCode() Method” on page 177, “Key.isDown() Method” on page 178

LevelN Global Property

_levelN

Description

The `_leveln` global property is used to specify the level into which to load a movie clip using the `loadMovie()` global function. It is also used to refer to that movie clip after it has been loaded. The root level movie clip loads at level 0 by default.

Note: This global property is not supported in preview mode (except for `_level0`).

Example

```
loadMovie("http://devtech.corp.adobe.com/livemotion/test.swf", "_level1");  
_level1.stop();
```

See Also

“LoadMovie() Global Function” on page 184

lmFrameOfLabel() Global Function

`lmFrameOfLabel(label)`

Description

The `lmFrameOfLabel()` global function returns the frame number at which *label* resides.

Parameters

<i>label</i>	String literal name of a label on the composition timeline. Must appear in quotes.
--------------	---

Returns

The frame number associated with *label*, or 0 if *label* is not found on the composition timeline.

Example

```
//returns frame number of "firstThrow" label  
lmFrameOfLabel("firstThrow");
```

LoadMovie() Global Function

```
loadMovie(url, target)  
loadMovie(url, target, howToSendVariables)
```

Description

The `loadMovie()` global function loads additional SWF files into the Flash player. These SWF files can be loaded into “higher” levels (the main movie “lives” at `_level0`), or they can be loaded into existing movie clips, replacing those movie clips.

If a new main movie clip is loaded at level 0, every level is unloaded and the effect is the same as starting a new SWF file in the Flash player. The movie clip loaded in level 0 sets the frame rate, background color, and frame size for all other loaded movie clips.

Movie clips loaded with the `loadMovie()` global function can be unloaded using the `unloadMovie()` global function or the `unloadMovieNum()` global function. Likewise, a new movie clip can be loaded into the level using the `loadMovie()` or `loadMovieNum()` global function.

Note: This method is not supported in preview mode.

Parameters

<i>url</i>	The URL from which to load the SWF file. The URL must be in the same sub-domain as the URL where the movie clip currently resides.
<i>target</i>	String representing the name of another movie clip that the new movie will replace, or the document level. The loaded movie inherits the position, scaling, and rotation of the movie it's replacing.
<i>howToSendVariables</i>	(Optional) GET or POST. String literal that specifies the method for sending variables to the server. The GET method appends them to the URL; the POST method sends them in a separate HTTP packet.

Example

```
loadMovie("http://devtech.corp.adobe.com/docs/livemotion/billys.swf",  
"_level1");
```

See Also

“LoadMovieNum() Global Function” on page 185, “UnloadMovie() Global Function” on page 271, “UnloadMovieNum() Global Function” on page 272, “MovieClip.loadMovie() Method” on page 216

LoadMovieNum() Global Function

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

Description

The `loadMovieNum()` global function is the same as `loadMovie()` except that the second parameter must be specified as a number rather than as a string. With `loadMovieNum()` you cannot specify the name of another movie clip to be replaced.

Note: This method is not supported in preview mode.

Parameters

<i>url</i>	The URL from which to load the movie.
<i>level</i>	Document level.
<i>method</i>	(Optional)String literal. GET or POST.

See Also

“LoadMovie() Global Function” on page 184, “UnloadMovie() Global Function” on page 271, “UnloadMovieNum() Global Function” on page 272

LoadVariables() Global Function

```
loadVariables(url, target)  
loadVariables(url, target, howToSendVariables)
```

Description

The `loadVariables()` global function loads variables fetched from the specified URL. The movie clip's `onData` event handler is called when the variables have been loaded. The data that's loaded has the same scope as the movie clip/level that it's loaded into. All the values loaded are considered the string data type.

The data fetched from the URL must be in the `application/x-www-form-urlencoded` MIME format.

Parameters

<i>url</i>	The URL from which to get the variables. The host for the URL must be in the same subdomain as the movie clip when accessed using a web browser.
<i>target</i>	The target to load the variables to. There are three possibilities: • if a number, the variables are loaded into the level the number specifies; • if a string, the variables are loaded into the movie clip specified by the path in the string; • if a movie clip object, the variables are loaded into it.
<i>howToSendVariables</i>	(Optional) Omit this parameter if you don't want to send the variables. GET or POST. String literal that specifies the method for sending variables to the movie clip that is to be loaded. The GET method appends them to the URL; the POST method sends them in a separate HTTP packet. Using either method they're sent in <code>application/x-www-form-urlencoded</code> MIME format. All user defined variables are sent, except for user-defined standard handlers.

Example

```
//last two arguments are optional placeholders used instead  
loadVariables("http://pdcmotion1.corp.adobe.com/cgi-  
bin/stockdata.pl?ticker=" + this.symbol,this,"DONT_SEND");
```

See Also

“`LoadVariablesNum()` Global Function” on page 187, “`GetURL` Global Function” on page 166, “`MovieClip.getURL()` Method” on page 211, “`MovieClip.loadVariables()` Method” on page 216

LoadVariablesNum() Global Function

```
loadVariablesNum (url, target)  
loadVariablesNum (url, target, howToSendVariables)
```

Description

The `loadVariablesNum()` global function is the same as `loadVariables()` except the second argument must be a level number.

Parameters

<i>url</i>	The URL from which to get the variables.
<i>level</i>	The level number of the target to which to load the variables.
<i>howToSendVariables</i>	(Optional) String literal. GET or POST.

See Also

“LoadVariables() Global Function” on page 185, “GetURL Global Function” on page 166, “LoadMovie() Global Function” on page 184, “LoadMovieNum() Global Function” on page 185

Math Object

Description

The `Math` object has constants and methods to facilitate use of constants and common mathematical functions. The `Math` object and its constants and methods are static—you do not create `Math` objects using a constructor. For example, you refer to the constant `PI` as `Math.PI` and you call the sine function as `Math.sin(x)`, where `x` is the method’s argument. Constants are defined with the full precision of real numbers.

Constants

E	See “Math.E Constant” on page 193.	Euler’s constant and the base of natural logarithms (approximately 2.718).
LN2	See “Math.LN2 Constant” on page 194.	Natural logarithm of 2 (approximately 0.693).
LN10	See “Math.LN10 Constant” on page 194.	Natural logarithm of 10 (approximately 2.302).
LOG2E	See “Math.LOG2E Constant” on page 195.	NBase 2 logarithm of E (approximately 1.442).
LOG10E	See “Math.LOG10E Constant” on page 195.	Base 10 logarithm of E (approximately 0.434).
PI	See “Math.PI Constant” on page 196.	Ratio of the circumference of a circle to its diameter (approximately 3.14159).
SQRT1_2	See “Math.SQRT1_2 Constant” on page 198.	Square root of 1/2; equivalently, 1 over the square root of 2 (approximately 0.707).
SQRT2	See “Math.SQRT2 Constant” on page 198.	Square root of 2 (approximately 1.414).

Methods

abs()	See “Math.abs() Method” on page 189.	Return the absolute value of a number.
acos()	See “Math.acos() Method” on page 190.	Return the arccosine (in radians) of a number.
asin()	See “Math.asin() Method” on page 190.	Return the arcsine (in radians) of a number.
atan()	See “Math.atan() Method” on page 191.	Return the arctangent (in radians) of a number.
atan2()	See “Math.atan2() Method” on page 191.	Return the arctangent of a point to the X-axis.
ceil()	See “Math.ceil() Method” on page 192.	Return the value rounded up.

<code>cos()</code>	See “Math.cos() Method” on page 192.	Return the cosine of a number.
<code>exp()</code>	See “Math.exp() Method” on page 193.	Return the computed exponential of E.
<code>floor()</code>	See “Math.floor() Method” on page 193.	Return the value rounded down.
<code>log()</code>	See “Math.log() Method” on page 194.	Return the natural logarithm of a number.
<code>max()</code>	See “Math.max() Method” on page 195.	Return the greater of two numbers.
<code>min()</code>	See “Math.min() Method” on page 195.	Return the lesser of two numbers.
<code>pow()</code>	See “Math.pow() Method” on page 196.	Return X^Y .
<code>random()</code>	See “Math.random() Method” on page 197.	Return a pseudo-random number between 0 and 1.
<code>round()</code>	See “Math.round() Method” on page 197.	Return the value of a number rounded to the nearest integer.
<code>sin()</code>	See “Math.sin() Method” on page 197.	Return the sine of a number.
<code>sqrt()</code>	See “Math.sqrt() Method” on page 198.	Return the square root of a number.
<code>tan()</code>	See “Math.tan() Method” on page 199.	Return the tangent of a number.

Math.abs() Method

`Math.abs(x)`

Description

The `abs()` method returns the absolute value of a number.

Parameters

x A number.

Math.acos() Method

`Math.acos(x)`

Description

The `acos()` method returns the arccosine (in radians) of a number. The value is between 0 and π radians. If the argument is outside this range, the method returns NaN.

Parameters

x A number between -1.0 and 1.0.

See Also

“Math.asin() Method” on page 190, “Math.atan() Method” on page 191, “Math.atan2() Method” on page 191, “Math.cos() Method” on page 192, “Math.sin() Method” on page 197, “Math.tan() Method” on page 199

Math.asin() Method

`Math.asin(x)`

Description

The `asin()` method returns the arcsine (in radians) of a number. The value is between $-\pi/2$ and $\pi/2$ radians. If the value of the argument is outside this range, it returns NaN.

Parameters

x A number between -1.0 and 1.0.

See Also

“Math.acos() Method” on page 190, “Math.atan() Method” on page 191, “Math.atan2() Method” on page 191, “Math.cos() Method” on page 192, “Math.sin() Method” on page 197, “Math.tan() Method” on page 199

Math.atan() Method

`Math.atan(x)`

Description

The `atan()` method returns the arctangent (in radians) of a number. The value is between $-\pi/2$ and $\pi/2$ radians.

Parameters

<code>x</code>	A number
----------------	----------

See Also

“Math.acos() Method” on page 190, “Math.asin() Method” on page 190, “Math.atan2() Method” on page 191, “Math.cos() Method” on page 192, “Math.sin() Method” on page 197, “Math.tan() Method” on page 199

Math.atan2() Method

`Math.atan2(y, x)`

Description

The `atan2()` method returns the arctangent of a point to the X-axis. The numeric value is between $-\pi$ and π . Note that the arguments to this function pass the y-coordinate first and the x-coordinate second.

Parameters

<code>x,y</code>	Numbers.
------------------	----------

See Also

“Math.acos() Method” on page 190, “Math.asin() Method” on page 190, “Math.atan() Method” on page 191, “Math.cos() Method” on page 192, “Math.sin() Method” on page 197, “Math.tan() Method” on page 199

Math.ceil() Method

`Math.ceil(x)`

Description

The `ceil()` method returns the value rounded up.

Parameters

`x` A number

See Also

“Math.floor() Method” on page 193

Math.cos() Method

`Math.cos(x)`

Description

The `cos()` method returns the cosine (in radians) of a number. The value is between -1 and 1.

Parameters

`x` A number

See Also

“Math.acos() Method” on page 190, “Math.asin() Method” on page 190, “Math.atan() Method” on page 191, “Math.sin() Method” on page 197, “Math.tan() Method” on page 199

Math.E Constant

`Math.E`

Description

The `E` property represents Euler's constant and the base of natural logarithms (approximately 2.718). This property can only be read.

Math.exp() Method

`Math.exp(x)`

Description

The `exp()` method returns the computed exponential of `E`.

Parameters

`x` A number

See Also

“Math.E Constant” on page 193, “Math.log() Method” on page 194, “Math.pow() Method” on page 196

Math.floor() Method

`Math.floor(x)`

Description

The `floor()` method returns the value rounded down.

Parameters

`x` A number

See Also

“Math.ceil() Method” on page 192

Math.LN2 Constant

`Math.LN2`

Description

The LN2 property is the natural logarithm of 2 (approximately 0.693). This property can only be read.

Math.LN10 Constant

`Math.LN10`

Description

The LN10 property is the natural logarithm of 10 (approximately 2.302). This property can only be read.

Math.log() Method

`Math.log(x)`

Description

The `log()` method returns the natural logarithm of a number. If the value is negative, the return value is always NaN.

Parameters

<i>x</i>	A number
----------	----------

See Also

“Math.exp() Method” on page 193, “Math.pow() Method” on page 196

Math.LOG2E Constant

`Math.LOG2E`

Description

The `LOG2E` property is the base 2 logarithm of `E` (approximately 1.442). This property can only be read.

Math.LOG10E Constant

`Math.LOG10E`

Description

The `LOG10E` property is the base 10 logarithm of `E` (approximately 0.434). This property can only be read.

Math.max() Method

`Math.max(x, y)`

Description

The `max()` method returns the larger of two numbers.

Parameters

`x, y` Numbers.

See Also

“`Math.min()` Method” on page 195

Math.min() Method

`Math.min(x, y)`

Description

The `min()` method returns the smaller of two numbers.

Parameters

x,y Numbers.

See Also

“`Math.max()` Method” on page 195

Math.PI Constant

`Math.PI`

Description

The `PI` property is the ratio of the circumference of a circle to its diameter (approximately 3.14159). This property can only be read.

Math.pow() Method

`Math.pow(base,exponent)`

Description

The `pow()` method returns X^Y .

Parameters

base The base number.

exponent The exponent to which *base* is raised.

See Also

“`Math.exp()` Method” on page 193, “`Math.log()` Method” on page 194

Math.random() Method

`Math.random()`

Description

The `random()` method returns a pseudo-random number between 0 and 1. The random number generator is seeded from the current time.

Parameters

None.

Math.round() Method

`math.round(x)`

Description

The `round()` method returns the value of a number rounded to the nearest integer. If the fractional portion of number is .5 or greater, the argument is rounded to the next higher integer. If the fractional portion of number is less than .5, the argument is rounded to the next lower integer.

Parameters

`x` A number

Math.sin() Method

`Math.sin(x)`

Description

The `sin()` method returns the sine of a number. The value is between -1 and 1.

Parameters

x A number

See Also

“Math.acos() Method” on page 190, “Math.asin() Method” on page 190, “Math.atan() Method” on page 191, “Math.atan2() Method” on page 191, “Math.cos() Method” on page 192, “Math.tan() Method” on page 199

Math.sqrt() Method

`Math.sqrt(x)`

Description

The `sqrt()` method returns the square root of a number.

Parameters

x A number

Math.SQRT1_2 Constant

`Math.SQRT1_2`

Description

The `SQRT1_2` property represents the square root of 1/2—equivalently, 1 over the square root of 2, approximately 0.707. This property can only be read.

Math.SQRT2 Constant

`Math.SQRT2`

Description

The SQRT2 property represents the square root of 2 (approximately 1.414). This property can only be read.

Math.tan() Method

`Math.tan(x)`

Description

The `tan()` method returns the tangent (in radians) of a number.

Parameters

<code>x</code>	A number
----------------	----------

See Also

“`Math.acos()` Method” on page 190, “`Math.asin()` Method” on page 190, “`Math.atan()` Method” on page 191, “`Math.atan2()` Method” on page 191, “`Math.cos()` Method” on page 192, “`Math.sin()` Method” on page 197

Maxscroll() Global Property

`variableName.maxscroll`

Description

The `maxscroll` global property specifies the maximum value allowed for the scroll property. `variableName` specifies a variable associated with a text field. `maxscroll` is used in conjunction with the `scroll` property. This property is read-only.

See Also

“Scroll Global Property” on page 243

Mouse Object

Description

The Mouse object is used to show or hide the cursor. The Mouse object and methods are static—you do not create Mouse objects using a constructor.

Properties

None.

Methods

<code>Mouse.hide</code>	See “Mouse.hide() Method” on page 200	Hide the mouse cursor.
<code>Mouse.show</code>	See “Mouse.show() Method” on page 200	Show the mouse cursor.

Mouse.hide() Method

`Mouse.hide()`

Description

The `hide()` method hides the mouse cursor.

Parameters

None.

See Also

“Mouse.show() Method” on page 200

Mouse.show() Method

`Mouse.show()`

Description

The `show()` method shows the mouse cursor.

Parameters

None

See Also

“`Mouse.hide()` Method” on page 200

MovieClip Object

Description

The `MovieClip` object is the object at the heart of LiveMotion. `_root` itself is an instance of the `MovieClip` object, and most of the `MovieClip` properties and methods are also available as global properties and functions.

Constructor

None. Movies are created using the LiveMotion composition window. Movie instances can be created with `attachMovie()` and `duplicateMovieClip()`.

Properties

<code>_alpha</code>	See “ <code>MovieClip._alpha</code> Property” on page 205	Opacity of the movie clip on a scale of 0 (transparent) to 100 (opaque).
<code>_currentframe</code>	See “ <code>MovieClip._currentframe</code> Property” on page 207	Location of the movie clip playhead.
<code>_droptarget</code>	See “ <code>MovieClip._droptarget</code> Property” on page 207	Absolute path (in slash notation) of a movie clip over which the <i>MovieClip</i> passes during drag operations by the user.
<code>_framesloaded</code>	See “ <code>MovieClip._framesloaded</code> Property” on page 208	Number of movie clip frames that have been loaded.

<code>_height</code>	See “ <code>MovieClip._height</code> Property” on page 213	Height of the movie clip in pixels.
<code>_name</code>	See “ <code>MovieClip._name</code> Property” on page 218	Name of the movie clip.
<code>_parent</code>	See “ <code>MovieClip._parent</code> Property” on page 218	The movie clip containing this movie clip.
<code>_rotation</code>	See “ <code>MovieClip._rotation</code> Property” on page 220	Rotation angle of the movie clip in degrees.
<code>_target</code>	See “ <code>MovieClip._target</code> Property” on page 223	Absolute path of the movie clip.
<code>_totalframes</code>	See “ <code>MovieClip._totalframes</code> Property” on page 223	Number of frames in the movie clip.
<code>_url</code>	See “ <code>MovieClip._url</code> Property” on page 224	URL from which the movie clip was loaded.
<code>_visible</code>	See “ <code>MovieClip._visible</code> Property” on page 225	Boolean indicating whether the movie clip is visible.
<code>_width</code>	See “ <code>MovieClip._width</code> Property” on page 225	Width of the movie clip in pixels. Also a global movie clip property.
<code>_x</code>	See “ <code>MovieClip._x</code> Property” on page 226	Horizontal location of the movie clip in pixels.
<code>_xmouse</code>	See “ <code>MovieClip._xmouse</code> Property” on page 226	Horizontal location of mouse pointer in pixels. Also a global movie clip property.
<code>_xscale</code>	See “ <code>MovieClip._xscale</code> Property” on page 227	Horizontal scaling factor of the movie clip (0% - 100%).
<code>_y</code>	See “ <code>MovieClip._y</code> Property” on page 227	Vertical location of the movie clip in pixels.
<code>_ymouse</code>	See “ <code>MovieClip._ymouse</code> Property” on page 228	Vertical location of mouse pointer in pixels. Also a global movie clip property.
<code>_yscale</code>	See “ <code>MovieClip._yscale</code> Property” on page 228	The vertical scaling factor of the movie clip (0% - 100%).

Methods

<code>attachMovie()</code>	See "MovieClip.attachMovie() Method" on page 205	Attach the named movie clip (passed in as an argument) to the movie clip.
<code>duplicateMovieClip()</code>	See "MovieClip.duplicateMovieClip() Method" on page 207	Duplicate this movie clip. Also a global movie clip function. See "DuplicateMovieClip() Global Function" on page 162
<code>getBounds()</code>	See "MovieClip.getBounds() Method" on page 209	Return bounds of the movie clip. The returned object contains the values in the properties <code>xMin</code> , <code>xMax</code> , <code>yMin</code> and <code>yMax</code> .
<code>getBytesLoaded()</code>	See "MovieClip.getBytesLoaded() Method" on page 210	Return the number of bytes already loaded if the movie clip is external (loaded with <code>MovieClip.loadMovie()</code>). If the movie clip is internal, the number returned is always the same as that returned by <code>MovieClip.getBytesTotal()</code> .
<code>getBytesTotal()</code>	See "MovieClip.getBytesTotal() Method" on page 210	Return the size of the movie clip in bytes. When running under the preview mode in LiveMotion, this number is always 1000.
<code>getURL()</code>	See "MovieClip.getURL() Method" on page 211	Load the URL into the browser. Also a global movie clip function. See "GetURL Global Function" on page 166.
<code>globalToLocal()</code>	See "MovieClip.globalToLocal() Method" on page 212	Convert the given global point to local coordinates.
<code>gotoAndPlay()</code>	See "MovieClip.gotoAndPlay() Method" on page 213	Go to the specified frame or label and play. Also a global movie clip function. See "GotoAndPlay() Global Function" on page 168.
<code>gotoAndStop()</code>	See "MovieClip.gotoAndStop() Method" on page 213	Go to the specified frame or label and stop. Also a global movie clip function. See "GotoAndStop() Global Function" on page 169.

<code>hitTest()</code>	See “ <code>MovieClip.hitTest()</code> Method” on page 214	Return a Boolean indicating whether the movie clip intersects with a given clip (passed in as an argument) or given x/y coordinates.
<code>lmSetCurrentState()</code>	See “ <code>MovieClip.lmSetCurrentState()</code> Method” on page 215	Change the state of the movie clip. The <code>LiveMotion</code> state of the movie clip must already be defined and appear in the state browser.
<code>loadMovie()</code>	See “ <code>MovieClip.loadMovie()</code> Method” on page 216	Load an external SWF file into the player. Also a global movie clip function. See “ <code>LoadMovie()</code> Global Function” on page 184
<code>loadVariables()</code>	See “ <code>MovieClip.loadVariables()</code> Method” on page 216	Load variables fetched from the specified URL. The movie clip’s <code>onData</code> handler is called when the variables have been loaded. Also a global movie clip function. See “ <code>LoadVariables()</code> Global Function” on page 185.
<code>localToGlobal()</code>	See “ <code>MovieClip.localToGlobal()</code> Method” on page 217	Convert the given local point to global coordinates.
<code>nextFrame()</code>	See “ <code>MovieClip.nextFrame()</code> Method” on page 218	Go to the next frame and stop playing. Also a global movie clip function. See “ <code>NextFrame()</code> Global Function” on page 229.
<code>play()</code>	See “ <code>MovieClip.play()</code> Method” on page 219	Start playing. Also a global movie clip function. See “ <code>Play()</code> Global Function” on page 241.
<code>prevFrame()</code>	See “ <code>MovieClip.prevFrame()</code> Method” on page 219	Go to the previous frame and stop playing. Also a global movie clip function. See “ <code>PrevFrame()</code> Global Function” on page 241.
<code>removeMovieClip()</code>	See “ <code>MovieClip.removeMovieClip()</code> Method” on page 220	Delete a duplicated or attached instance. Also a global movie clip function. See “ <code>RemoveMovieClip()</code> Global Function” on page 242.

<code>startDrag()</code>	See “ MovieClip.startDrag() Method ” on page 220	Start dragging a movie clip. Also a global movie clip function. See “ StartDrag() Global Function ” on page 255.
<code>stop()</code>	See “ MovieClip.stop() Method ” on page 221	Stop playing. Also a global movie clip function. See “ Stop() Global Function ” on page 255.
<code>stopDrag()</code>	See “ MovieClip.stopDrag() Method ” on page 222	Stop any drag operation in progress. Also a global movie clip function. See “ StartDrag() Global Function ” on page 255.
<code>swapDepths()</code>	See “ MovieClip.swapDepths() Method ” on page 222	Swap the movie clip’s depth with that of another movie clip.
<code>unloadMovie()</code>	See “ MovieClip.unloadMovie() Method ” on page 224	Unload a movie that was previously loaded with <code>loadmovie()</code> . Also a global movie clip function. See “ UnloadMovie() Global Function ” on page 271.
<code>valueOf()</code>	See “ MovieClip.valueOf() Method ” on page 225	Returns the absolute path to the movie clip using dot (as opposed to slash) notation.

MovieClip._alpha Property

MovieClip._alpha

Description

The `_alpha` property sets the opacity of the movie clip. 0 is transparent; 100 is opaque. This property may be read or written.

MovieClip.attachMovie() Method

MovieClip.attachMovie([exportName](#), [newName](#), [depth](#))

Description

The `attachMovie()` method creates a new instance of *exportName* and attaches it to the movie clip by placing it at the designated depth in *MovieClip*'s programmatic stack. Remove the attached movie clip by using the *MovieClip.removeMovieClip()* method or the `removeMovieClip()` global function. The movie clip may also be removed by placing another movie clip at the same depth in the programmatic stack.

exportName is the sharing name of the movie clip that is to be attached.

A movie clip can be attached to the main movie clip as well using the syntax `_root.attachMovie(library, newName, depth)`.

A movie clip instanced using `attachMovie()` becomes a child of the movie clip through which the method was called, and is in that movie's programmatic stack. For example:

```
clipA.attachMovie(url, "clipB", depth);
```

`clipB` is a child of `clipA` and is in `clipA`'s programmatic stack.

In contrast, a movie clip instanced using `duplicateMovieClip()` becomes a child of the parent of the movie clip through which the method was called, and is in the parent's programmatic stack. For example:

```
clipA.duplicateMovieClip("clipB", depth);
```

`clipB` is a child of `clipA._parent` and is in `clipA._parent`'s programmatic stack.

Note: In preview, the movie clip that is attached is the local version only. If the "Use External Asset" feature is used from the Export palette, this will not be the same movie clip that is actually used when the SWF file is executing in the Flash player.

Parameters

<i>exportName</i>	The movie clip to be attached. This movie clip already exists in the current SWF file. It was assigned its sharing name (<i>exportName</i>) via the Export palette. A remote copy may or may not have been loaded in when the SWF file was loaded into the Flash player, depending on whether the "Use External Asset" feature was used from the Export palette.
<i>newName</i>	Name for the movie clip.
<i>depth</i>	Depth for the movie clip on the programmatic stack.

See Also

“RemoveMovieClip() Global Function” on page 242, “UnloadMovie() Global Function” on page 271, “MovieClip.removeMovieClip() Method” on page 220, “MovieClip.unloadMovie() Method” on page 224, “MovieClip.attachMovie() Method” on page 205

MovieClip._currentframe Property

MovieClip._currentframe

Description

The `_currentframe` property specifies the location (frame number) of the playhead of *MovieClip*. This property can only be read.

MovieClip._droptarget Property

MovieClip._droptarget

Description

The `_droptarget` property is a string value that specifies the absolute path (in slash notation) of a movie clip over which *MovieClip* passes during drag operations by the user. This property can only be read.

MovieClip.duplicateMovieClip() Method

MovieClip.duplicateMovieClip(newName, depth)

Description

The `duplicateMovieClip()` method duplicates *MovieClip*. Duplicated movies always start playing at frame 1. The duplicated movie clip inherits shape transformations but not the timeline variables. The duplicated movie clip is placed in *MovieClip*'s parent's programmatic stack. A programmatic stack holds child movie clips; when you duplicate a movie clip it will have the same parent as the original, and thus “live” in the parent's programmatic stack. The `removeMovieClip()` method is used to delete duplicated movie clips.

`MovieClip.removeMovieClip()` can be used by duplicated movie clips to delete themselves, or the `removeMovieClip()` global function can be used to delete duplicated movie clips. Duplicated movie clips can also be removed by placing another movie clip at the same depth in the programmatic stack.

A movie clip instanced using `duplicateMovieClip()` becomes a child of the parent of the movie clip through which the method was called, and is in the parent's programmatic stack. For example:

```
clipA.duplicateMovieClip("clipB", depth);
```

`clipB` is a child of `clipA._parent` and is in `clipA._parent`'s programmatic stack.

In contrast, a movie clip instanced using `attachMovie()` becomes a child of the movie clip through which the method was called, and is in that movie's programmatic stack. For example:

```
clipA.attachMovie(url, "clipB", depth);
```

`clipB` is a child of `clipA` and is in `clipA`'s programmatic stack.

Parameters

<i>newName</i>	A string indicating the new name for the duplicate movie clip.
<i>depth</i>	An integer indicating the depth at which the duplicated movie clip is placed in <i>MovieClip</i> 's parent's programmatic stack.

Example

```
_root.baseball.duplicateMovieClip ("newBaseball", 1); //creates new baseball  
_root.newBaseball._x += 25; //moves new baseball along x axis  
_root.newBaseball._y += 25; //moves new baseball along y axis
```

See Also

“RemoveMovieClip() Global Function” on page 242, “MovieClip.removeMovieClip() Method” on page 220, “DuplicateMovieClip() Global Function” on page 162, “MovieClip.attachMovie() Method” on page 205

MovieClip._framesloaded Property

MovieClip.framesloaded

Description

The `_framesloaded` property holds the number of frames that have already been downloaded. This property is read-only.

This property is often used in conjunction with the `_totalframes` property to create a preloader for the `_root` movie clip. For example, you could place the following code in a script on a keyframe somewhere between the `beginLoop` and `Start` labels. The `_root` movie clip loops between the `beginLoop` label and the keyframe where the script is, then jumps to the `Start` label when the entire `_root` movie clip has downloaded.

```
if (_root._framesloaded == _root._totalframes)
{
    _root.gotoAndPlay("Start");
}
else
{
    _root.gotoAndPlay("beginLoop");
}
```

See Also

“`MovieClip._totalframes` Property” on page 223

MovieClip.getBounds() Method

`MovieClip.getBounds()`
`MovieClip.getBounds(targetCoordinateSpace)`

Description

The `getBounds()` method returns the bounds of the movie clip as an object. The values returned represent the x and y coordinates of *targetCoordinateSpace*.

Parameters

<i>targetCoordinateSpace</i>	(Optional) String showing the path to the movie clip. Defaults to <i>MovieClip</i> if not specified.
------------------------------	--

Returns

An object with four properties: *object.xMin*, *object.xMax*, *object.yMin*, *object.yMax*.

Example

```
var coordinates = _root.baseball.getBounds();  
trace(coordinates.xMin); //prints value  
trace(coordinates.xMax); //prints value  
trace(coordinates.yMin); //prints value  
trace(coordinates.yMax); //prints value
```

See Also

“MovieClip.globalToLocal() Method” on page 212, “MovieClip.localToGlobal() Method” on page 217

MovieClip.getBytesLoaded() Method

MovieClip.getBytesLoaded()

Description

The `getBytesLoaded()` method returns the number of bytes already loaded if this movie clip is external. If internal, the number returned is always the same as that returned by *MovieClip.getBytesTotal()*.

Parameters

None.

Returns

The number of bytes already loaded for *MovieClip*.

See Also

“MovieClip.getBytesTotal() Method” on page 210

MovieClip.getBytesTotal() Method

MovieClip.getBytesTotal()

Description

The size of *MovieClip* in bytes. When running under the preview tool in LiveMotion, this number is always 1000.

Parameters

None

Returns

The size of *MovieClip* in bytes.

See Also

“*MovieClip.getBytesLoaded()* Method” on page 210

MovieClip.getURL() Method

MovieClip.getURL(url, target)

MovieClip.getURL(url, target, howToSendVariables)

Description

The `getURL()` method loads a URL into the web browser. It operates the same as the global form, except when variables are sent they are sent from the *MovieClip* timeline rather than the main timeline.

Note: This method is not supported in preview mode.

Parameters

url

A string specifying the URL to which to hyperlink. This may be a relative or an absolute pathname, or the name of a document or script.

<i>target</i>	(Optional) A string specifying the target frame in the browser—e.g., <code>_self</code> (the default), <code>_parent</code> , <code>_top</code> , <code>_blank</code> . If omitted, <code>_self</code> is used. Custom names can also be used.
<i>howToSendVariables</i>	(Optional) Omit this parameter if you don't want to send the variables. This parameter is a string literal. Specify <code>GET</code> to send variables via get (i.e., tacked onto the end of the URL) or <code>POST</code> to send them with post (i.e., put into the body of the request). Both methods send them in application/x-www-form-urlencoded MIME format. All user-defined variables are sent, except for user-defined standard handlers.

See Also

“GetURL Global Function” on page 166

MovieClip.globalToLocal() Method

MovieClip.globalToLocal(point)

Description

The `globalToLocal()` method converts the given global point to local coordinates.

Parameters

point An object of type `Object` with two properties: `x` and `y`. `x` and `y` are set to the global coordinates before the object *point* is passed to `globalToLocal()`.

Example

```
wheresTheMouse = new Object();
wheresTheMouse.x = _root._xmouse;
wheresTheMouse.y = _root._ymouse;
this.globalToLocal(wheresTheMouse);
//wheresTheMouse.x and wheresTheMouse.y now contain local coordinates
```

See Also

“MovieClip.getBounds() Method” on page 209, “MovieClip.localToGlobal() Method” on page 217

MovieClip.gotoAndPlay() Method

MovieClip.gotoAndPlay(frameLabel)

Description

The `gotoAndPlay()` method goes to the specified frame or label and plays.

Parameters

frameLabel String representing a label.

See Also

“`MovieClip.gotoAndStop()` Method” on page 213

MovieClip.gotoAndStop() Method

MovieClip.gotoAndStop(frameLabel)

Description

The `gotoAndStop()` method goes to the specified frame or label and stops.

Parameters

framenumber Integer or expression that evaluates to an integer. Indicates a frame number.

frameLabel String representing a label.

See Also

“`MovieClip.gotoAndPlay()` Method” on page 213

MovieClip._height Property

MovieClip._height

Description

The `_height` property represents the height of the movie clip in pixels. The `_height` property is based on the content within *MovieClip*. If *MovieClip* has no content, then `_height` is 0. `_height` is also determined by placement of the objects within *MovieClip*: the farthest object toward the top or bottom determines the value of `_height`. This property is read-only.

Note: Only `_root.height` and `_root.width` return dimensions of the `_root` movie clip. Technically, these are local properties that can also be used on the `_root` movie clip.

See Also

“`MovieClip._width` Property” on page 225

MovieClip.hitTest() Method

MovieClip.hitTest(x, y, shapeFlag)

MovieClip.hitTest(target)

Description

The `hitTest()` method returns a Boolean indicating whether *MovieClip* intersects with a specific point on the stage, or overlaps with another movie clip. When specifying the hit test, you indicate whether the test involves matching a specific x/y point on the stage against just the border of *MovieClip* or all of it (first form), or finding any overlap with the other clip (*target* in the second form).

Parameters

<i>x</i>	Horizontal component of the hit test. Defined in global coordinate space.
<i>y</i>	Vertical component of the hit test. Defined in global coordinate space.
<i>shapeFlag</i>	Boolean indicating whether the test should test just the bounding box (<code>false</code>) or all pixels (<code>true</code>) of the movie clip.
<i>target</i>	String indicating path to the movie clip against which the hit test is made.

Returns

true if a hit occurred; false otherwise.

Example

```
if (this.hitTest(_root._xmouse, _root._ymouse, true))
{
    trace("The mouse has passed over the movie clip");
};
```

See Also

“MovieClip.getBounds() Method” on page 209

MovieClip.lmSetCurrentState() Method

MovieClip.lmSetCurrentState(label)

Description

The `lmSetCurrentState()` sets the state of *MovieClip*. *MovieClip*'s states are defined from the composition window and are shown in the state browser.

Parameters

<i>label</i>	String literal representing <i>MovieClip</i> state that was already defined for <i>MovieClip</i> and that appears in the state browser. This can be a standard state like “over”, or a custom state. Must appear in quotes.
--------------	---

Example

```
if (_xmouse < 175 && _ymouse > 100)
{
    _root.Spiral.lmSetCurrentState("Purple");
}
if (_xmouse > 175 && _ymouse > 100)
{
    _root.Spiral.lmSetCurrentState("Green");
}
```

MovieClip.loadMovie() Method

```
MovieClip.loadMovie(url)  
MovieClip.loadMovie(url, howToSendVariables)
```

Description

The `loadMovie()` method brings an external SWF file into the player and optionally loads variables. *MovieClip* and any associated programmatically generated movies associated with it (previously created with `attachMovie()` or `duplicateMovie()`) are replaced with the new SWF file. Use `unloadMovie()` to remove the movie. The `unloadMovie()` global function can also be used to remove the movie clip.

Note: This method is not supported in preview mode.

Parameters

<i>url</i>	String literal representing URL from which to get the SWF file to load. This can be an absolute or a relative URL. The URL must be in the same subdomain as the URL where the movie clip currently resides.
<i>howToSendVariables</i>	(Optional) GET or POST. String literal that specifies the method for sending variables to the server. The GET method appends them to the URL; the POST method sends them in a separate HTTP packet.

Example

```
_root.baseball.loadMovie("http://devtech.corp.adobe.com/docs/livemotion/bil  
lys.swf");
```

See Also

“LoadMovie() Global Function” on page 184, “UnloadMovie() Global Function” on page 271, “MovieClip.unloadMovie() Method” on page 224

MovieClip.loadVariables() Method

```
MovieClip.loadVariables(URL, howToSendVariables)
```

Description

The `loadVariables()` method loads variables fetched from the specified URL. The movie clip's `onData` event handler is called when all of the variables have been loaded.

The data fetched from the URL must be in the `application/x-www-form-urlencoded` MIME format.

Parameters

<i>URL</i>	The URL from which to get the variables. The host for the URL must be in the same subdomain as the movie clip.
<i>howToSendVariables</i>	(Optional) Omit this parameter if you don't want to send the variables. GET or POST. String literal that specifies the method for sending variables to the movie clip that is to be loaded. The GET method appends them to the URL; the POST method sends them in a separate HTTP packet. Using either method they're sent in <code>application/x-www-form-urlencoded</code> MIME format. All user defined variables are sent, except for user-defined standard handlers.

See Also

“`LoadVariables()` Global Function” on page 185, “`LoadVariablesNum()` Global Function” on page 187, “`GetURL` Global Function” on page 166, “`MovieClip.getURL()` Method” on page 211,

MovieClip.localToGlobal() Method

MovieClip.localToGlobal(point)

Description

The `localToGlobal()` method converts the given local point to global coordinates.

Parameters

<i>point</i>	An object of type <code>Object</code> with two properties: <code>x</code> and <code>y</code> . <code>x</code> and <code>y</code> are set to the local coordinates before the object <i>point</i> is passed to <code>localToGlobal()</code> .
--------------	--

See Also

“MovieClip.getBounds() Method” on page 209, “MovieClip.localToGlobal() Method” on page 217

MovieClip._name Property

MovieClip._name

Description

The `_name` property of the movie clip represents the name of the movie clip as a string (as opposed to a reference). This is a relative reference (no pathname is returned). This property can be read or written.

MovieClip.nextFrame() Method

MovieClip.nextFrame()

Description

The `nextFrame()` method moves the playhead to the next frame and stops the playhead.

Parameters

None.

See Also

“NextFrame() Global Function” on page 229, “MovieClip.prevFrame() Method” on page 219, “MovieClip.stop() Method” on page 221, “MovieClip.play() Method” on page 219

MovieClip._parent Property

MovieClip._parent

Description

The `_parent` property is a reference (does not contain a string) to the movie clip or movie clip group containing this *MovieClip*. This allows syntax such as: `_parent._parent.stop()` ;

MovieClip.play() Method

MovieClip.play()

Description

The `play()` method starts *MovieClip* playing.

Parameters

None.

See Also

“Play() Global Function” on page 241, “*MovieClip*.prevFrame() Method” on page 219, “*MovieClip*.stop() Method” on page 221

MovieClip.prevFrame() Method

MovieClip.prevFrame()

Description

The `prevFrame()` method takes the playhead to the previous frame and stops playing.

Parameters

None.

See Also

“PrevFrame() Global Function” on page 241, “*MovieClip*.nextFrame() Method” on page 218, “*MovieClip*.stop() Method” on page 221, “*MovieClip*.play() Method” on page 219

MovieClip.removeMovieClip() Method

MovieClip.removeMovieClip()

Description

The `removeMovieClip()` method deletes a duplicated or attached movie clip. Unlike the `removeMovieClip()` global function, duplicated or attached movie clips that call this method can only delete themselves.

Parameters

None

See Also

“RemoveMovieClip() Global Function” on page 242, “DuplicateMovieClip() Global Function” on page 162, “MovieClip.duplicateMovieClip() Method” on page 207, “MovieClip.attachMovie() Method” on page 205

MovieClip._rotation Property

MovieClip._rotation

Description

The `_rotation` property specifies the rotation of the movie clip in degrees. This property can be read or written.

MovieClip.startDrag() Method

MovieClip.startDrag()

MovieClip.startDrag(lockCenter)

MovieClip.startDrag(lockCenter, left, top, right, bottom)

Description

The `startDrag()` method causes *MovieClip* to physically follow the mouse pointer. Use `stopDrag()` to halt dragging.

Parameters

<i>lockCenter</i>	(Optional) Boolean indicating whether the draggable <i>MovieClip</i> should be centered under the mouse pointer (<i>true</i>) or dragged relative to the mouse pointer's location when clicked (<i>false</i>). Default is <i>false</i> .
<i>left</i>	(Optional) x-coordinate boundary to the left of which <i>MovieClip</i> cannot be dragged.
<i>top</i>	(Optional) y-coordinate boundary above which <i>MovieClip</i> cannot be dragged.
<i>right</i>	(Optional) x-coordinate boundary to the right of which <i>MovieClip</i> cannot be dragged.
<i>bottom</i>	(Optional) y-coordinate boundary below which <i>MovieClip</i> cannot be dragged.

Example

```
//onButtonPress event  
this.startDrag();  
//onButtonRelease event  
this.stopDrag();
```

See Also

“*MovieClip.stopDrag()* Method” on page 222, “*startDrag()* Global Function” on page 255

MovieClip.stop() Method

MovieClip.stop()

Description

The stop() method stops the playing of *MovieClip*.

Parameters

None.

MovieClip.stopDrag() Method

MovieClip.stopDrag()

Description

The `stopDrag()` method ends any drag operation currently in progress.

Parameters

None.

Example

```
//onButtonPress event  
this.startDrag();  
//onButtonRelease event  
this.stopDrag();
```

See Also

“`MovieClip.startDrag()` Method” on page 220, “`StopDrag()` Global Function” on page 256

MovieClip.swapDepths() Method

MovieClip.swapDepths(target)
MovieClip.swapDepths(depth)

Description

The `swapDepths()` method changes the position of *MovieClip* in *MovieClip*’s parent’s visual stacking order. Movie clips at the top of the stack (higher level numbers) cover those lower in the stack. You can swap the depths of attached or duplicated movie clips with manually created clips, but ensure that you test extensively since this has been a problem area with the Flash player in the past.

Parameters

<i>target</i>	String indicating the path to the movie clip to be swapped with <i>MovieClip</i> . The movie clip and <i>MovieClip</i> must have the same parent.
<i>depth</i>	Integer specifying the level in <i>MovieClip</i> 's parent's visual stack with which to swap. If another movie clip resides at this level, then full swapping occurs. Otherwise, <i>MovieClip</i> is simply moved to that level. This integer may be positive, negative, or 0. The higher the number, the more visible the layer is.

Example

```
swapDepths(_root.ellipse);//swaps depths with the movie clip ellipse  
swapDepths(3);//swaps depths at level 3
```

MovieClip._target Property

MovieClip._target

Description

The `_target` property represents the target path of *MovieClip* in absolute terms using slash notation. To get the path in dot notation, use the `targetPath()` global function.

Example

```
trace(this._target);//prints .root/baseball/homer (this = homer)
```

See Also

“`TargetPath()` Global Function” on page 269

MovieClip._totalframes Property

MovieClip._totalframes

Description

The `_totalframes` property specifies the total number of frames in *MovieClip*. It is often used in conjunction with the `_framesloaded` property to determine the percentage of total frames that have already downloaded; when an acceptable number are ready, the movie clip is started. This property is read-only.

See Also

“`MovieClip._framesloaded` Property” on page 208

MovieClip.unloadMovie() Method

MovieClip.unloadMovie()

Description

The `unloadMovie()` method unloads a movie clip that was previously loaded with `loadmovie()`.

Parameters

None.

See Also

“`UnloadMovie()` Global Function” on page 271, “`MovieClip.loadMovie()` Method” on page 216

MovieClip._url Property

MovieClip._url

Description

The `_url` property specifies the URL from which *MovieClip* was loaded.

See Also

“`LoadMovie()` Global Function” on page 184, “`LoadMovieNum()` Global Function” on page 185

MovieClip.valueOf() Method

MovieClip.valueOf()

Description

The `valueOf()` method returns the path to the instance in absolute terms using dot notation.

Parameters

None.

MovieClip._visible Property

MovieClip._visible

Description

Boolean indicating whether *MovieClip* is visible. Visibility: true if visible; false if hidden. This property can be read or written.

See Also

“[MovieClip.swapDepths\(\) Method](#)” on page 222

MovieClip._width Property

MovieClip._width

Description

The `_width` property represents the width of the movie clip in pixels. The `_width` property is based on the content within *MovieClip*. If *MovieClip* has no content, then `_width` is 0. `_width` is also determined by placement of the objects within *MovieClip*: the farthest object to the left or right determines the value of `_width`. This property is read-only.

Note: Only `_root._width` and `_root._height` return dimensions of the `_root` movie clip. Technically, these are local properties that can also be used on the `_root` movie clip.

See Also

“MovieClip._height Property” on page 213

MovieClip._x Property

MovieClip._x

Description

The `_x` property specifies the horizontal position of *MovieClip* in pixels. If *MovieClip* is on the main timeline, then the coordinate system is based on 0,0 x/y coordinates in the upper left corner of the stage. If *MovieClip* is contained within another movie clip, *MovieClip*'s coordinates are relative to the position of the enclosing movie clip's anchor point—that point which represents the 0,0 point for the enclosing movie clip's x/y coordinate system. This property can be read or written.

See Also

“MovieClip._x Property” on page 226

MovieClip._xmouse Property

MovieClip._xmouse

Description

The `_xmouse` property specifies the horizontal location of the mouse pointer in pixels. If *MovieClip* is on the main timeline, then the coordinate system is based on 0,0 x/y coordinates in the upper left corner of the stage. If *MovieClip* is contained within another movie clip, the `_xmouse` coordinate is relative to the position of the enclosing movie clip's anchor point—that point which represents the 0,0 point for the enclosing movie clip's x/y coordinate system. This property can only be read.

Note: The `_xmouse` and `_ymouse` coordinates are relative to the anchor point of the parent movie clip (unless the point of reference is the `_root` movie clip, in which case the upper left-hand corner is the absolute point of origin). Only `_root._xmouse` and `_root._ymouse` return absolute positions. Technically, these are local properties that can also be used on the `_root` movie clip.

See Also

“MovieClip._ymouse Property” on page 228

MovieClip._xscale Property

MovieClip._xscale

Description

The *_xscale* property of *MovieClip* represents the horizontal scaling percentage (0% - 100%) of the movie clip relative to its original size. This property can be read or written.

See Also

“MovieClip._yscale Property” on page 228

MovieClip._y Property

MovieClip._y

Description

The *_y* property specifies the vertical position of *MovieClip* in pixels. If *MovieClip* is on the main timeline, then the coordinate system is based on 0,0 x/y coordinates in the upper left corner of the stage. If *MovieClip* is contained within another movie clip, *MovieClip*'s coordinates are relative to the position of the enclosing movie clip's anchor point—that point which represents the 0,0 point for the enclosing movie clip's x/y coordinate system. This property can be read or written.

Note: In the Flash player, the y-axis is inverted—that is, positive values increase in the “downward” direction rather than upward.

See Also

“MovieClip._x Property” on page 226

MovieClip._ymouse Property

MovieClip._ymouse

Description

The `_ymouse` property specifies the vertical location of mouse pointer in pixels. If *MovieClip* is on the main timeline, then the coordinate system is based on 0,0 x/y coordinates in the upper left corner of the stage. If *MovieClip* is contained within another movie clip, the `_ymouse` coordinate is relative to the position of the enclosing movie clip's anchor point—that point which represents the 0,0 point for the enclosing movie clip's x/y coordinate system. This property can only be read.

Note: The `_ymouse` and `_xmouse` coordinates are relative to the anchor point of the parent movie clip (unless the point of reference is the `_root` movie clip, in which case the upper left-hand corner is the absolute point of origin). Only `_root._ymouse` and `_root._xmouse` return absolute positions. Technically, these are local properties that can also be used on the `_root` movie clip.

Note: In Flash player, the y-axis is inverted—that is, positive values increase in the “downward” direction rather than upward.

See Also

“*MovieClip._xmouse* Property” on page 226

MovieClip._yscale Property

MovieClip._yscale

Description

The `_yscale` property of *MovieClip* represents the vertical scaling percentage (0% - 100%) of the movie clip relative to its original size. This property can be read or written.

See Also

“*MovieClip._xscale* Property” on page 227

NaN Global Property

NaN

Description

The NaN global property is a predefined variable with the value NaN (Not-a-Number), as specified by the IEEE-754 standard.

Example

```
trace(NaN);//prints NaN  
var redFish = NaN;  
trace(redFish);//prints NaN
```

See Also

“IsNaN() Global Function” on page 171, “Number.NaN Property” on page 233

Newline Constant

newline

Description

The `newline` constant is used wherever a `\n` could be used in text to force a line break. It is equivalent to the ASCII value of 10.

NextFrame() Global Function

`nextFrame()`

Description

The `nextFrame()` global function moves the playhead to the next frame and stops it.

Parameters

None

See Also

“MovieClip.nextFrame() Method” on page 218, “PrevFrame() Global Function” on page 241

Number() Global Function

`Number(expression)`

Description

The `Number()` global function converts *expression* into a number.

Parameters

expression String, Boolean, or other expression to convert into a number.

Returns

A number representing the expression.

Example

```
trace(Number(2 * 2)); //prints 4
```

See Also

“parseFloat() Global Function” on page 239, “parseInt() Global Function” on page 240

Number Object**Description**

The `Number` object helps you work with numeric values. It is an object wrapper for primitive numeric values.

The primary uses for the `Number` object are to access constant properties that represent the largest and smallest representable numbers, positive and negative infinity, and the Not-a-Number (NaN) value.

The properties of `Number` are properties of the object itself, not of individual `Number` objects. You need to create an individual object instance of type `Number` only when you wish to use its methods.

Constructor

`new Number(value)`

Parameters

value The numeric value of the object being created.

Properties

MAX_VALUE	See “Number.MAX_VALUE Prop- erty” on page 232	A constant representing the largest rep- resentable number
MIN_VALUE	See “Number.MIN_VALUE Prop- erty” on page 232	A constant representing the smallest representable number.
NaN	See “Number.NaN Property” on page 233	A constant representing the special "Not a Number" value.
NEGATIVE_INFINITY	See “Num- ber.NEGATIVE_INFINITY Prop- erty” on page 233	A constant representing negative infin- ity.
POSITIVE_INFINITY	See “Num- ber.POSITIVE_INFINITYProperty” on page 234	A constant representing positive infinity.

Methods

This table lists each method alphabetically and provides a brief description.

<code>Number.toString</code>	See “Number.toString() Method” on page 234	Return a string representing the object
<code>Number.valueOf</code>	See “Number.valueOf() Method” on page 235	Return the primitive value of the object.

Number.MAX_VALUE Property

Number.MAX_VALUE

Description

The `MAX_VALUE` property represents the maximum representable numeric value. It has value of approximately 1.79E+308. Values larger than `MAX_VALUE` are represented as infinity (see “`Number.POSITIVE_INFINITY` Property” on page 234 and “`Number.NEGATIVE_INFINITY` Property” on page 233). This property can only be read.

Example

```
if (1000 * 100001 <= Number.MAX_VALUE)
    trace("No overflow");//prints "No overflow"
else
    trace("Overflow");
```

See Also

“Number.MIN_VALUE Property” on page 232, “Infinity Global Property” on page 170

Number.MIN_VALUE Property

Number.MIN_VALUE

Description

The `MIN_VALUE` property represents the smallest positive representable numeric value. It is the number closest to 0—not the most negative number that can be represented. `MIN_VALUE` has a value of approximately 2.22E-308. Values smaller than `MIN_VALUE` (“underflow values”) are converted to 0.

Example

```
if (1/100000000000000000000000000000000<= Number.MAX_VALUE)
    trace("No underflow");//prints "No underflow"
else
    trace("Underflow");
```

See Also

“Number.MAX_VALUE Property” on page 232

Number.NaN Property

`Number.NaN`

Description

The `NaN` property is a special value representing Not-A-Number. This value complies with the IEEE-754 value for Not-A-Number. This property can only be read.

Example

```
var twoFish = 1;
if (twoFish < 2 || twoFish > 2) {
    twoFish = Number.NaN;
}
trace(twoFish); //prints "NaN"
```

Number.NEGATIVE_INFINITY Property

`Number.NEGATIVE_INFINITY`

Description

The `NEGATIVE_INFINITY` property is a special numeric value representing negative infinity. Mathematically, this value behaves like infinity—for example, anything multiplied by infinity is infinity, and anything divided by infinity is 0.

Example

```
var IQ = -Number.MAX_VALUE*10;
if (IQ == Number.NEGATIVE_INFINITY)
    trace("Really low"); //prints "Really low"
else
    trace("Not so low");
```

See Also

“Number.POSITIVE_INFINITY Property” on page 234

Number.POSITIVE_INFINITY Property

`Number.POSITIVE_INFINITY`

Description

The `POSITIVE_INFINITY` property is a special numeric value representing infinity. This value behaves mathematically like infinity—for example, anything multiplied by infinity is infinity, and anything divided by infinity is 0.

Example

```
var IQ = Number.MAX_VALUE*10;
if (IQ == Number.POSITIVE_INFINITY)
    trace("Really high");//prints "Really high"
else
    trace("Not so high");
```

See Also

“Number.NEGATIVE_INFINITY Property” on page 233

Number.toString() Method

`Number.toString()`
`Number.toString(radix)`

Description

The `toString()` method returns a string representing the specified object. Since every JavaScript object has the `Object` class as its base class, every object has a `toString()` method that is automatically called when it is to be represented as a text value or when an object is referred to it in a string concatenation.

Parameters

radix

(Optional) An integer between 2 and 16 specifying the base to use for representing numeric values.

Returns

A string representing the specified object.

Example

```
var tenFish = new Number(10);  
trace("Billy and Monica caught " + tenFish.toString() + " fish.");  
//prints "Billy and Monica caught 10 fish."
```

See Also

“Object.toString() Method” on page 237

Number.valueOf() Method

Number.valueOf()

Description

The `valueOf()` method returns the primitive value of the specified object. Since every JavaScript object has the `Object` object as its base class, every object has a `valueOf()` method that is automatically called when its primitive value is to be returned.

Parameters

None.

See Also

“Object.valueOf() Method” on page 239

Object Class

Description

Object is the primitive JavaScript object type. All JavaScript objects are derived from Object. That is, all JavaScript objects have the methods and properties defined for Object available to them. In C++ terminology, Object is the base class that is inherited by all JavaScript objects.

In addition to using a constructor to create a new Object object, you can also use the bracket syntax (e.g., `newObject = { value1: 1, value2: 2};`).

Constructor

`new Object()`
`new Object(value)`

Parameters

value (Optional) A number, Boolean, or string.

Properties

<code>constructor</code>	See “Object.constructor Property” on page 236	Specifies the function that creates an object's prototype.
<code>__proto__</code>	See “Object.__proto__ Property” on page 237	A reference to <code>Object.prototype</code> . The prototype object is used to pass properties and methods to inherited objects.

Methods

<code>toString()</code>	See “Object.toString() Method” on page 237	Returns a string representing the specified object.
<code>valueOf()</code>	See “Object.valueOf() Method” on page 239	Returns the primitive value of the specified object.

Object.constructor Property

`Object.constructor`

Description

The constructor property identifies the function that creates the `Object` object's prototype. The value of this property is a reference to the function itself, not a string containing the function's name. This property can be read or written.

Example

```
beret = new Object();  
trace (beret.constructor == Object); //prints "true"  
  
beret = {};  
trace (beret.constructor == Object); //prints "true"
```

Object.__proto__ Property

`Object.__proto__`

Description

The `__proto__` (double underscores) property is a reference to the `Object.prototype` object. The prototype object is used to pass properties and methods to objects that inherit the `Object` class. Note that the `__proto__` property and prototype object are common to all scripting objects. Since all LiveMotion objects are derived from the `Object` class, you can use the `__proto__` property to add methods and properties to all static and dynamic LiveMotion objects. This property can be read or written.

Example

```
oval = new Object();  
trace(oval.__proto__ == Object.prototype); //prints "true"  
  
Object.prototype.newProp = "office";  
oval2 = new Date();  
trace(oval2.newProp); //Displays "office"
```

Object.toString() Method

`Object.toString()`

Description

Returns a string representing the specified object. Many objects override this method in favor of their own implementation (for example, *Date.toString()*).

If an object has no string value and no user-defined *toString()* method, *toString()* returns [object *type*], where *type* is the object type or the name of the constructor function that created the object.

Parameters

None.

Example

```
function Cat(name,breed,color,sex) {  
    this.name=name  
    this.breed=breed  
    this.color=color  
    this.sex=sex  
}  
theCat = new Cat("Socks","Calico","chocolate","girl");
```

The following code creates *catToString()*, the function that will be used in place of the default *toString()* method. This function generates a string containing each property, of the form “property = value”.

```
function catToString() {  
    var ret = "Cat " + this.name + " is [";  
    for (var prop in this)  
        ret += " " + prop + " is " + this[prop] + " ";  
    return ret + "]"  
}
```

The following code assigns the user-defined function to the object's *toString()* method:

```
Cat.__proto__.toString = catToString;
```

With the preceding code in place, any time *theCat* is used in a string context, JavaScript automatically calls the *catToString* function, which returns the following string:

```
Cat Socks is [ name is Socks; breed is Calico; color is chocolate; sex is  
girl; toString is function catToString() { var ret = "Object " + this.name +  
" is ["; for (var prop in this) { ret += " " + prop + " is " + this[prop] +  
";"; } return ret + "]" ; } ]
```

See Also

“Array.toString() Method” on page 125, “Date.toString() Method” on page 160,
“Number.toString() Method” on page 234, “Object.valueOf() Method” on page 239

Object.valueOf() Method

`Object.valueOf()`

Description

The `valueOf()` method returns the primitive value of the specified object. If an object has no primitive value, `valueOf()` returns the object itself. Note that you rarely need to invoke the `valueOf()` method yourself. JavaScript automatically invokes it when encountering an object where a primitive value is expected.

The following shows the object types for which the `valueOf()` method is most useful. Most other objects have no primitive value.

- Number object type—`valueOf()` returns primitive numeric value associated with the object.
- Boolean object type—`valueOf()` returns primitive Boolean value associated with the object.
- String object type—`valueOf()` returns string associated with the object.

You can create a function to be called in place of the default `valueOf` method. Your function must take no arguments.

Parameters

None.

See Also

“Boolean.valueOf() Method” on page 129, “MovieClip.valueOf() Method” on page 225,
“Number.valueOf() Method” on page 235, “Object.toString() Method” on page 237

parseFloat() Global Function

`parseFloat(string)`

Description

The `parseFloat()` global function converts *string* to a floating-point number. If the function encounters a character that it cannot convert to a number, it returns NaN. The function supports exponential notation (see examples).

Parameters

<i>string</i>	The string to convert to a floating-point number.
---------------	---

Returns

A floating-point decimal number.

Example

```
trace(parseFloat("2.12")); //prints 2.12
trace(parseFloat("a23")); //prints NaN
trace(parseFloat("25e10")); //prints 250000000000
```

See Also

“Number() Global Function” on page 230, “parseInt() Global Function” on page 240

parseInt() Global Function

```
parseInt(string, base)
```

Description

The `parseInt()` global function converts a string to an integer.

Parameters

<i>string</i>	A string, number, or expression. Leading 0x represents hexadecimal. Leading 0 represents octal.
---------------	---

base The base of the string to parse (from base 2 to base 36). If not supplied, *base* is determined by the format of *string*.

Returns

An integer decimal number.

Example

```
trace(parseInt("10")); //prints 10
trace(parseInt("10", 2)); //prints 2 (decimal equivalent of binary 10)
trace(parseInt(10 * 10)); //prints 100
trace(parseInt("0xFF")); //prints 255 (decimal equivalent of hex FF)
trace(parseInt("0377")); //prints 255 (decimal equivalent of octal 377)
```

See Also

“Number() Global Function” on page 230, “parseFloat() Global Function” on page 239

Play() Global Function

play()

Description

The play() global function moves the playhead forward on the timeline.

Parameters

None.

See Also

“GotoAndPlay() Global Function” on page 168, “MovieClip.play() Method” on page 219, “Stop() Global Function” on page 255

PrevFrame() Global Function

prevFrame()

Description

The prevFrame() global function moves the playhead to the previous frame and stops it.

Parameters

None

See Also

“MovieClip.prevFrame() Method” on page 219, “NextFrame() Global Function” on page 229

Quality Global Property

`_quality`

Description

The `_quality` global property sets the level of rendering quality. It takes one of the following strings (must be used with quotes):

- "LOW"—Graphics aren't anti-aliased; bitmaps aren't smoothed.
- "MEDIUM"—Graphics are anti-aliased using a 2x2 grid; bitmaps aren't smoothed.
- "HIGH"—Graphics are anti-aliased using a 4x4 grid; bitmaps are smoothed if the movie clip is static. (Default)
- "BEST"—Graphics are anti-aliased using a 4x4 grid; bitmaps are always smoothed.

See Also

“Highquality Global Property” on page 169

RemoveMovieClip() Global Function

`removeMovieClip(target)`

Description

The `removeMovieClip()` global function deletes a movie clip created with the `duplicateMovieClip()`, `MovieClip.duplicateMovieClip()`, or `MovieClip.attachMovie()`.

Parameters

target The movie clip to be deleted.

See Also

“DuplicateMovieClip() Global Function” on page 162, “MovieClip.duplicateMovieClip() Method” on page 207, “MovieClip.attachMovie() Method” on page 205

Root Global Property

_root

Description

_root is a special case of the MovieClip object. *_root* is a reference to the main movie clip, and as such it can be used in absolute paths to any object. It's equivalent to saying *_level14* if the script is also at *_level14*. It is most often used to invoke methods and reference properties that are members of the *_root* movie clip. For example:

```
_root.attachMovie(exportName, newName, depth) //attaches a movie clip to root  
_root._x = -150 //causes a horizontal offset of the entire SWF file
```

See Also

“LevelN Global Property” on page 182, “MovieClip._parent Property” on page 218

Scroll Global Property

variableName.scroll

Description

The *scroll* global property allows you to control the display of information in a text field by moving the text field to a specific position. It is set to the line number of the topmost visible line in a text field. *variableName* is the name of the variable associated with the text field. *scroll* is used in conjunction with the *maxscroll* property. This property can be read or written.

See Also

“Maxscroll() Global Property” on page 199

Selection Object

Description

The `Selection` object contains information about the text field that currently has focus. A text field gets focus when the user clicks on the text field with the mouse. Since only one text field can have focus at a time, the `Selection` object is static. No constructor is required. In `LiveMotion`, text fields are created using the `Text Field Tool`.

Using the `Selection` object you can control a user’s interaction with text fields and capture text from the text fields. You can position or get the position of the cursor in a text field.

Properties

None

Methods

<code>getBeginIndex()</code>	See “ <code>Selection.getBeginIndex()</code> Method” on page 245	Return the index of the beginning of the selection span. Return -1 if there is no currently selected field.
<code>getCaretIndex()</code>	See “ <code>Selection.getCaretIndex()</code> Method” on page 245	Return the index of the current caret (vertical text cursor).
<code>getEndIndex()</code>	See “ <code>Selection.getEndIndex()</code> Method” on page 246	Return the index of the end of the current selection. Returns -1 if there is no selection.
<code>getFocus()</code>	See “ <code>Selection.getFocus()</code> Method” on page 246	Return the name of the text field with the current focus.

<code>setFocus()</code>	See “ <code>Selection.setFocus()</code> Method” on page 247	Set the focus of the editable text field associated with the variable in the argument.
<code>setSelection()</code>	See “ <code>Selection.setSelection()</code> Method” on page 247	Set the beginning and ending index of the selection span.

Selection.getBeginIndex() Method

`selection.getBeginIndex()`

Description

The `getBeginIndex()` method returns the index of the first character of the selection span. It returns -1 if there is no currently selected field. The index is zero-based, where the first position in the text field is 0. If no text is selected, the position of the cursor is returned.

Parameters

None.

Returns

The index of the beginning of the selection span. Returns -1 if there is no currently selected field. If no text is selected, the position of the cursor is returned.

See Also

“`Selection.getEndIndex()` Method” on page 246

Selection.getCaretIndex() Method

`selection.getCaretIndex()`

Description

The `getCaretIndex()` method returns the index of the current caret (vertical text cursor) in the selection that currently has focus. If there is no current selection, -1 is returned.

Parameters

None.

Returns

The index of the current caret (vertical text cursor) in the selection that currently has focus. If there is no current selection, -1 is returned.

Selection.getEndIndex() Method

```
selection.getEndIndex()
```

Description

The `getEndIndex()` method returns the index of the last character of the selection span. It returns -1 if there is no currently selected field. The index is zero-based, where the first position in the text field is 0. If no text is selected, the position of the cursor is returned.

Parameters

None.

Returns

The index of the end of the selection span. Returns -1 if there is no currently selected field. If no text is selected, the position of the cursor is returned.

See Also

“Selection.getEndIndex() Method” on page 246

Selection.getFocus() Method

```
selection.getFocus()
```

Description

Returns the name of the text field that has the current focus. If no text field is selected, `null` is returned.

Parameters

None.

Returns

The name of the text field that has the current focus. If no text field is selected, `null` is returned.

See Also

“`Selection.setFocus()` Method” on page 247

Selection.setFocus() Method

```
selection.setFocus(variable)
```

Description

The `setFocus()` method sets the focus of the editable text field associated with the variable in the argument.

Parameters

variable

The name of the variable that is associated with the text field that gets focus.

See Also

“`Selection.getFocus()` Method” on page 246

Selection.setSelection() Method

```
selection.setSelection(start, end)
```

Description

The `setSelection()` method sets the beginning and ending indices of the selection span. The indices are zero-based, where the first position in the text field is 0. The method has no effect if there is no currently selected text field. If `start = end`, the cursor is set at that point in the text.

Parameters

<i>start</i>	Index of the beginning of the selection.
<i>end</i>	Index of the end of the selection.

See Also

“Selection.getBeginIndex() Method” on page 245, “Selection.getEndIndex() Method” on page 246

Sound Object

Description

The Sound object is used to create a new sound object. The Sound object can then be set and controlled to provide the sounds for an individual movie clip, including `_root`, or for the global timeline.

Constructor

`new Sound()`
`new Sound(target)`

Parameters

target (*Optional*) The name of the movie clip to which sound is applied. If not specified, the Sound object created controls all sounds in the global timeline.

Methods

<code>attachSound()</code>	See “Sound.attachSound() Method” on page 249	Add a new sound to a movie clip.
<code>getPan()</code>	See “Sound.getPan() Method” on page 249	Get the current pan value of a sound.
<code>getTransform()</code>	See “Sound.getTransform() Method” on page 250	Get the current panning transform value of a sound.

<code>getVolume()</code>	See “Sound.getVolume() Method” on page 251	Get the current volume of a sound.
<code>setPan()</code>	See “Sound.setPan() Method” on page 251	Set the current pan value of a sound.
<code>setTransform()</code>	See “Sound.setTransform() Method” on page 252	Set the current panning transform value of a sound.
<code>setVolume()</code>	See “Sound.setVolume() Method” on page 253	Set the current volume of a sound.
<code>start()</code>	See “Sound.start() Method” on page 253	Play a sound.
<code>stop()</code>	See “Sound.stop() Method” on page 254	Stop playing a sound or all sounds.

Sound.attachSound() Method

`sound.attachSound(exportName)`

Description

The `attachSound()` method attaches a sound to a Sound object. *exportName* is the sharing name of the sound. This is the sound file that was imported into LiveMotion, then assigned a sharing name using the Export palette.

Note: In preview, the sound that is attached is the local version only. If the “Use External Asset” feature is used from the Export palette, this will not be the same sound that is actually used when the SWF file is executing in the Flash player.

Parameters

<i>exportName</i>	Sharing name of the sound to attach. This name was assigned to the sound using the Export palette.
-------------------	--

Sound.getPan() Method

`sound.getPan()`

Description

The `getPan()` method gets the current pan value of the sound. This value was set by the last call to `setPan()`. The pan value is used to implement the balance function between audio channels. A value of `-100` routes all sound through the left channel only; a value of `100` routes all sound through the right channel. Values in between reflect the range between these two extremes, with a value of `0` indicating equal balance between the two channels.

Parameters

None.

Returns

The pan value of the sound (an integer in the range of `-100` to `100`).

See Also

“`Sound.setPan()` Method” on page 251

Sound.getTransform() Method

`sound.getTransform()`

Description

The `getTransform()` method gets the current panning transform values of a `Sound` object. The panning transform values are similar to the pan value, but they let you specify the relative amounts of right channel sound to be included in the left speaker, or vice versa.

Parameters

None

Returns

An object of type `Object` with the following properties:

- `l1`— the percentage of the left channel to play in the left speaker (an integer value in the range of `0` to `100`).

- **lr**—the percentage of the left channel to play in the right speaker (an integer value in the range of 0 to 100).
- **rl**—the percentage of the right channel to play in the left speaker (an integer value in the range of 0 to 100).
- **rr**—the percentage of the right channel to play in the right speaker (an integer value in the range of 0 to 100).

See Also

“Sound.setTransform() Method” on page 252

Sound.getVolume() Method

sound.getVolume()

Description

The `getVolume()` method gets the current volume of a sound. This is the volume set by the last `setVolume()` call. Values are from 0 - 100.

Parameters

None

Returns

The volume of the sound (an integer value in the range from 0 - 100).

See Also

“Sound.setVolume() Method” on page 253

Sound.setPan() Method

sound.setPan(*pan*)

Description

The `setPan()` method sets the current pan value of a `Sound` object. The pan value is used to implement the balance function between audio channels. A value of `-100` routes all sound through the left channel only; a value of `100` routes all sound through the right channel. Values in between reflect the range between these two extremes, with a value of `0` indicating equal balance between the two channels.

Parameters

<i>pan</i>	The pan value of the sound (an integer in the range of <code>-100</code> to <code>100</code>).
------------	---

See Also

“`Sound.getPan()` Method” on page 249

Sound.setTransform() Method

`sound.setTransform(transform)`

Description

The `setTransform()` method sets the current panning transform values of a `Sound` object. The panning transform values are similar to the pan value, but they let you specify the relative amounts of right channel sound to be included in the left speaker, or vice versa. The panning transform values are passed into the `setTransform()` method by instantiating an object of type `Object` and setting the following four properties:

- `l1`—the percentage of the left channel to play in the left speaker (an integer value in the range of `0` to `100`);
- `lr`—the percentage of the left channel to play in the right speaker (an integer value in the range of `0` to `100`);
- `r1`—the percentage of the right channel to play in the left speaker (an integer value in the range of `0` to `100`);
- `rr`—the percentage of the right channel to play in the right speaker (an integer value in the range of `0` to `100`).

An `ll` value of, for example, 50% indicates that 50% of the left channel content should be played through the left speaker. The default is `ll = 100%` and `rr = 100%`.

Parameters

`transform` An object with `ll`, `lr`, `rl`, and `rr` members.

Example

```
waveringVoice = new Object();  
voice.ll = 50;  
voice.lr = 50;  
voice.rl = 50;  
voice.rr = 50;  
sound.setTransform(waveringVoice);
```

See Also

“`Sound.getTransform()` Method” on page 250

Sound.setVolume() Method

`sound.setVolume(volume)`

Description

The `setVolume()` method sets the current volume of a sound.

Parameters

`volume` The volume of the sound (an integer in the range of 0 - 100).

See Also

“`Sound.setVolume()` Method” on page 253

Sound.start() Method

`sound.start(offset, loops)`

Description

The `start()` method plays a sound.

Parameters

<i>offset</i>	The number of seconds to wait before playing the sound.
<i>loops</i>	The number of times to loop the sound before stopping.

See Also

“`Sound.stop()` Method” on page 254

Sound.stop() Method

`sound.stop()`
`sound.stop(string)`

Description

The `stop()` method stops playing a sound or all sounds. All sounds are stopped if no argument is provided.

Parameters

<i>string</i>	(Optional) The name of the sound to stop playing.
---------------	---

See Also

“`Sound.start()` Method” on page 253

Soundbuftime Global Property

`_soundbuftime`

Description

The number of seconds before the movie clip starts to stream.

StartDrag() Global Function

```
startDrag(target)  
startDrag(target, lockCenter)  
startDrag(target, lockCenter, left, top, right, bottom)
```

Description

The `startDrag()` global function causes *target* to physically follow the mouse pointer. Use the `stopDrag()` global function to halt dragging.

Parameters

<i>target</i>	The pathname of the movie clip to drag.
<i>lockCenter</i>	(Optional) Boolean indicating whether the draggable <i>target</i> should be centered under the mouse pointer (<code>true</code>) or dragged relative to the mouse pointer's location when clicked (<code>false</code>). The default is <code>false</code> .
<i>left</i>	(Optional) x-coordinate boundary to the left of which <i>target</i> cannot be dragged.
<i>top</i>	(Optional) y-coordinate boundary above which <i>target</i> cannot be dragged.
<i>right</i>	(Optional) x-coordinate boundary to the right of which <i>target</i> cannot be dragged.
<i>bottom</i>	(Optional) y-coordinate boundary below which <i>target</i> cannot be dragged.

See Also

“`StopDrag()` Global Function” on page 256, “`MovieClip.startDrag()` Method” on page 220

Stop() Global Function

```
stop()
```

Description

The `stop()` global function stops a movie clip from playing.

Parameters

None.

See Also

“Play() Global Function” on page 241

StopAllSounds() Global Function

`stopAllSounds()`

Description

The `stopAllSounds()` method stops all the sounds in the global timeline. It doesn't stop the playhead.

Parameters

None

StopDrag() Global Function

`stopDrag()`

Description

The `stopDrag()` global function stops the dragging of the currently draggable object.

Parameters

None.

See Also

“StartDrag() Global Function” on page 255, “MovieClip.stopDrag() Method” on page 222

String() Global Function

```
String(value)
```

Description

The `String()` global function returns a string representation of *value*.

Parameters

<i>value</i>	A number, string, variable, or Boolean to convert to a string.
--------------	--

Returns

- If *value* is a Boolean, returns true or false.
- If *value* is a string, returns the string.
- If *value* is a number, returns a string representation of the number.
- If *value* is a MovieClip object, returns the absolute path in dot notation.
- If *value* is an object, returns a string representation of the object by referencing the string property for the object. If there is no string property for the object, the base class Object.toString() method is called.
- If *value* is undefined, returns an empty string.

See Also

“String Object” on page 257, “Object.toString() Method” on page 237

String Object

Description

The `String` object is a wrapper around the `String` primitive data type. Do not confuse a `String` literal with the `String` object. For example, the following code creates the `String` literal `s1` and also the `String` object `s2`:

```
s1 = "foo" // creates a string literal value
```

```
s2 = new String("foo") // creates a String object
```

You can call any of the methods of the `String` object on a string literal value— JavaScript automatically converts the string literal to a temporary `String` object, calls the method, then discards the temporary `String` object. You can also use the `String.length` property with a string literal.

Constructor

`new String(value)`

Parameters

<i>value</i>	The string encapsulated in the newly created object. If this parameter is not supplied, the string will be set to "".
--------------	---

Properties

<code>String.length</code>	See “String.length Property” The length of the string. on page 264
----------------------------	---

Methods

<code>charAt()</code>	See “String.charAt() Method”	Return the character at the specified index. on page 259
<code>charCodeAt()</code>	See “String.charCodeAt() Method” on page 260	Return the character at the specified index as a 16-bit integer.
<code>concat()</code>	See “String.concat() Method” on page 261	Concatenate the text of two or more strings and return the new string.
<code>fromCharCode()</code>	See “String.fromCharCode() Method” on page 261	Return a string created from the characters specified in the argument list.
<code>indexOf()</code>	See “String.indexOf() Method” on page 262	Return the index of the first occurrence of the specified value in the string, or -1 if not found.
<code>lastIndexOf()</code>	See “String.lastIndexOf() Method” on page 263	Return the index of the last occurrence of the specified value in the string, or -1 if not found.
<code>splice()</code>	See “String.slice() Method” on page 264	Return a string consisting of the sub-string specified in the argument list.

<code>split()</code>	See “String.split() Method” on page 265	Split a string into an array of sub-strings.
<code>substr()</code>	See “String.substr() Method” on page 266	Return the specified number of characters in a string beginning at the specified location.
<code>substring()</code>	See “String.substring() Method” on page 267	Return the characters between the two indices into the string.
<code>toLowerCase()</code>	See “String.toLowerCase() Method” on page 268	Convert the string to lowercase and return.
<code>toUpperCase()</code>	See “String.toUpperCase() Method” on page 269	Convert the string to uppercase and return.

String.charAt() Method

`String.charAt(index)`

Description

The `charAt()` method returns the specified character from the string. Characters in a string are indexed from left to right. The index of the first character is 0, and the index of the last character is the length of string minus 1 (zero-based indexing). If the index is out of range, JavaScript returns an empty string.

Parameters

<i>index</i>	An integer between 0 and the length of the string - 1 (zero-based indexing).
--------------	--

Returns

A string consisting of one character or an empty string (if the index is out of range).

Example

The following example displays characters at sequential locations in the string “Billy”:

```
var anyString="Billy"
trace("The character at index 0 is " + anyString.charAt(0));
trace("The character at index 1 is " + anyString.charAt(1));
```

```
trace("The character at index 2 is " + anyString.charAt(2));  
trace("The character at index 3 is " + anyString.charAt(3));  
trace("The character at index 4 is " + anyString.charAt(4));
```

These lines display the following:

```
The character at index 0 is B  
The character at index 1 is i  
The character at index 2 is l  
The character at index 3 is l  
The character at index 4 is y
```

See Also

“String.indexOf() Method” on page 262, “String.lastIndexOf() Method” on page 263

String.charCodeAt() Method

String.charCodeAt(*index*)

Description

The charCodeAt() method returns the ASCII value of the character at the given index.

Parameters

index An integer between 0 and the length of the string minus 1 (zero-based). The default value is 0.

Returns

The ASCII value of the character.

Example

```
trace("ICE".charCodeAt(0));// prints 73 - the ASCII value of "I"  
trace("ICE".charCodeAt());// prints 73 - the ASCII value of "I"  
trace("ICE".charCodeAt(1));// prints 67 - the ASCII value of "C"  
trace("ICE".charCodeAt(2));// prints 69 - the ASCII value of "E"
```

String.concat() Method

`String.concat(string1, string2, ..., stringN)`

Description

The `concat()` method concatenates the text of two or more strings and returns the new string.

Parameters

string1, string2, ..., stringN

Strings to concatenate to the current string.

Returns

The concatenated string.

Example

The following example combines two strings into a new string.

```
s1="Billy ";  
s2="and ";  
s3="Monica are fishing."  
trace(s1.concat(s2,s3)); // prints "Billy and Monica are fishing."
```

String.fromCharCode() Method

`String.fromCharCode(num1, ...numN)`

Description

The `fromCharCode()` method returns a string created by using the specified sequence of ASCII values. This method returns a string and not a `String` object. Because `fromCharCode` is a static method of `String`, you always use it as `String.fromCharCode()`, rather than as a method of a `String` object you created.

Parameters

num1, ..., numN

A sequence of ASCII values.

Returns

A string consisting of the characters provided as ASCII values.

Example

```
trace(String.fromCharCode(66,105,108,108,121)); //Returns "Billy"
```

String.indexOf() Method

`String.indexOf(searchValue, fromIndex)`

Description

The `indexOf()` method returns the index within the string of the first occurrence of the specified value, starting the search at *fromIndex* if provided. The method returns -1 if the value is not found.

Characters in a string are indexed from left to right. The index of the first character is 0, and the index of the last character is length of the string minus 1 (zero-based).

Parameters

<i>searchValue</i>	The string value for which to search.
<i>fromIndex</i>	(Optional) The location within the current string from which to start the search. Can be any integer between 0 and the length of the string minus 1 (zero-based). If this argument is not supplied, the default value is 0.

Returns

The position (zero-based) within the string where the first occurrence of *searchValue* was found, or -1 if it was not found.

Example

```
trace("Favorite beret".indexOf("Favorite")); // prints 0
trace("Favorite beret".indexOf("Hat")); // prints -1
trace("Favorite beret".indexOf("beret",0)); // prints 9
trace("Favorite beret".indexOf("beret",9)); // prints 9
```

The `indexOf()` method is case sensitive. For example, the following expression prints -1:

```
trace("Favorite beret".indexOf("favorite"));
```

See Also

“String.charAt() Method” on page 259, “String.lastIndexOf() Method” on page 263

String.lastIndexOf() Method

String.lastIndexOf(*searchValue*, *fromIndex*)

Description

The `lastIndexOf()` method returns the index within string of the last occurrence of the specified value, or -1 if not found. The string is searched backward, starting at *fromIndex*.

Characters in a string are indexed from left to right. The index of the first character is 0, and the index of the last character is the length of the string minus 1. The `lastIndexOf()` method is case sensitive.

Parameters

<i>searchValue</i>	A string representing the value to search for.
<i>fromIndex</i>	(Optional) The location within the current string from which to start the search. Can be any integer between 0 and the length of the string minus 1 (zero-based). If this argument is not supplied, the default value is 0.

Returns

The position (zero-based) within the string where the last occurrence of *searchValue* was found, or -1 if it was not found.

Example

```
trace("Billy".lastIndexOf("l")); // prints 3  
trace("Billy".lastIndexOf("l",2)); // prints 2  
trace("Billy".lastIndexOf("x")); // prints -1
```

See Also

“String.charAt() Method” on page 259, “String.indexOf() Method” on page 262

String.length Property

String.length

Description

The length property is the length of the string. A null string has a length of 0. This property can only be read.

Example

```
var x="Billy";  
trace("Length is " + x.length);//Prints "Length is 5"
```

String.slice() Method

String.slice(*beginslice*, *endSlice*)

Description

The slice() method extracts a section of a string and returns the new string. slice() extracts up to but not including *endSlice*. Indexing is zero-based. For example, slice(1, 4) extracts the second character through the fourth character (characters indexed 1, 2, and 3).

As a negative index, *endSlice* indicates an offset from the end of the string. For example, slice(2, -1) extracts the third character through the second to last character in the string.

Parameters

beginslice

The zero-based index at which to begin extraction.

endSlice

(Optional) The zero-based index at which to end extraction. If omitted, slice extracts to the end of the string.

Returns

A string.

Example

```
str1="Billy and Monica are ice skating.";
```

```
str2=str1.slice(10, -5);  
trace(str2); //Prints "Monica are ice ska"
```

See Also

“String.substring() Method” on page 267, “String.substr() Method” on page 266

String.split() Method

String.split(*delimiter*)

Description

The split() method splits a string into an array of strings by separating the string into substrings. The method returns the strings as an array.

When found, *delimiter* is removed from the string and the resulting substrings are returned in the array. If *delimiter* is omitted, the array contains one element consisting of the entire string.

Parameters

delimiter

(Optional) Specifies the character to use for delimiting. The delimiter is treated as a string. If *delimiter* is omitted, the array returned contains one element consisting of the entire string.

Returns

An array whose elements are the sub-strings.

Example

```
string = "Hello Billy. Let's go fishing."  
splits = string.split(" ");  
for(i=0; (splits[i] != "fishing."); ++i)  
{  
  trace(splits[i]);  
};  
trace(splits[i]);  
//Displays  
//"Hello"
```

```
//"Billy."  
//"Let's"  
//"go"  
//"fishing."
```

See Also

“String.charAt() Method” on page 259, “String.lastIndexOf() Method” on page 263,
“String.indexOf() Method” on page 262

String.substr() Method

String.substr(*start*, *length*)

Description

The `substr()` method returns the characters in a string beginning at the specified location through the specified number of characters. *start* is a character index. The index of the first character is 0, and the index of the last character is the length of the string minus 1 (zero-based). `substr()` begins extracting characters at *start* and collects *length* number of characters. If *start* is positive and is the length of the string or longer, `substr()` returns no characters.

If *start* is negative, `substr()` uses it as a character index from the end of the string. If *start* is negative and `abs(start)` is larger than the length of the string, `substr()` uses 0 as the start index.

If *length* is 0 or negative, `substr()` returns no characters. If *length* is omitted, *start* extracts characters to the end of the string.

Parameters

<i>start</i>	Location at which to begin extracting characters.
<i>length</i>	(Optional) The number of characters to extract.

Returns

A string.

Example

```
str = "phonecall"
trace("(1,2): " + str.substr(1,2));
trace("(-2,2): " + str.substr(-2,2));
trace("(1): " + str.substr(1));
trace("(20, 2): " + str.substr(20,2));
//prints
//(1,2): ho
//(-2,2): ll
//(1): honecall
//(20, 2):
```

See Also

“String.substring() Method” on page 267, “String.slice() Method” on page 264

String.substring() Method

String.substring(*indexA*, *indexB*)

Description

The `substring()` method returns a subset of a string by extracting characters from *indexA* up to but not including *indexB*. Specifically:

- If *indexA* is less than 0, *indexA* is treated as if it were 0.
- If *indexB* is greater than *stringName*.length, *indexB* is treated as if it were *stringName*.length.
- If *indexA* equals *indexB*, `substring` returns an empty string.
- If *indexB* is omitted, *indexB* extracts characters to the end of the string.

Parameters

<i>indexA</i>	An integer between 0 and the length of the string minus 1 (zero-based).
<i>indexB</i>	(Optional) An integer between 0 and the length of the string minus 1 (zero-based).

Returns

A string.

Example

```
var str="trolling";  
// Displays "tro"  
trace(str.substring(0,3));  
trace(str.substring(3,0));  
// Displays "lin"  
trace(str.substring(4,7));  
trace(str.substring(7,4));  
// Displays "trollin"  
trace(str.substring(0,7));  
// Displays "trolling"  
trace(str.substring(0,8));  
trace(str.substring(0,10));
```

See Also

“String.substr() Method” on page 266, “String.slice() Method” on page 264

String.toLowerCase() Method

String.toLowerCase()

Description

The toLowerCase() method returns the calling string value converted to lowercase without affecting the value of the string itself.

Parameters

None.

Returns

A lower case string.

Example

The following example displays the lowercase string “white house”:

```
var upperCase="WHITE HOUSE";  
trace(upperCase.toLowerCase()) //Displays "white house"
```

See Also

“String.toUpperCase() Method” on page 269

String.toUpperCase() Method

String.toUpperCase()

Description

The toUpperCase() method returns the calling string value converted to uppercase without affecting the value of the string itself.

Parameters

None.

Returns

An upper case string.

Example

The following example displays the string “WHITE HOUSE”:

```
var lowerCase="white house";  
trace(lowerCase.toUpperCase()); //displays "WHITE HOUSE"
```

See Also

“String.toLowerCase() Method” on page 268

TargetPath() Global Function

targetPath(*movieClip*)

Description

The `targetPath()` global function returns the path to the *movieClip*. in dot notation. To get the path in slash notation, use the `_target` property of `MovieClip`.

Parameters

movieClip

The movie clip for which the path is requested.

Returns

The path to the *movieClip*.

See Also

“MovieClip._target Property” on page 223

Trace() Global Function

```
trace(expression)
```

Description

The `trace()` global function evaluates *expression* and outputs the results to the console window. Used for debugging.

`trace()` is only useful from within LiveMotion. If you want to display the results of an expression to a text field of the executing SWF file, use the following code, where *display* is the var name of your text field:

```
_root.display = expression;
```

Parameters

expression

The expression to evaluate.

Example

```
trace(this);//prints MovieClip ("primitive" type)
trace(2 * 2);//prints 4
trace("Monica and Billy were here.");//prints "Monica and Billy were here"
```

Unescape() Global Function

```
unescape(stringExpression)
```

Description

The `unescape()` global function translates URL-encoded string *stringExpression* into a regular string. Use the `escape()` global function to URL-encode strings.

Parameters

<i>stringExpression</i>	A URL-encoded string with hexadecimal characters.
-------------------------	---

Example

```
//prints "Billy went fishing!#?!"
trace(unescape("Billy%20went%20fishing%21%24%23%21"));
```

See Also

“Escape() Global Function” on page 163

UnloadMovie() Global Function

```
unloadMovie(target)
```

Description

The `unloadMovie()` global function unloads a movie clip that was previously loaded using the `loadMovie()` global function, the `loadMovieNum()` global function, or the `MovieClip.loadMovie()` method.

Parameters

target

Where to unload the movie from. There are three possibilities:

1. If a number, the movie clip is unloaded from the level the number specifies;
2. If a string, the movie is unloaded from the movie clip specified by the path in the string;
- 3 If a movie clip object, the movie is unloaded from it.

See Also

“LoadMovie() Global Function” on page 184, “LoadMovieNum() Global Function” on page 185, “MovieClip.loadMovie() Method” on page 216, “MovieClip.unloadMovie() Method” on page 224

UnloadMovieNum() Global Function

`unloadMovieNum(number)`

Description

Same as `unloadMovie()` except that a number is used to specify the movie clip level. Therefore, it can only be used to unload movie clips previously loaded using the `loadMovie()` global function or the `loadMovieNum()` global function.

Parameters

number Integer specifying movie clip level.

See Also

“LoadMovie() Global Function” on page 184, “LoadMovieNum() Global Function” on page 185, “MovieClip.loadMovie() Method” on page 216, “MovieClip.unloadMovie() Method” on page 224

UpdateAfterEvent() Global Function

`updateAfterEvent()`

Description

The `updateAfterEvent()` global function updates the display when one of the following events occurs: `onMouseMove`, `onMouseDown`, `onMouseUp`, `onKeyDown`, `onKeyUp`. Place this function in the appropriate event handler to cause refresh to occur.

Parameters

None.

XML Object

Description

The XML object enables you to load, parse, send, build, and manipulate EXtensible Markup Language (XML) document trees. Unlike HTML, which uses a defined set of tags, XML allows you to define your own document tags. LiveMotion allows you to either build an XML document from scratch or read in and modify an existing XML document.

The following shows three levels of child nodes (the document itself is the parent):

```
<fish>//level 1 child node
  <type>Bass</type>//"type" tag is level 2 child node; "Bass" is level 3
</fish>
```

For example, the following creates an XML document:

```
xmlDocument = new XML("<fish><type>Bass</type></fish>");
```

The text can then be accessed as follows:

```
//prints "Bass"
trace(xmlDocument.firstChild.firstChild.firstChild.nodeValue);
```

Constructor

```
new XML()
new XML(source)
```

Parameters

source (Optional) Source XML document. If not provided, the XML object will contain a new, empty XML document.

Properties

<code>attributes</code>	See “XML.attributes Property” on page 276	An array listing all of the attributes of the given node.
<code>childNodes</code>	See “XML.childNodesProperty” on page 277	An array of child nodes of this node.
<code>contentType</code>	See “XML.contentType Property” on page 278	The MIME content type.
<code>docTypeDecl</code>	See “XML.docTypeDecl Property” on page 280	The DOCTYPE declaration of the XML document.
<code>firstChild</code>	See “XML.firstChild Property” on page 281	The first child of this node, null if there are no children.
<code>ignoreWhite</code>	See “XML.ignoreWhite Property” on page 282	Whether to ignore whitespace during XML parsing.
<code>lastChild</code>	See “XML.lastChild Property” on page 283	The last child of this node, null if there are no children.
<code>loaded</code>	See “XML.loaded Property” on page 285	true if the load or SendAndLoad operation has completed.
<code>nextSibling</code>	See “XML.nextSiblingProperty” on page 285	The next sibling of this node, null if this is the last node.
<code>nodeName</code>	See “XML.nodeNameProperty” on page 286	The tag name of this node. null if this node is a text node.
<code>nodeType</code>	See “XML.nodeType Property” on page 286	The type of this node. Either 1 if this node is an element node, or 3 if this node is a text node.
<code>nodeValue</code>	See “XML.nodeValue Property” on page 287	The text contained in this node. null if this node is not a text node.
<code>parentNode</code>	See “XML.parentNode Property” on page 289	The parent node of this node. null if this node is at the top of the hierarchy.

previousSibling	See “XML.previousSibling Property” on page 290	The previous sibling of this node, null if this is the first node.
status	See “XML.status Property” on page 292	Indicates whether there was an error parsing the XML document. 0 indicates no errors.
xmlDecl	See “XML.xmlDecl Property” on page 294	The DOCTYPE declaration of the XML document.

Methods

appendChild()	See “XML.appendChild() Method” on page 276	Append a child to this node.
cloneNode()	See “XML.cloneNode() Method” on page 278	Clone this node.
createElement()	See “XML.createElement() Method” on page 279	Create an XML element node.
createTextNode()	See “XML.createTextNode() Method” on page 280	Create an XML text node.
hasChildNodes()	See “XML.hasChildNodes() Method” on page 281	Return an indication whether this node has children.
insertBefore()	See “XML.insertBefore() Method” on page 283	Insert a child node before another child node.
load()	See “XML.load() Method” on page 284	Load and parse an XML document from the given URL.
parseXML()	See “XML.parseXML() Method” on page 289	Parse the given text as an XML document.
removeNode()	See “XML.removeNode() Method” on page 290	Delete this node and all of its children from the containing document.
send()	See “XML.send() Method” on page 291	Convert the XML document into a string and send it to the given URL.
sendAndLoad()	See “XML.sendAndLoad() Method” on page 292	Convert the XML document into a string and send it to the given URL. The receiving application is to reply with an XML document.
toString()	See “XML.toString() Method” on page 293	Convert the XML object into a string.

Event Handlers

onData	See “XML.onData() Event Handler” on page 287	Indicates that the XML doc parsing can begin.
onLoad	See “XML.onLoad() Event Handler” on page 288	Indicates that the load of an XML doc completed successfully.

XML.appendChild() Method

XML.appendChild(*node*)

Description

The `appendChild(node)` appends a child node to this node.

Parameters

node The node of the child to append.

Example

```
xmlDocument = new XML("<fish><type>Bass</type></fish>");
newDocument = new XML();
node = xmlDocument.firstChild.cloneNode(true);
newDocument.appendChild(node);
trace(newDocument.firstChild.nodeValue);
```

See Also

“XML.createElement() Method” on page 279, “XML.createTextNode() Method” on page 280, “XML.cloneNode() Method” on page 278, “XML.insertBefore() Method” on page 283

XML.attributes Property

XML.attributes

Description

The `attributes` property holds an array containing the attributes of the given node as properties. This property may be read or written.

Example

```
string = "<testtag name=\"value\"> Bass </testtag>"
xmlDocument = new XML(string);
nameAttribute = xmlDocument.firstChild.attributes.name;
trace(nameAttribute);//prints "value"
```

See Also

“XML.nodeType Property” on page 286

XML.childNodes Property

`XML.childNodes[n]`

Description

The `childNodes` property holds an array of child nodes of this node. Each element in the array is a reference to a child node of the XML object. Use the methods `appendChild()`, `insertBefore()`, and `removeNode()` to manipulate child nodes. This property can only be read.

Example

```
xmlDocument = new XML("<fish><type>Bass</type><color>grey</color></fish>");
trace(xmlDocument.childNodes[0].childNodes[0].nodeValue);//prints "type"
trace(xmlDocument.childNodes[0].childNodes[1].nodeValue);//prints "color"
```

See Also

“XML.firstChild Property” on page 281, “XML.hasChildNodes() Method” on page 281, “XML.lastChild Property” on page 283, “XML.nextSibling Property” on page 285, “XML.previousSibling Property” on page 290, “XML.appendChild() Method” on page 276, “XML.insertBefore() Method” on page 283, “XML.removeNode() Method” on page 290

XML.cloneNode() Method

XML.cloneNode(deep)

Description

The `cloneNode()` method clones this node and, optionally, all of its children.

Parameters

deep

A Boolean indicating whether a deep clone (all of the node's children as well as this node) should be performed. If `true`, a deep clone is performed. If `false`, only this node is cloned.

Returns

The cloned node or, optionally, all of its children as well.

Example

```
xmlDocument = new XML("<fish><type>Bass</type></fish>");
newDocument = new XML();
node = xmlDocument.firstChild.cloneNode(true);
newDocument.appendChild(node);
trace(newDocument.firstChild.nodeValue);
```

See Also

“XML.appendChild() Method” on page 276, “XML.createElement() Method” on page 279, “XML.createTextNode() Method” on page 280, “XML.insertBefore() Method” on page 283

XML.contentType Property

XML.contentType

Description

The `contentType` property holds the MIME content type. The MIME type is sent to the server when either the `XML.send()` or `XML.sendAndLoad()` methods are used. This property may be read or written. The default is `application/x-www-urlform-encoded`.

XML.createTextNode() Method

XML.createTextNode(*text*)

Description

The createTextNode() method creates a text node (as opposed to an element, or tag, node). The new node has no parent and no children. Note that the new node that is returned is not inserted into *XML*. To do that, you must use appendChild() or insertBefore().

As an example of a text node, examine the line:

```
<type>Bass</type>
```

type is a tag node, whereas Bass is the associated text node.

Parameters

text The text of the node to create.

Returns

The new text node.

Example

```
xmlDocument = new XML();  
node = xmlDocument.createElement("fish");  
xmlDocument.appendChild(node);  
textString = xmlDocument.createTextNode("Bass");  
xmlDocument.firstChild.appendChild(textString);  
trace(xmlDocument.firstChild.nodeValue);//prints "fish"  
trace(xmlDocument.firstChild.firstChild.nodeValue);//prints "Bass"
```

See Also

“XML.appendChild() Method” on page 276, “XML.cloneNode() Method” on page 278,
“XML.createElement() Method” on page 279, “XML.insertBefore() Method” on page 283

XML.docTypeDecl Property

XML.docTypeDecl

Description

The `docTypeDecl` property specifies the DOCTYPE declaration of the XML document. If there is no DOCTYPE, then this property is undefined. This property may be read or written.

Example

```
xmlDocument = new XML("<fish><type>Bass</type><color>grey</color></fish>");  
xmlDocument.docTypeDecl = "<!DOCTYPE salutation SYSTEM \"hello.dtd\">";  
trace(xmlDocument.docTypeDecl);  
//prints "<!DOCTYPE salutation SYSTEM \"hello.dtd\">"
```

See Also

“XML.xmlDecl Property” on page 294

XML.firstChild Property

XML.firstChild

Description

The `firstChild` property specifies the first child of this node, or null if there are no children. This property is read-only.

Example

```
xmlDocument = new XML("<fish><type>Bass</type></fish>");  
trace(xmlDocument.firstChild.nodeValue); //prints "fish"
```

See Also

“XML.childNodes Property” on page 277, “XML.lastChild Property” on page 283,
“XML.nextSibling Property” on page 285, “XML.previousSibling Property” on page 290

XML.hasChildNodes() Method

XML.hasChildNodes()

Description

The `hasChildNodes()` method returns an indication of whether this node has children.

Parameters

None

Returns

true if this node has children; false otherwise.

Example

```
xmlDocument = new XML("<fish><type>Bass</type></fish>");
if (xmlDocument.hasChildNodes())
{
    trace("yes");//prints "yes"
}
else
{
    trace("no");
}
```

See Also

“XML.childNodes Property” on page 277

XML.ignoreWhite Property

`XML.ignoreWhite`

Description

The `ignoreWhite` property stores a Boolean that indicates whether to ignore whitespace during XML parsing. The default is false. This property is read-only.

Note: Previous to release 41 of the Netscape Flash player and release 42 of the Internet Explorer Flash player, the Flash 5 player treated whitespace (carriage returns, tabs, spaces) as nodes. The ignore-white property is supported in the later releases. If your XML code needs to run on earlier versions of the Flash 5 player, you will need to include code that strips out whitespace from incoming XML documents.

```
temp = new Boolean(true);
trace(temp.valueOf()); //prints "true"
xmlDocument = new XML("<fish><type>Bass</type></fish>");
temp = xmlDocument.ignoreWhite;
trace(temp.valueOf()); //prints "false"
```

XML.insertBefore() Method

XML.insertBefore(newChild, insertBeforeChild)

Description

The `insertBefore()` method inserts a child node before another child node.

Parameters

<i>newChild</i>	The child to insert.
<i>insertBeforeChild</i>	The child to insert the new child before.

Example

```
xmlDocument = new XML("<color>grey</color>");
newNode = xmlDocument.createElement("<color>");
newText = xmlDocument.createTextNode("white");
newNode.appendChild(newText);
xmlDocument.insertBefore(newNode, xmlDocument.firstChild);
trace(xmlDocument.childNodes[0].childNodes[0].nodeValue); //prints "grey"
trace(xmlDocument.childNodes[1].childNodes[0].nodeValue); //prints "white"

//xmlDocument = new XML("<color>white</color><color>grey</color>");
//trace(xmlDocument.childNodes[0].childNodes[0].nodeValue); //prints "white"
//trace(xmlDocument.childNodes[1].childNodes[0].nodeValue); //prints "grey"
```

See Also

“XML.appendChild() Method” on page 276

XML.lastChild Property

XML.lastChild

Description

The `lastChild` property holds the last child of this node, or `null` if there are no children. It is equivalent to `childNodes[childNodes.length-1]`. This property is read-only. Do not use this method to manipulate child nodes—use the `appendChild()`, `insertBefore()`, and `removeNode()` methods instead/

Example

```
xmlDocument = new XML("<color>white</color><color2>grey</color2>");  
trace(xmlDocument.lastChild.nodeValue);//prints "color2"
```

See Also

“XML.childNodes Property” on page 277, “XML.firstChild Property” on page 281, “XML.nextSibling Property” on page 285, “XML.previousSibling Property” on page 290, “XML.appendChild() Method” on page 276, “XML.insertBefore() Method” on page 283, “XML.removeNode() Method” on page 290

XML.load() Method

XML.load(url)

Description

The `load()` method loads and parses an XML document from *url*. The load doesn’t happen immediately. Use the `XML.onLoad()` event handler to hold instructions for when the document has finished downloading. The loaded document replaces the contents of *XML* with the downloaded XML data. When `load()` is first executed, the `loaded` property is set to `false`; then, when the download is complete, the `loaded` property is set to `true` and the `onLoad()` method is invoked. The XML data is not parsed until the entire document is loaded. The parsing may be done using the default JavaScript parser, or the `XML.onData()` event handler may be used to write your own parser.

Parameters

<i>url</i>	URL of the document to load and parse. The URL must be in the same subdomain as the URL where the movie clip currently resides.
------------	---

Returns

The root of the parsed XML document.

See Also

“XML.loaded Property” on page 285, “XML.onLoad() Event Handler” on page 288, “XML.sendAndLoad() Method” on page 292, “XML.status Property” on page 292, “XML.onData() Event Handler” on page 287

XML.loaded Property

`XML.loaded`

Description

The loaded property holds `true` if the `load()` or `sendAndLoad()` operation has completed. Otherwise it holds `false`. This property is read-only.

See Also

“XML.load() Method” on page 284, “XML.onLoad() Event Handler” on page 288, “XML.sendAndLoad() Method” on page 292

XML.nextSibling Property

`XML.nextSibling`

Description

The `nextSibling` property holds the next sibling of this node, or `null` if this is the last node. This property is read-only. Don't use this method to attempt to manipulated child nodes. Use `appendChild()`, `insertBefore()`, and `removeNode()` to manipulate child nodes.

Example

```
xmlDocument = new XML("<color>white</color><color2>grey</color2>");
tempNode = xmlDocument.childNodes[0];
trace(tempNode.firstChild.nodeValue);//prints "white"
tempNode = tempNode.nextSibling;
```

```
trace(tempNode.firstChild.nodeValue);//prints "grey"
```

See Also

“XML.childNodes Property” on page 277, “XML.firstChild Property” on page 281, “XML.lastChild Property” on page 283, “XML.nodeName Property” on page 286, “XML.nodeValue Property” on page 287, “XML.previousSibling Property” on page 290, “XML.appendChild() Method” on page 276, “XML.insertBefore() Method” on page 283, “XML.removeNode() Method” on page 290

XML.nodeName Property

XML . nodeName

Description

The nodeName property holds the tag name of this node, or null if this node is a text node. If the tag is <myNode> then the nodeName is myNode. This property may only be read.

See Also

“XML.nodeType Property” on page 286, “XML.nodeValue Property” on page 287

XML.nodeType Property

XML . nodeType

Description

The nodeType property holds the type of this node. The possible values are 1 if this node is an element node, or 3 if this node is a text node. This property is read-only.

See Also

“XML.nodeName Property” on page 286, “XML.nodeValue Property” on page 287

XML.nodeValue Property

XML.nodeValue

Description

The `nodeValue` property holds the text contained in this node, or null if this node is an element node. This property may be read or written, though writing to it only makes sense if the node is a text node.

See Also

“XML.nodeName Property” on page 286, “XML.nodeType Property” on page 286

XML.onData() Event Handler

XML.onData(*source*)

Description

The `onData()` event handler executes automatically whenever raw XML source has finished loading into the XML document due to a previous `XML.load()` or `XML.sendAndLoad()` call. This allows you to write a custom function that handles the raw XML, or you can simply let the JavaScript XML parser execute on the raw XML. If the raw source that is received is undefined, the `onData()` event handler calls the `XML.onLoad()` event handler with the *success* parameter set to false. Otherwise, the `onData()` event handler parses the raw XML, sets the `XML.loaded` property to true, and calls the `XML.onLoad()` event handler with the *success* parameter set to true.

Parameters

source A string with the raw XML source.

Example

This example shows how to intercept the raw XML using the `onData()` event handler. It uses a function literal.

```
xmlDocument = new XML();
```

```
xmlDocument.onData = function(source)
{
    trace("Print the raw XML: \n" + source);
};
```

See Also

“XML.onLoad() Event Handler” on page 288, “XML.load() Method” on page 284,
“XML.sendAndLoad() Method” on page 292, “XML.loaded Property” on page 285

XML.onLoad() Event Handler

XML.onLoad(*result*)

Description

The onLoad() event handler is automatically executed whenever an external XML file is loaded into *XML* via the *XML*.load() or *XML*.sendAndLoad() method. By default, the onLoad() event handler is an empty function: you must provide your own callback handler, as shown in the example. The onLoad() event handler offers an alternative to monitoring the state to the *XML*.loaded property before proceeding with processing the downloaded XML.

Parameters

result Boolean indicating success (true) or failure (false) of the *XML*.load() or *XML*.sendAndLoad() method.

Example

```
xmlDocument = new XML();
xmlDocument.onLoad = xmlProcessor;
function xmlProcessor(success)
{
    //function body
};
```

See Also

“XML.onData() Event Handler” on page 287, “XML.load() Method” on page 284,
“XML.sendAndLoad() Method” on page 292

XML.parentNode Property

XML.parentNode

Description

The parentNode property holds the parent node of this node, or null if this node is at the top of the hierarchy. This property is read-only. Don't use this method to attempt to manipulated child nodes. Use appendChild(), insertBefore(), and removeNode() to manipulate child nodes.

Example

```
xmlDocument = new XML("<color>white</color><color2>grey</color2>");  
tempNode = xmlDocument.childNodes[0];  
trace(tempNode.parentNode.nodeValue);
```

See Also

"XML.childNodes Property" on page 277, "XML.firstChild Property" on page 281, "XML.lastChild Property" on page 283, "XML.previousSibling Property" on page 290, "XML.appendChild() Method" on page 276, "XML.insertBefore() Method" on page 283, "XML.removeNode() Method" on page 290

XML.parseXML() Method

XML.parseXML(*xml*)

Description

The parseXML() method parses *xml* as an XML document. It also replaces any existing XML in *XML* with the resulting XML tree from *xml*. This method is similar to the load() method, but the source is passed in as a string so can be used, for example, to pass in user input rather than just the contents of a url or file.

Parameters

<i>xml</i>	The text to parse.
------------	--------------------

See Also

“XML.load() Method” on page 284, “XML.status Property” on page 292

XML.previousSibling Property

XML.previousSibling

Description

The previousSibling property holds the previous sibling of this node, or null if this is the first node. This property is read-only. Don't use this method to attempt to manipulated child nodes. Use appendChild(), insertbefore(), and removeNode() to manipulate child nodes.

Example

```
xmlDocument = new XML("<color>white</color><color2>grey</color2>");
tempNode = xmlDocument.childNodes[1];
trace(tempNode.firstChild.nodeValue);//prints "grey"
tempNode = tempNode.previousSibling;
trace(tempNode.firstChild.nodeValue);//prints "white"
```

See Also

“XML.childNodes Property” on page 277, “XML.firstChild Property” on page 281, “XML.lastChild Property” on page 283, “XML.nextSibling Property” on page 285, “XML.nodeName Property” on page 286, “XML.nodeValue Property” on page 287, “XML.parentNode Property” on page 289, “XML.appendChild() Method” on page 276, “XML.insertBefore() Method” on page 283, “XML.removeNode() Method” on page 290

XML.removeNode() Method

XML.removeNode()

Description

The removeNode() method deletes this node and all of its children from the containing document.

Parameters

None.

See Also

“XML.appendChild() Method” on page 276

XML.send() Method

```
XML.send(url)  
XML.send(url, window)
```

Description

The `send()` method converts *XML* into a string and sends it to *url*. The document is sent via the POST method (in a separate HTTP packet, not attached to *url*). The response data is usually an HTML file for display in a browser window; this contrasts with the `sendAndLoad()` method, which receives XML for display directly from the Flash movie clip.

Parameters

<i>url</i>	The URL to which to send the XML text. The URL must be in the same subdomain as the URL where the movie clip was downloaded from.
<i>window</i>	(Optional) The window in which to display data returned by the server. This may be a custom name or one of the standard JavaScript windows (<code>_blank</code> , <code>_parent</code> , <code>_self</code> , or <code>_top</code>). Default is <code>_self</code> .

See Also

“XML.sendAndLoad() Method” on page 292, “XML.load() Method” on page 284, “XML.loaded Property” on page 285, “XML.onLoad() Event Handler” on page 288, “XML.status Property” on page 292

XML.sendAndLoad() Method

XML.sendAndLoad(url, responseXML)

Description

The `sendAndLoad()` method converts *XML* into a string and sends it to the given URL. The receiving application is supposed to reply with an XML document; this contrasts with the `send()` method, which receives an HTML file for display in a browser window. Any previous contents of *XML* is replaced with the parsed *responseXML*.

Parameters

<i>url</i>	The URL to which to send the XML text.
<i>responseXML</i>	The XML object into which to parse the response.

See Also

“XML.load() Method” on page 284, “XML.loaded Property” on page 285, “XML.send() Method” on page 291, “XML.status Property” on page 292, “XML.onData() Event Handler” on page 287, “XML.onLoad() Event Handler” on page 288

XML.status Property

XML.status

Description

The status property indicates whether there was an error parsing the XML document. This property is read-only. The possible error codes are:

- 0 — No error; parsing completed successfully.
- -2 — A CDATA section was not properly terminated.
- -3 — The XML declaration was not properly terminated.
- -4 — The DOCTYPE declaration was not properly terminated.
- -5 — A comment was not properly terminated.

- -6 — An XML element was malformed.
- -7 — Out of memory.
- -8 — An attribute value was not properly terminated.
- -9 — A start-tag was not matched with an end-tag.
- -10 — An end-tag was not properly matched with a start-tag.

Parsing occurs in several instances: when an XML object is first instantiated using the XML constructor, when an XML object is loaded using the `load()` or `sendAndLoad()` method, or XML is passed for parsing to the `parseXML()` method. Before checking the value of this property, check the `loaded` property to ensure that the `load()` or `sendAndLoad()` method has completed successfully.

See Also

“XML.load() Method” on page 284, “XML.loaded Property” on page 285, “XML.onLoad() Event Handler” on page 288, “XML.parseXML() Method” on page 289, “XML.sendAndLoad() Method” on page 292,

XML.toString() Method

`XML.toString()`

Description

The `toString()` method converts *XML* into a string. If you’re debugging with `trace()`, you probably won’t use this much.

Parameters

None

Returns

The string.

Example

```
xmlDocument = new XML("<color>white</color><color2>grey</color2>");  
trace(xmlDocument.toString());
```

```
//displays "<color>white</color><color2>grey</color2>"
```

See Also

“Object.toString() Method” on page 237, “XML.nodeValue Property” on page 287

XML.xmlDecl Property

`XML.xmlDecl`

Description

The `xmlDecl` property holds the XML declaration tag of the XML document. This property may be read or written.

Example

```
xmlDocument = new XML("<?xml version=\"1.0\"?><type>Bass</type>");  
trace(xmlDocument.xmlDecl);  
//prints "<?xml version=\"1.0\"?>"
```

See Also

“XML.docTypeDecl Property” on page 280

XMLnode Object

Description

The `XMLnode` object is the base class defining core properties and methods of nodes in an XML object hierarchy. Few programmers will need to access this object, but it is possible to use it to extend the default functionality of XML objects.

XMLSocket Object

Description

The `XMLSocket` object is used to implement a client socket that allows the Flash player to communicate with a server via an “open” connection. A socket connection is useful because it remains “open”—that is, a TCP/IP connection doesn’t have to be made between the client and the server each time communications occur between the two, as is required when the HTTP/IP protocol is used. This enables the Flash player to listen for incoming messages and quickly process them; it also allows it to respond quickly.

The three primary characteristics of an XML socket connection between a Flash player movie clip and a server are the following:

- XML messages are sent over a full-duplex (two-way) TCP/IP connection;
- Each XML message is a complete XML document, terminated by a zero byte (ASCII null character);
- An unlimited number of XML messages can be sent and received over a single connection.

If all of these requirements are not required by your application, consider using LiveMotion’s other Internet dynamic connectivity global functions, objects, and methods: `loadVariables()`, `loadVariablesNum()`, `MovieClip.loadVariables()`, `XML.load()`, `XML.sendAndLoad()`, and `XML.send()`.

Constructor

`new XMLSocket()`

Parameters

None

Methods

<code>close()</code>	See “XMLSocket.close() Method” on page 296	Close an open socket connection.
----------------------	--	----------------------------------

<code>connect()</code>	See “XMLSocket.connect() Method” on page 297	Create a connection to a specified server.
<code>send()</code>	See “XMLSocket.send() Method” on page 301	Send an XML object to the server.

Event Handlers

<code>onClose()</code>	See “XMLSocket.onClose() Event Handler” on page 298	Callback function that is called when a connection has closed.
<code>onConnect()</code>	See “XMLSocket.onConnect() Event Handler” on page 298	Callback function that is called when a connection is created.
<code>onData()</code>	See “XMLSocket.onData() Event Handler” on page 299	Callback function that is called when data is received but has not yet been parsed as XML.
<code>onXML()</code>	See “XMLSocket.onXML() Event Handler” on page 300	Callback function that is called when data has been received and parsed into an XML object hierarchy.

XMLSocket.close() Method

XMLSocket.close()

Description

The `close()` method closes an open socket connection.

Parameters

None

See Also

“XMLSocket.connect() Method” on page 297, “XMLSocket.onClose() Event Handler” on page 298

XMLSocket.connect() Method

XMLSocket.connect(host, port)

Description

The `connect()` method creates a connection to a specified server. If this method returns `true`, then the `onConnect()` event handler is invoked to complete the connection.

Parameters

<i>host</i>	Full DNS name or an IP address. <code>null</code> if you want to specify the current server (where the Flash movie clip is running). If the Flash Netscape plug-in or an ActiveX control is being used, the host must have the same subdomain as the host from which the Flash movie clip was downloaded.
<i>port</i>	TCP port to which you wish to establish a connection. Must be a number equal to or greater than 1024.

Returns

`true` if a connection is successfully created; `false` otherwise.

Example

```
function socketConnect(success)
{
    if (success)
    {
        trace("Full connection achieved");
    };
};

newSocket = new XMLSocket();
newSocket.onConnect = socketConnect();
if (newSocket.connect("http://www.adobe.com", 2000))
{
    trace("Initial connection achieved");
};
```

See Also

“XMLSocket.close() Method” on page 296, “XMLSocket.onConnect() Event Handler” on page 298

XMLSocket.onClose() Event Handler

XMLSocket.onClose() = functionName

Description

The `onClose()` event handler is a callback function that is called when a connection has closed. The default implementation of this method performs no action. To override default implementation, you must write your own handler, as shown in the example.

Parameters

<i>functionName</i>	The name of the function to call when the indicated connection has been closed. If omitted, the function does nothing.
---------------------	--

Example

```
newSocket = new XMLSocket();
newSocket.onClose = socketClosed;
function socketClosed()
{
    trace("The connection was closed by the server");
};
```

See Also

“XMLSocket.close() Method” on page 296

XMLSocket.onConnect() Event Handler

XMLSocket.onConnect(success)
XMLSocket.onConnect() = functionName

Description

The `onConnect()` event handler is a callback function that is called when a connection is created. The default implementation of this method performs no action. To override default implementation, you must write your own handler, as shown in the example.

Parameters

<i>success</i>	Boolean indicating success.
<i>functionName</i>	The name of the function to call when the indicated connection has been successfully created.

Returns

true if a connection is successfully created; false otherwise.

Example

```
function socketConnect(success)
{
    if (success)
    {
        trace("Full connection achieved");
    };
};

newSocket = new XMLSocket();
newSocket.onConnect = socketConnect();
if (newSocket.connect("http://www.adobe.com", 2000))
{
    trace("Initial connection achieved");
};
```

See Also

“XMLSocket.connect() Method” on page 297

XMLSocket.onData() Event Handler

XMLSocket.onData(source)

Description

The `onData()` event handler is a callback function that is called when data is received but has not yet been parsed as XML. The `onData()` event handler executes automatically whenever a zero byte (ASCII null character) is transmitted to Flash over *XMLSocket*. This allows you to write a function that handles the raw XML instead of the JavaScript parser that would otherwise be used before the XML is passed onto the *XMLSocket.onXML()* event handler. If you have not supplied `onData()` with a custom callback function, the XML is passed onto the default JavaScript XML parser.

Parameters

source A string containing loaded data, which is usually XML source code.

Example

The following shows how to implement the `onData()` event handler using a function literal.

```
newSocket = new XMLSocket();
newSocket.onData = function(source)
{
    trace("Print the raw XML: \n" + source);
};
```

See Also

“XMLSocket.onXML() Event Handler” on page 300; “XML.onData() Event Handler” on page 287

XMLSocket.onXML() Event Handler

XMLSocket.onXML(object)
XMLSocket.onXML() = functionName

Description

The `onXML()` event handler is a callback function that is called when data has been received and parsed into an XML object hierarchy. It has been parsed either by the default JavaScript parser or by a custom `onData()` event handler. The default implementation of this method performs no action. To override default implementation, you must write your own handler, as shown in the example.

Parameters

<i>object</i>	An instance of the XML object containing a parsed XML document that was received from the server.
<i>functionName</i>	Function to call when the specified XML object has been received

Example

The following shows how to implement the `onXML()` event handler using a function literal.

```
newSocket = new XMLSocket();
newSocket.onXML = function(object)
{
    trace("Handle the object in some way");
};
```

See Also

“XMLSocket.send() Method” on page 301, “XMLSocket.onData() Event Handler” on page 299, “XML.onLoad() Event Handler” on page 288

XMLSocket.send() Method

`XMLSocket.send(object)`

Description

The `send()` method converts *object* to a string and sends it to the server over the *XMLSocket* connection, followed by a zero byte (ASCII null character). This operation is asynchronous: the `send()` is initiated, but the operating system and networking software may not complete the transmission until some amount of time has passed.

Parameters

object XML object to send.

See Also

“XMLSocket.onXML() Event Handler” on page 300, “XMLSocket.send() Method” on page 301

Preliminary