macromedia®

FLASH™ MX 2004

ActionScript Language Reference

Acknowledgments

Director: Erick Vera

Project Management: Julee Burdekin, Erick Vera

Writing: Jay Armstrong, Jody Bleyle, Mary Burger, Francis Cheng, Jen deHaan, Stephanie Gowin, Phillip Heinz, Shimul Rahim, Samuel R. Neff

Managing Editor: Rosana Francescato

Editing: Mary Ferguson, Mary Kraemer, Noreen Maher, Antonio Padial, Lisa Stanziano, Anne Szabla, Barbara Milligan

Production Management: Patrice O'Neill

Media Design and Production: Adam Barnett, Christopher Basmajian, Aaron Begley, John Francis

Second Edition: June 2004

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

CONTENTS

CHAPTER 1
Introduction

This manual describes the syntax and use of ActionScript elements in Macromedia Flash MX 2004 and Macromedia Flash MX Professional 2004. To use examples in a script, copy the code example from this book and paste it into the Script pane or into an external script file. The manual lists all ActionScript elements—operators, keywords, statements, actions, properties, functions, classes, and methods.

Note: The examples are written in ActionScript 2.0, and will be unlikely to compile if you have specified ActionScript 1.0 or Flash Player 5 or earlier in the Flash tab of your FLA file's Publish Settings dialog box.

There are two types of entries in this manual:

- Individual entries for operators, keywords, functions, variables, properties, methods, and statements
- Class entries, which provide general information about built-in classes

Use the information in the sample entries to interpret the structure and conventions used in these types of entries.

This chapter contains the following sections:

Sample entry for most ActionScript elements

The following sample entry explains the conventions used for all ActionScript elements that are not classes.

Entry title

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

Availability

Unless otherwise noted, the Availability section tells which versions of Flash Player support the element. The supported Flash Player version is not the same as the version of Flash used to author the content. For example, if you use Macromedia Flash MX 2004 or Macromedia Flash MX Professional 2004 to create content for Flash Player 6, you can use only ActionScript elements that are available to Flash Player 6.

In a few cases, this section also indicates which version of the authoring tool supports an element. For an example, see System.setClipboard().

If an element is supported only in ActionScript 2.0, that information is also noted in this section.

Usage

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

Parameters

This section describes any parameters listed in the syntax.

Returns

This section identifies what, if any, values the element returns.

Description

This section identifies the type of element (for example, operator, method, function, and so on) and then describes how to use the element.

Example

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

See also

This section lists related ActionScript dictionary entries.

Sample entry for classes

The following sample dictionary entry explains the conventions used for built-in ActionScript classes. .

Entry title

The entry title provides the name of the class. The class name is followed by general descriptive information.

Method and property summary tables

Each class entry contains a table listing all of the associated methods. If the class has properties (often constants), event handlers, or event listeners, these elements are summarized in additional tables. All the elements listed in these tables also have their own entries, which follow the class entry.

Constructor

If a class requires that you use a constructor to access its methods and properties, the constructor is described in each class entry. This description has all of the standard elements (usage, description, and so on) of other entries.

Method, property, and event handler listings

The methods, properties, and event handlers of a class are listed alphabetically after the class entry.

Examples Folder

A set of ActionScript example files can be found in the HelpExamples folder in the Flash installation directory. Typical paths to this folder are:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/

The HelpExamples folder contains a set of FLA files that are complete projects with working ActionScript code. All ActionScript code can be found in the main Timeline of each FLA file or in external ActionScript (.as) files.

CHAPTER 2
ActionScript Language Reference

–– (decrement)

Availability

Flash Player 4.

Usage

```
--expression
expression--
```

Parameters

`expression` A number or a variable that evaluates to a number.

Returns

A number.

Description

Operator (arithmetic); a pre-decrement and post-decrement unary operator that subtracts 1 from the `expression`. The `expression` can be a variable, element in an array, or property of an object. The pre-decrement form of the operator (`--expression`) subtracts 1 from `expression` and returns the result. The post-decrement form of the operator (`expression--`) subtracts 1 from the `expression` and returns the initial value of `expression` (the value prior to the subtraction). For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The pre-decrement form of the operator decrements `x` to 2 (`x - 1 = 2`) and returns the result as `y`:

```
var x:Number = 3;
var y:Number = --x;
//y is equal to 2
```

The post-decrement form of the operator decrements `x` to 2 (`x - 1 = 2`) and returns the original value of `x` as the result `y`:

```
var x:Number = 3;
var y:Number = x--;
//y is equal to 3
```

The following example loops from 10 to 1, and each iteration of the loop decreases the counter variable `i` by 1.

```
for (var i = 10; i>0; i--) {
  trace(i);
}
```

++ (increment)

Availability

Flash Player 4.

Usage

++expression

expression++

Parameters

expression A number or a variable that evaluates to a number.

Returns

A number.

Description

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

The pre-increment form of the operator increments `x` to 2 (`x + 1 = 2`) and returns the result as `y`:

```
var x:Number = 1;
var y:Number = ++x;
trace("x:"+x);//traces x:2
trace("y:"+y);//traces y:2
```

The post-increment form of the operator increments `x` to 2 (`x + 1 = 2`) and returns the original value of `x` as the result `y`:

```
var x:Number = 1;
var y:Number = x++;
trace("x:"+x);//traces x:2
trace("y:"+y);//traces y:1
```

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

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

```
var i:Number = 0;
while (i++<5) {
  trace("this is execution "+i);
}
/* output:
  this is execution 1
  this is execution 2
  this is execution 3
  this is execution 4
  this is execution 5
*/
```

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

```
var a:Array = [];
// var a:Array = new Array();
var i:Number = 0;
while (i<10) {
  a.push(++i);
}
trace(a.toString());//traces: 1,2,3,4,5,6,7,8,9,10
```

This example also uses ++ as a pre-increment operator.

```
var a:Array = [];
for (var i = 1; i<=10; ++i) {
  a.push(i);
}
trace(a.toString());//traces: 1,2,3,4,5,6,7,8,9,10
```

This script shows the following result in the Output panel:

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

The following example uses ++ as a post-increment operator in a `while` loop:

```
// using a while loop
var a:Array = [];
// var a:Array = new Array();
var i:Number = 0;
while (i<10) {
  a.push(i++);
}
trace(a.toString());//traces 0,1,2,3,4,5,6,7,8,9
```

The following example uses ++ as a post-increment operator in a `for` loop:

```
// using a for loop
var a:Array = [];
// var a:Array = new Array();
for (var i = 0; i<10; i++) {
  a.push(i);
}
trace(a.toString());//traces 0,1,2,3,4,5,6,7,8,9
```

This script displays the following result in the Output panel:

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

! (logical NOT)

Availability

Flash Player 4.

Usage

`!expression`

Parameters

`expression` An expression or a variable that evaluates to a Boolean value.

Returns

A Boolean value.

Description

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

The following expressions illustrate the result of using the logical NOT (!) operator:

`!true` returns `false`

`!false` returns `true`

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

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

```
var happy:Boolean = false;
if (!happy) {
  trace("don't worry, be happy");//traces don't worry, be happy
}
```

The statement traces because `!false` equals `true`.

See also

!= (inequality), !== (strict inequality), && (logical AND), || (logical OR), == (equality), === (strict equality)

!= (inequality)

Availability

Flash Player 5.

Usage

expression1 != *expression2*

Parameters

None.

Returns

A Boolean value.

Description

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

- Numbers, strings, and Boolean values are compared by value.
- Objects, arrays, and functions are compared by reference.
- A variable is compared by value or by reference, depending on its type.

Comparison by value means what most people would expect *equals* to mean—that two expressions have the same value. For example, the expression (2 + 3) is equal to the expression (1 + 4) when compared by value.

Comparison by reference means that two expressions are equal only if they both refer to the same object, array, or function. Values inside the object, array, or function are not compared.

When comparing by value, if *expression1* and *expression2* are different data types, ActionScript will attempt to convert the data type of *expression2* to match that of *expression1*. For more information, see "Automatic data typing" and "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The following example illustrates the result of the inequality (`!=`) operator:

```
trace(5 != 8);// returns true
trace(5 != 5) //returns false
```

The following example illustrates the use of the inequality (`!=`) operator in an `if` statement:

```
var a:String = "David";
var b:String = "Fool";
if (a != b) {
  trace("David is not a fool");
}
```

The following example illustrates comparison by reference with two functions:

```
var a:Function = function() {
```

```
trace("foo");
};
var b:Function = function() {
trace("foo");
};
a(); // foo
b(); // foo
trace(a!=b); // true
a = b;
a(); // foo
b(); // foo
trace(a!=b); // false

// trace statement output:
foo
foo
true
foo
foo
false
```

The following example illustrates comparison by reference with two arrays:

```
var a:Array = [ 1, 2, 3 ];
var b:Array = [ 1, 2, 3 ];
trace(a); // 1, 2, 3
trace(b); // 1, 2, 3
trace(a!=b); // true
a = b;
trace(a); // 1, 2, 3
trace(b); // 1, 2, 3
trace(a!=b); // false

// trace statement output:
1,2,3
1,2,3
true
1,2,3
1,2,3
false
```

See also

! (logical NOT), !== (strict inequality), && (logical AND), || (logical OR), == (equality), === (strict equality)

!== (strict inequality)

Availability

Flash Player 6.

Usage

expression1 `!==` *expression2*

Parameters

None.

Returns

A Boolean value.

Description

Operator; tests for the exact opposite of the strict equality (===) operator. The strict inequality operator performs the same as the inequality operator except that data types are not converted. For more information, see "Automatic data typing" in *Using ActionScript in Flash*. If `expression1` is equal to `expression2`, and their data types are equal, the result is `false`. As with the strict equality (===) operator, the definition of *equal* depends on the data types being compared, as illustrated in the following list:

- Numbers, strings, and Boolean values are compared by value.
- Objects, arrays, and functions are compared by reference.
- A variable is compared by value or by reference, depending on its type.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The comments in the following code show the returned value of operations that use the equality (==), strict equality (===), and strict inequality (!==) operators:

```
var s1:String = "5";
var s2:String = "5";
var s3:String = "Hello";
var n:Number = 5;
var b:Boolean = true;

trace(s1 == s2); // true
trace(s1 == s3); // false
trace(s1 == n);  // true
trace(s1 == b);  // false

trace(s1 === s2); // true
trace(s1 === s3); // false
trace(s1 === n);  // false
trace(s1 === b);  // false

trace(s1 !== s2); // false
trace(s1 !== s3); // true
trace(s1 !== n);  // true
trace(s1 !== b);  // true
```

See also

! (logical NOT), != (inequality), && (logical AND), || (logical OR), == (equality), === (strict equality)

% (modulo)

Availability

Flash Player 4. In Flash 4 files, the `%` operator is expanded in the SWF file as `x - int(x/y) * y` and may not be as fast or as accurate in later versions of Flash Player.

Usage

`expression1 % expression2`

Parameters

None.

Returns

A number.

Description

Operator (arithmetic); calculates the remainder of *`expression1`* divided by *`expression2`*. If either of the *`expression`* parameters are non-numeric, the modulo (`%`) operator attempts to convert them to numbers. The *`expression`* can be a number or string that converts to a numeric value. For more information, see "Automatic data typing" and "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The following numeric example uses the modulo (`%)` operator:

```
trace(12%5); // traces 2
trace(4.3%2.1); // traces 0.0999999999999996
trace(4%4); // traces 0
```

The first trace returns 2, rather than 12/5 or 2.4, because the modulo (`%`) operator returns only the remainder. The second trace returns 0.0999999999999996 instead of the expected 0.1 because of the limitations of floating-point accuracy in binary computing.

See Also

`int`, / (division).

%= (modulo assignment)

Availability

Flash Player 4.

Usage

`expression1 %= expression2`

Parameters

None.

Returns

A number.

Description

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

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

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

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

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

See also

% (modulo)

& (bitwise AND)

Availability

Flash Player 5. In Flash 4, the AND (&) operator was used for concatenating strings. In Flash 5 and later, the AND (&) operator is a bitwise AND, and you must use the addition (`+`) operator to concatenate strings. Flash 4 files that use the AND (`&`) operator are automatically updated to use addition (+) operator when imported into the Flash 5 or later authoring environment.

Usage

```
expression1 & expression2
```

Parameters

None.

Returns

A 32-bit integer.

Description

Operator (bitwise); converts `expression1` and `expression2` to 32-bit unsigned integers, and performs a Boolean AND operation on each bit of the integer parameters. Floating-point numbers are converted to integers by discarding any digits after the decimal point. The result is a new 32-bit integer.

Positive integers are converted to an unsigned hex value with a maximum value of 4294967295 or 0xFFFFFFFF; values larger than the maximum have their most significant digits discarded when they are converted so the value is still 32-bit. Negative numbers are converted to an unsigned hex value using the two's complement notation, with the minimum being -2147483648 or 0x800000000; numbers less than the minimum are converted to two's complement with greater precision and then have the most significant digits discarded as well.

The return value is interpreted as a two's complement number with sign, so the return is an integer in the range -2147483648 to 2147483647.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The following example compares the bit representation of the numbers and returns 1 only if both bits at the same position are 1. In this ActionScript, you add 13 (binary 1101) and 11 (binary 1011) and return 1 only in the position where both numbers have a 1.

```
var insert:Number = 13;
var update:Number = 11;
trace(insert & update);// output : 9 (or 1001 binary)
```

In the numbers 13 and 11 the result is 9 because only the first and last positions in both numbers have the number 1.

The following examples show the behavior of the return value conversion:

```
trace(0xFFFFFFFF); // 4294967295
trace(0xFFFFFFFF & 0xFFFFFFFF); // -1
trace(0xFFFFFFFF & -1); // -1
trace(4294967295 & -1); // -1
trace(4294967295 & 4294967295); // -1
```

See Also

&= (bitwise AND assignment), ^ (bitwise XOR), ^= (bitwise XOR assignment), | (bitwise OR), |= (bitwise OR assignment), ~ (bitwise NOT)

&& (logical AND)

Availability

Flash Player 4.

Usage

`expression1 && expression2`

Parameters

None.

Returns

A Boolean value.

Description

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

Expression	Evaluates
`true&&true`	true
`true&&false`	false
`false&&false`	false
`false&&true`	false

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The following example uses the logical AND (`&&`) operator to perform a test to determine if a player has won the game. The `turns` variable and the `score` variable are updated when a player takes a turn or scores points during the game. The script shows "You Win the Game!" in the Output panel when the player's score reaches 75 or higher in 3 turns or less.

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

See also

! (logical NOT), != (inequality), !== (strict inequality), || (logical OR), == (equality), === (strict equality)

&= (bitwise AND assignment)

Availability

Flash Player 5.

Usage

`expression1 &= expression2`

Parameters

None.

Returns

A 32-bit integer; the value of *`expression1`* `&` *`expression2`*.

Description

Operator; assigns *`expression1`* the value of *`expression1`* `&` *`expression2`*. For example, the following two expressions are equivalent:

```
x &= y;
x = x & y;
```

For more information, see “Operator precedence and associativity” in *Using ActionScript in Flash*.

Example

The following example assigns the value `9` to `x`:

```
var x:Number = 15;
var y:Number = 9;
trace(x &= y); // output: 9
```

See also

& (bitwise AND), ^ (bitwise XOR), ^= (bitwise XOR assignment), | (bitwise OR), |= (bitwise OR assignment), ~ (bitwise NOT)

() (parentheses)

Availability

Flash Player 4.

Usage

(expression1 [, expression2])

(*expression1, expression2*)

function(*parameter1,..., parameterN*)

Parameters

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

`function` The function to be performed on the contents of the parentheses.

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

Returns

Nothing.

Description

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

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

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

Usage 3: Surrounds one or more parameters and passes them as parameters to the function outside the parentheses.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

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

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

Usage 2: The following example evaluates the function `foo()`, and then the function `bar()`, and returns the result of the expression `a + b`:

```
var a:Number = 1;
var b:Number = 2;
```

```
function foo() {
  a += b;
}
function bar() {
  b *= 10;
}
trace((foo(), bar(), a+b)); // outputs 23
```

Usage 3: The following example shows the use of parentheses with functions:

```
var today:Date = new Date();
trace(today.getFullYear()); // traces current year
function traceParameter(param):Void {
  trace(param);
}
traceParameter(2*2);//traces 4
```

See also

`with`

– (minus)

Availability

Flash Player 4.

Usage

(Negation) `-expression`

(Subtraction) *`expression1`* - *`expression2`*

Parameters

None.

Returns

An integer or floating-point number.

Description

Operator (arithmetic); used for negating or subtracting.

Usage 1: When used for negating, it reverses the sign of the numerical *`expression`*.

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

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

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

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

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

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

The result, 3, is an integer.

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

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

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

* (multiplication)

Availability

Flash Player 4.

Usage

```
expression1 * expression2
```

Parameters

None.

Returns

An integer or floating-point number.

Description

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

For more information, see “Operator precedence and associativity” in *Using ActionScript in Flash*.

Example

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

```
trace(2*3); // output: 6
```

The result, 6, is an integer.

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

```
trace(2.0 * 3.1416); // output: 6.2832
```

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

*= (multiplication assignment)

Availability

Flash Player 4.

Usage

```
expression1 *= expression2
```

Parameters

None.

Returns

The value of `expression1 * expression2`. If an expression cannot be converted to a numeric value, it returns `NaN` (not a number).

Description

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

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

For more information, see “Operator precedence and associativity” in *Using ActionScript in Flash*.

Example

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

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

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

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

See also

* (multiplication)

, (comma)

Availability

Flash Player 4.

Usage

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

Parameters

None.

Returns

The value of `expression1`, `expression2`, and so on.

Description

Operator; evaluates `expression1`, then `expression2`, and so on. This operator is primarily used with the `for` loop statement and is often used with the parentheses () operator.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The following example uses the comma (,) operator in a `for` loop:

```
for (i = 0, j = 0; i < 3 && j < 3; i++, j+=2) {
  trace("i = " + i + ", j = " + j);
}
// Output:
// i = 0, j = 0
// i = 1, j = 2
```

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

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

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

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

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

```
var v:Number = 0;
var z:Number = 0;
```

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

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

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

See also

() (parentheses)

. (dot)

Availability

Flash Player 4.

Usage

`object.property_or_method`

`instancename.variable`

`instancename.childinstance`

`instancename.childinstance.variable`

Parameters

`object` An instance of a class. The object can be an instance of any of the built-in ActionScript classes or a custom class. This parameter is always to the left of the dot (.) operator.

`property_or_method` The name of a property or method associated with an object. All the valid methods and properties for the built-in classes are listed in the method and property summary tables for that class. This parameter is always to the right of the dot (.) operator.

`instancename` The instance name of a movie clip.

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

`variable` A variable on the Timeline of the movie clip instance name to the left of the dot (`.`) operator.

Returns

The method, property or movie clip named on the right side of the dot.

Description

Operator; used to navigate movie clip hierarchies to access nested (child) movie clips, variables, or properties. The dot operator is also used to test or set the properties of an object or top-level class, execute a method of an object or top-level class, or create a data structure.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

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

```
person_mc.hairColor
```

The Flash 4 authoring environment did not support dot syntax, but Flash MX 2004 files published for Flash Player 4 can use the dot operator. The preceding example is equivalent to the following (deprecated) Flash 4 syntax:

```
/person_mc:hairColor
```

The following example creates a new movie clip within the `_root` scope. Then a text field is created inside the movie clip called `container_mc`. The text field's `autoSize` property is set to `true` and then populated with the current date.

```
this.createEmptyMovieClip("container_mc", this.getNextHighestDepth());
this.container_mc.createTextField("date_txt", this.getNextHighestDepth(), 0,
  0, 100, 22);
this.container_mc.date_txt.autoSize = true;
this.container_mc.date_txt.text = new Date();
```

The dot (.) operator is used when targeting instances within the SWF file and when you need to set properties and values for those instances.

: (type)

Availability

Flash Player 6.

Usage

```
[modifiers] var variableName:type
```

```
function functionName():type { ... }
```

```
function functionName(parameter1:type, ... , parameterN:type) [ :type ]{ ... }
```

Note: To use this operator, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box.

Parameters

variableName	An identifier for a variable.
type	A native data type, class name that you have defined, or interface name.
functionName	An identifier for a function.
parameter	An identifier for a function parameter.

Description

Operator; used for strict data typing; this operator specifies the variable type, function return type, or function parameter type. When used in a variable declaration or assignment, this operator specifies the variable's type; when used in a function declaration or definition, this operator specifies the function's return type; when used with a function parameter in a function definition, this operator specifies the variable type expected for that parameter.

Types are a compile-time-only feature. All types are checked at compile time, and errors are generated when there is a mismatch. (For more information, see Appendix A, "Error Messages," in *Using ActionScript in Flash.*) Mismatches can occur during assignment operations, function calls, and class member dereferencing using the dot (`.`) operator. To avoid type mismatch errors, use strict data typing (see "Strict data typing" in *Using ActionScript in Flash*).

Types that you can use include all native object types, classes and interfaces that you define, and Function and Void. The recognized native types are Boolean, Number, and String. All built-in classes are also supported as native types. For more information, see "About data types" in *Using ActionScript in Flash*.

For information about operator precedence, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

Usage 1: The following example declares a public variable named `userName` whose type is String and assigns an empty string to it:

```
var userName:String = "";
```

Usage 2: The following example shows how to specify a function's parameter type by defining a function named `randomInt()` that takes a parameter named `integer` of type Number:

```
function randomInt(integer:Number):Number {
  return Math.round(Math.random()*integer);
}
trace(randomInt(8));
```

Usage 3: The following example defines a function named `squareRoot()` that takes a parameter named `val` of the Number type and returns the square root of `val`, also a Number type:

```
function squareRoot(val:Number):Number {
  return Math.sqrt(val);
}
trace(squareRoot(121));
```

See Also

`var`, `function`.

?: (conditional)

Availability

Flash Player 4.

Usage

expression1 ? *expression2* : *expression3*

Parameters

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

expression2, *expression3* Values of any type.

Returns

The value of *expression2* or *expression3*.

Description

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

For more information, see “Operator precedence and associativity” in *Using ActionScript in Flash*.

Example

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

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

The following example shows a conditional statement written in shorthand:

```
var timecode:String = (new Date().getHours()<11) ? "AM" : "PM";
trace(timecode);
```

The same conditional statement could also be written in longhand, as shown in the following example:

```
if (new Date().getHours()<11) {
  var timecode:String = "AM";
} else {
  var timecode:String = "PM";
}
trace(timecode);
```

/ (division)

Availability

Flash Player 4.

Usage

expression1 / expression2

Parameters

expression A number or a variable that evaluates to a number.

Returns

A floating-point number.

Description

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

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The following statement divides the current width and height of the Stage, and then displays the result in the Output panel.

```
trace(Stage.width/2);
trace(Stage.height/2);
```

For a default Stage width and height of 550 x 400, the output is 275 and 150.

See Also

% (modulo)

// (comment delimiter)

Availability

Flash 1.

Usage

`// comment`

Parameters

comment	Any characters.

Returns

Nothing.

Description

Comment; indicates the beginning of a script comment. Any characters that appear between the comment delimiter (`//`) and the end-of-line character are interpreted as a comment and ignored by the ActionScript interpreter.

For more information, see "Deprecated Flash 4 operators" in *Using ActionScript in Flash*.

Example

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

```
// record the X position of the ball movie clip
var ballX:Number = ball_mc._x;
// record the Y position of the ball movie clip
var ballY:Number = ball_mc._y;
// record the X position of the bat movie clip
var batX:Number = bat_mc._x;
// record the Y position of the ball movie clip
var batY:Number = bat_mc._y;
```

See also

/* (comment delimiter)

/* (comment delimiter)

Availability

Flash Player 5.

Usage

```
/* comment */
```

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

Parameters

`comment` Any characters.

Returns

Nothing.

Description

Comment; indicates one or more lines of script comments. Any characters that appear between the opening comment tag (`/*`) and the closing comment tag (`*/`), are interpreted as a comment and ignored by the ActionScript interpreter. Use the // (comment delimiter) to identify single-line comments. Use the /* comment delimiter to identify comments on multiple successive lines. Leaving off the closing tag (`*/`) when using this form of comment delimiter returns an error message. Attempting to nest comments also returns an error message. After an opening comment tag (/*) is used, the first closing comment tag (`*/`) will end the comment, regardless of the number of opening comment tags (/*) placed between them.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The following script uses comment delimiters at the beginning of the script:

```
/* records the X and Y positions of the
ball and bat movie clips */

var ballX:Number = ball_mc._x;
var ballY:Number = ball_mc._y;
var batX:Number = bat_mc._x;
var batY:Number = bat_mc._y;
```

The following attempt to nest comments will result in an error message:

```
/* this is an attempt to nest comments.
/* But the first closing tag will be paired with the first opening tag */
and this text will not be interpreted as a comment */
```

See also

// (comment delimiter)

/= (division assignment)

Availability

Flash Player 4.

Usage

```
expression1 /= expression2
```

Parameters

`expression1,expression2` A number or a variable that evaluates to a number.

Returns

A number.

Description

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

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

For more information, see “Operator precedence and associativity” in *Using ActionScript in Flash*.

Example

The following code illustrates using the division assignment (`/=`) operator with variables and numbers:

```
var x:Number = 10;
var y:Number = 2;
x /= y;
trace(x); // output: 5
```

See Also

/ (division)

[] (array access)

Availability

Flash Player 4.

Usage

myArray = [*a0, a1,...aN*]

myArray[*i*] = *value*

myObject[*propertyName*]

Parameters

myArray The name of an array.

a0, a1,...aN Elements in an array; any native type or object instance, including nested arrays.

i An integer index greater than or equal to 0.

myObject The name of an object.

propertyName A string that names a property of the object.

Returns

Usage 1: A reference to an array.

Usage 2: A value from the array; either a native type or an object instance (including an Array instance).

Usage 3: A property from the object; either a native type or an object instance (including an Array instance).

Description

Operator; initializes a new array or multidimensional array with the specified elements (*a0*, and so on), or accesses elements in an array. The array access operator lets you dynamically set and retrieve instance, variable, and object names. It also lets you access object properties.

Usage 1: An array is an object whose properties are called *elements*, which are each identified by a number called an *index*. When you create an array, you surround the elements with the array access ([]) operator (or *brackets*). An array can contain elements of various types. For example, the following array, called `employee`, has three elements; the first is a number and the second two are strings (inside quotation marks):

```
var employee:Array = [15, "Barbara", "Jay"];
```

You can nest brackets to simulate multidimensional arrays. You can nest arrays up to 256 levels deep. The following code creates an array called `ticTacToe` with three elements; each element is also an array with three elements:

```
var ticTacToe:Array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
// Select Debug > List Variables in test mode
// to see a list of the array elements.
```

Usage 2: Surround the index of each element with brackets ([]) to access it directly; you can add a new element to an array, or you can change or retrieve the value of an existing element. The first index in an array is always 0, as shown in the following example:

```
var my_array:Array = new Array();
my_array[0] = 15;
my_array[1] = "Hello";
my_array[2] = true;
```

You can use brackets ([]) to add a fourth element, as shown in the following example:

```
my_array[3] = "George";
```

You can use brackets ([]) to access an element in a multidimensional array. The first set of brackets identifies the element in the original array, and the second set identifies the element in the nested array. The following lines of code send the number 6 to the Output panel.

```
var ticTacToe:Array = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
trace(ticTacToe[1][2]);// output: 6
```

Usage 3: You can use the array access ([]) operator instead of the `eval()` function to dynamically set and retrieve values for movie clip names or any property of an object. The following line of code sends the number 6 to the Output panel.

```
name["mc" + i] = "left_corner";
```

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The following example shows two ways to create a new empty Array object; the first line uses brackets ([]):

```
var my_array:Array = [];
var my_array:Array = new Array();
```

The following example creates an array called `employee_array` and uses the `trace()` statement to send the elements to the Output panel. In the fourth line, an element in the array is changed, and the fifth line sends the newly modified array to the Output panel:

```
var employee_array = ["Barbara", "George", "Mary"];
trace(employee_array); // output: Barbara,George,Mary
employee_array[2] = "Sam";
trace(employee_array); // output: Barbara,George,Sam
```

In the following example, the expression inside the brackets (`"piece" + i`) is evaluated and the result is used as the name of the variable to be retrieved from the `my_mc` movie clip. In this example, the variable `i` must live on the same Timeline as the button. If the variable `i` is equal to 5, for example, the value of the variable `piece5` in the `my_mc` movie clip is displayed in the Output panel:

```
  myBtn_btn.onRelease = function() {
    x = my_mc["piece"+i];
    trace(x);
  };
```

In the following example, the expression inside the brackets is evaluated, and the result is used as the name of the variable to be retrieved from movie clip `name_mc`:

```
name_mc["A" + i];
```

If you are familiar with the Flash 4 ActionScript slash syntax, you can use the `eval()` function to accomplish the same result:

```
  eval("name_mc.A" & i);
```

You can use the following ActionScript to loop over all objects in the `_root` scope, which is useful for debugging:

```
for (i in _root) {
  trace(i+": "+_root[i]);
}
```

You can also use the array access ([]) operator on the left side of an assignment statement to dynamically set instance, variable, and object names:

```
employee_array[2] = "Sam";
```

See also

Array class, Object class

^ (bitwise XOR)

Availability

Flash Player 5.

Usage

expression1 ^ *expression2*

Parameters

expression1,*expression2* A number.

Returns

A 32-bit integer.

Description

Operator (bitwise); converts *expression1* and *expression2* to 32-bit unsigned integers, and returns a 1 in each bit position where the corresponding bits in *expression1* or *expression2*, but not both, are 1. Floating-point numbers are converted to integers by discarding any digits after the decimal point. The result is a new 32-bit integer.

Positive integers are converted to an unsigned hex value with a maximum value of 4294967295 or 0xFFFFFFFF; values larger than the maximum have their most significant digits discarded when they are converted so the value is still 32-bit. Negative numbers are converted to an unsigned hex value via the two's complement notation, with the minimum being -2147483648 or 0x800000000; numbers less than the minimum are converted to two's complement with greater precision and also have the most significant digits discarded.

The return value is interpreted as a two's complement number with sign, so the return value will be an integer in the range -2147483648 to 2147483647.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The following example uses the bitwise XOR operator on the decimals 15 and 9, and assigns the result to the variable `x`:

```
// 15 decimal = 1111 binary
// 9 decimal = 1001 binary
var x:Number = 15 ^ 9;
trace(x);
// 1111 ^ 1001 = 0110
// returns 6 decimal (0110 binary)
```

See also

& (bitwise AND), &= (bitwise AND assignment), ^= (bitwise XOR assignment), | (bitwise OR), |= (bitwise OR assignment), ~ (bitwise NOT)

^= (bitwise XOR assignment)

Availability

Flash Player 5.

Usage

`expression1 ^= expression2`

Parameters

`expression1,expression2` Integers and variables.

Returns

A 32-bit integer. Specifically, the value of *`expression1 ^ expression2`*.

Description

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

```
x ^= y
x = x ^ y
```

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The following example shows a bitwise XOR assignment (^=) operation:

```
// 15 decimal = 1111 binary
var x:Number = 15;
// 9 decimal = 1001 binary
var y:Number = 9;
trace(x ^= y);
// returns 6 decimal (0110 binary)
```

See also

& (bitwise AND), &= (bitwise AND assignment), ^ (bitwise XOR), | (bitwise OR), |= (bitwise OR assignment), ~ (bitwise NOT)

{} (object initializer)

Availability

Flash Player 5.

Usage

```
object = {name1: value1, name2: value2,...nameN: valueN}
{expression1; [...expressionN]}
```

Parameters

`object` The object to create.

`name1,2,...N` The names of the properties.

`value1,2,...N` The corresponding values for each `name` property.

Returns

Usage 1: An Object object.

Usage 2: Nothing, except when a function has an explicit `return` statement, in which case the return type is specified in the function implementation.

Description

Operator; creates a new object and initializes it with the specified *`name`* and *`value`* property pairs. Using this operator is the same as using the `new Object` syntax and populating the property pairs using the assignment operator. The prototype of the newly created object is generically named the Object object.

This operator is also used to mark blocks of contiguous code associated with flow control statements (`for`, `while`, `if`, `else`, `switch`) and functions.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The first line of the following code creates an empty object using the object initializer ({}) operator; the second line creates a new object using a constructor function:

```
var object:Object = {};
var object:Object = new Object();
```

The following example creates an object `account` and initializes the properties `name`, `address`, `city`, `state`, `zip`, and `balance` with accompanying values:

```
var account:Object = {name:"Macromedia, Inc.",

  address:"600 Townsend Street",
  city:"San Francisco",
  state:"California",
  zip:"94103",
  balance:"1000"};

for (i in account) {
  trace("account."+i+" = "+account[i]);
}
```

The following example shows how array and object initializers can be nested within each other:

```
var person:Object = {name:"Gina Vechio",
  children:["Ruby", "Chickie", "Puppa"]};
```

The following example uses the information in the previous example and produces the same result using constructor functions:

```
var person:Object = new Object();
person.name = "Gina Vechio";
person.children = new Array();
person.children[0] = "Ruby";
person.children[1] = "Chickie";
person.children[2] = "Puppa";
```

The previous ActionScript example can also be written in the following format:

```
var person:Object = new Object();
person.name = "Gina Vechio";
person.children = new Array("Ruby", "Chickie", "Puppa");
```

See also

`[] (array access)`, `new`, Object class

| (bitwise OR)

Availability

Flash Player 5.

Usage

`expression1 | expression2`

Parameters

`expression1,expression2` A number.

Returns

A 32-bit integer.

Description

Operator (bitwise); converts *`expression1`* and *`expression2`* to 32-bit unsigned integers, and returns a 1 in each bit position where the corresponding bits of either *`expression1`* or *`expression2`* are 1. Floating-point numbers are converted to integers by discarding any digits after the decimal point. The result is a new 32-bit integer.

Positive integers are converted to an unsigned hex value with a maximum value of 4294967295 or 0xFFFFFFFF; values larger than the maximum have their most significant digits discarded when they are converted so the value is still 32-bit. Negative numbers are converted to an unsigned hex value via the two's complement notation, with the minimum being -2147483648 or 0x800000000; numbers less than the minimum are converted to two's complement with greater precision and also have the most significant digits discarded.

The return value is interpreted as a two's complement number with sign, so the return value will be an integer in the range -2147483648 to 2147483647.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The following is an example of a bitwise OR (|) operation:

```
// 15 decimal = 1111 binary
var x:Number = 15;
// 9 decimal = 1001 binary
var y:Number = 9;
// 1111 | 1001 = 1111
trace(x | y);  // returns 15 decimal (1111 binary)
```

Don't confuse the single `| (bitwise OR)` with `|| (logical OR)`.

See also

& (bitwise AND), &= (bitwise AND assignment), ^ (bitwise XOR), ^= (bitwise XOR assignment), |= (bitwise OR assignment), ~ (bitwise NOT)

|| (logical OR)

Availability

Flash Player 4.

Usage

`expression1 || expression2`

Parameters

`expression1, expression2` A Boolean value or an expression that converts to a Boolean value.

Returns

A Boolean value.

Description

Operator (logical); evaluates *`expression1`* (the expression on the left side of the operator) and returns `true` if the expression evaluates to `true`. If *`expression1`* evaluates to `false`, *`expression2`* (the expression on the right side of the operator) is evaluated. If *`expression2`* evaluates to `false`, the final result is `false`; otherwise, it is `true`.

If you use a function call as *`expression2`*, the function will not be executed by that call if *`expression1`* evaluates to true.

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

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

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

```
var x:Number = 10;
var y:Number = 250;
var start:Boolean = false;
if ((x>25) || (y>200) || (start)) {
  trace("the logical OR test passed"); // output: the logical OR test passed
}
```

The message `the logical OR test passed` appears because one of the conditions in the `if` statement is true (`y>200`). Although the other two expressions evaluate to `false`, as long as one condition evaluates to `true`, the `if` block executes.

The following example demonstrates how using a function call as *`expression2`* can lead to unexpected results. If the expression on the left of the operator evaluates to `true`, that result is returned without evaluating the expression on the right (the function `fx2()` is not called).

```
function fx1():Boolean {
  trace("fx1 called");
  return true;
}
function fx2():Boolean {
  trace("fx2 called");
  return true;
}
if (fx1() || fx2()) {
  trace("IF statement entered");
}
/* The following is sent to the Output panel:
  fx1 called
  IF statement entered
*/
```

See also

! (logical NOT), != (inequality), !== (strict inequality), && (logical AND), == (equality), === (strict equality)

|= (bitwise OR assignment)

Availability

Flash Player 5.

Usage

```
expression1 |= expression2
```

Parameters

expression1,expression2 A number or variable.

Returns

An integer.

Description

Operator (bitwise compound assignment); assigns `expression1` the value of `expression1 | expression2`. For example, the following two statements are equivalent:

```
x |= y;
x = x | y;
```

For more information, see “Operator precedence and associativity” in *Using ActionScript in Flash*.

Example

The following example uses the bitwise OR assignment (`|=`) operator:

```
// 15 decimal = 1111 binary
var x:Number = 15;
// 9 decimal = 1001 binary
var y:Number = 9;
// 1111 |= 1001 = 1111
trace(x |= y);  // returns 15 decimal (1111 binary)
```

See also

& (bitwise AND), &= (bitwise AND assignment), ^ (bitwise XOR), ^= (bitwise XOR assignment), | (bitwise OR), |= (bitwise OR assignment), ~ (bitwise NOT)

~ (bitwise NOT)

Availability

Flash Player 5.

Usage

```
~ expression
```

Parameters

`expression`	A number.

Returns

A 32-bit integer.

Description

Operator (bitwise); also known as the one's complement operator or the bitwise complement operator. Converts the `expression` to a 32-bit signed integer, and then applies a bitwise one's complement. That is, every bit that is a 0 is set to 1 in the result, and every bit that is a 1 is set to 0 in the result. The result is a signed 32-bit integer.

For example, the hex value 0x7777 is represented as this binary number:

0111011101110111

The bitwise negation of that hex value, ~0x7777, is this binary number:

1000100010001000

In hexadecimal, this is 0x8888. Therefore, ~0x7777 is 0x8888.

The most common use of bitwise operators is for representing *flag bits* (Boolean values packed into 1 bit each).

Floating-point numbers are converted to integers by discarding any digits after the decimal point. Positive integers are converted to an unsigned hex value with a maximum value of 4294967295 or 0xFFFFFFFF; values larger than the maximum have their most significant digits discarded when they are converted so the value is still 32-bit. Negative numbers are converted to an unsigned hex value via the two's complement notation, with the minimum being -2147483648 or 0x800000000; numbers less than the minimum are converted to two's complement with greater precision and also have the most significant digits discarded.

The return value is interpreted as a two's complement number with sign, so the return value is an integer in the range -2147483648 to 2147483647.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The following example demonstrates a use of the bitwise NOT (~) operator with flag bits:

```
var ReadOnlyFlag:Number = 0x0001;
// defines bit 0 as the read-only flag
var flags:Number = 0;
trace(flags);
```

```
/* To set the read-only flag in the flags variable, the following code uses the
   bitwise OR: */
flags |= ReadOnlyFlag;
trace(flags);
/* To clear the read-only flag in the flags variable, first construct a mask by
   using bitwise NOT on ReadOnlyFlag. In the mask, every bit is a 1 except for
   the read-only flag. Then, use bitwise AND with the mask to clear the read-
   only flag. The following code constructs the mask and performs the bitwise
   AND: */
flags &= ~ReadOnlyFlag;
trace(flags);
/* output:
   0
   1
   0
*/
```

See also

& (bitwise AND), &= (bitwise AND assignment), ^ (bitwise XOR), ^= (bitwise XOR assignment), | (bitwise OR), |= (bitwise OR assignment)

+ (addition)

Availability

Flash Player 4; Flash Player 5.In Flash 5 and later, + is either a numeric operator or a string concatenator depending on the data type of the parameter. In Flash 4, + is only a numeric operator. Flash 4 files that are brought into the Flash 5 or later authoring environment undergo a conversion process to maintain data type integrity. The following example illustrates the conversion of a Flash 4 file containing a numeric quality comparison:

Flash 4 file:

```
x + y
```

Converted Flash 5 or later file:

```
Number(x) + Number(y)
```

Usage

`expression1 + expression2`

Parameters

`expression1,expression2` A number or string.

Returns

A string, integer, or floating-point number.

Description

Operator; adds numeric expressions or concatenates (combines) strings. If one expression is a string, all other expressions are converted to strings and concatenated.

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

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

Usage 1: The following example concatenates two strings and displays the result in the Output panel.

```
var name:String = "Cola";
var instrument:String = "Drums";
trace(name+" plays "+instrument);  // output: Cola plays Drums
```

Usage 2: This statement adds the integers 2 and 3 and displays the resulting integer, 5, in the Output panel:

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

This statement adds the floating-point numbers 2.5 and 3.25 and displays the resulting floating-point number, 5.75, in the Output panel

```
trace(2.5+3.25);  // output: 5.75
```

Usage 3: Variables associated with dynamic and input text fields have the data type String. In the following example, the variable `deposit` is an input text field on the Stage. After a user enters a deposit amount, the script attempts to add `deposit` to `oldBalance`. However, because `deposit` is a String data type, the script concatenates (combines to form one string) the variable values rather than summing them.

```
var oldBalance:Number = 1345.23;
var currentBalance = deposit_txt.text + oldBalance;
trace(currentBalance);
```

For example, if a user enters 475 in the deposit text field, the `trace()` statement sends the value 4751345.23 to the Output panel.

To correct this, use the `Number()` function to convert the string to a number, as in the following:

```
var oldBalance:Number = 1345.23;
var currentBalance:Number = Number(deposit_txt.text) + oldBalance;
trace(currentBalance);
```

The following example shows how numeric sums to the right of a string expression are not calculated:

```
var a:String = 3 + 10 + "asdf";
trace(a); // 13asdf
var b:String = "asdf" + 3 + 10;
trace(b); // asdf310
```

+= (addition assignment)

Availability

Flash Player 4.

Usage

```
expression1 += expression2
```

Parameters

`expression1,expression2` A number or string.

Returns

A string or a number.

Description

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

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

This operator also performs string concatenation. All the rules of the addition (+) operator apply to the addition assignment `(+=)` operator.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

Usage 1: This example uses the += operator with a string expression and sends "My name is Gilbert" to the Output panel.

```
 var x1:String = "My name is ";
x1 += "Gilbert";
trace(x1);  // output: My name is Gilbert
```

Usage 2: The following example shows a numeric use of the addition assignment (`+=`) operator:

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

See also

+ (addition)

< (less than)

Availability

Flash Player 4; Flash Player 5. In Flash 5 and later, the `<` (less than) operator is a comparison operator capable of handling various data types. In Flash 4, `<` is an numeric operator. Flash 4 files that are brought into the Flash 5 or later authoring environment undergo a conversion process to maintain data type integrity. The following illustrates the conversion of a Flash 4 file containing a numeric quality comparison.

Flash 4 file:

```
x < y
```

Converted Flash 5 or later file:

```
Number(x) < Number(y)
```

Usage

`expression1 < expression2`

Parameters

`expression1,expression2` A number or string.

Returns

A Boolean value.

Description

Operator (comparison); compares two expressions and determines whether *`expression1`* is less than *`expression2`*; if so, the operator returns `true`. If *`expression1`* is greater than or equal to *`expression2`*, the operator returns `false`. String expressions are evaluated using alphabetical order; all capital letters come before lowercase letters.

For more information, see “Operator precedence and associativity” in *Using ActionScript in Flash*.

Example

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

```
trace(3<10); // true
trace(10<3); // false
trace("Allen"<"Jack"); // true
trace("Jack"<"Allen"); //false
trace("11"<"3"); // true
trace("11"<3); // false (numeric comparison)
trace("C"<"abc"); // true
trace("A"<"a"); // true
```

<< (bitwise left shift)

Availability

Flash Player 5.

Usage

`expression1 << expression2`

Parameters

`expression1`	A number or expression to be shifted left.
`expression2`	A number or expression that converts to an integer from 0 to 31.

Returns

A 32-bit integer.

Description

Operator (bitwise); converts `expression1` and `expression2` to 32-bit integers, and shifts all the bits in `expression1` to the left by the number of places specified by the integer resulting from the conversion of `expression2`. The bit positions that are emptied as a result of this operation are filled in with 0 and bits shifted off the left end are discarded. Shifting a value left by one position is the equivalent of multiplying it by 2.

Floating-point numbers are converted to integers by discarding any digits after the decimal point. Positive integers are converted to an unsigned hex value with a maximum value of 4294967295 or 0xFFFFFFFF; values larger than the maximum have their most significant digits discarded when they are converted so the value is still 32-bit. Negative numbers are converted to an unsigned hex value via the two's complement notation, with the minimum being -2147483648 or 0x800000000; numbers less than the minimum are converted to two's complement with greater precision and also have the most significant digits discarded.

The return value is interpreted as a two's complement number with sign, so the return value will be an integer in the range -2147483648 to 2147483647.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

In the following example, the integer 1 is shifted 10 bits to the left:

```
x = 1 << 10
```

The result of this operation is `x = 1024`. This is because 1 decimal equals 1 binary, 1 binary shifted left by 10 is 10000000000 binary, and 10000000000 binary is 1024 decimal.

In the following example, the integer 7 is shifted 8 bits to the left:

```
x = 7 << 8
```

The result of this operation is `x = 1792`. This is because 7 decimal equals 111 binary, 111 binary shifted left by 8 bits is 11100000000 binary, and 11100000000 binary is 1792 decimal.

If you trace the following example, you see that the bits have been pushed two spaces to the left:

```
// 2 binary == 0010
// 8 binary == 1000
trace(2 << 2);  // output: 8
```

See also

>>= (bitwise right shift and assignment), >> (bitwise right shift), <<= (bitwise left shift and assignment), >>> (bitwise unsigned right shift), >>>= (bitwise unsigned right shift and assignment)

<<= (bitwise left shift and assignment)

Availability

Flash Player 5.

Usage

`expression1 <<= expression2`

Parameters

`expression1`	A number or expression to be shifted left.
`expression2`	A number or expression that converts to an integer from 0 to 31.

Returns

A 32-bit integer.

Description

Operator (bitwise compound assignment); this operator performs a bitwise left shift (<<=) operation and stores the contents as a result in *`expression1`*. The following two expressions are equivalent:

```
A <<= B
A = (A << B)
```

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

In the following example, you use the bitwise left shift and assignment (<<=) operator to shift all bits one space to the left:

```
var x:Number = 4;
// shift all bits one slot to the left.
x <<= 1;
trace(x);  // output: 8
// 4 decimal = 0100 binary
// 8 decimal = 1000 binary
```

See also

<< (bitwise left shift), >>= (bitwise right shift and assignment), >> (bitwise right shift)

<= (less than or equal to)

Availability

Flash Player 4.

In Flash 5 or later, the less than or equal to (`<=`) operator is a comparison operator capable of handling various data types. In Flash 4, `<=` is a numeric operator. Flash 4 files that are brought into the Flash 5 or later authoring environment undergo a conversion process to maintain data type integrity. The following illustrates the conversion of a Flash 4 file containing a numeric quality comparison.

Flash 4 file:

```
x <= y
```

Converted Flash 5 or later file:

```
Number(x) <= Number(y)
```

Usage

```
expression1 <= expression2
```

Parameters

`expression1,expression2` A number or string.

Returns

A Boolean value.

Description

Operator (comparison); compares two expressions and determines whether *`expression1`* is less than or equal to *`expression2`*; if it is, the operator returns `true`. If *`expression1`* is greater than *`expression2`*, the operator returns `false`. String expressions are evaluated using alphabetical order; all capital letters come before lowercase letters.

For more information, see “Operator precedence and associativity” in *Using ActionScript in Flash*.

Example

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

```
trace(5<=10); // true
trace(2<=2); // true
trace(10<=3); // false
trace("Allen"<="Jack"); // true
trace("Jack"<="Allen"); // false
trace("11"<="3"); // true
trace("11"<=3); // false (numeric comparison)
trace("C"<="abc"); // true
trace("A"<=a); // true
```

= (assignment)

Availability

Flash Player 4.

In Flash 5 or later, = is an assignment operator, and the == operator is used to evaluate equality. In Flash 4, = is a numeric equality operator. Flash 4 files that are brought into the Flash 5 or later authoring environment undergo a conversion process to maintain data type integrity.

Flash 4 file:

```
x = y
```

Converted Flash 5 or later file:

```
Number(x) == Number(y)
```

Usage

`expression1 = expression2`

Parameters

`expression1`	A variable, element of an array, or property of an object.
`expression2`	A value of any type.

Returns

The assigned value, *`expression2`*.

Description

Operator; assigns the value of *`expression2`* (the parameter on the right) to the variable, array element, or property in *`expression1`*. Assignment can be either by value or by reference. Assignment by value copies the actual value of *`expression2`* and stores it in *`expression1`*. Assignment by value is used when a variable is assigned a number or string literal. Assignment by reference stores a reference to *`expression2`* in *`expression1`*. Assignment by reference is commonly used with the `new` operator. Use of the `new` operator creates an object in memory and a reference to that location in memory is assigned to a variable.

For more information, see “Operator precedence and associativity” in Using ActionScript in Flash.

Example

The following example uses assignment by value to assign the value of 5 to the variable `x`.

```
var x:Number = 5;
```

The following example uses assignment by value to assign the value `"hello"` to the variable `x`:

```
var x:String;
x = "hello";
```

The following example uses assignment by reference to create the `moonsOfJupiter` variable, which contains a reference to a newly created Array object. Assignment by value is then used to copy the value `"Callisto"` to the first element of the array referenced by the variable `moonsOfJupiter`:

```
var moonsOfJupiter:Array = new Array();
moonsOfJupiter[0] = "Callisto";
```

The following example uses assignment by reference to create a new object, and assign a reference to that object to the variable `neptune`. Assignment by value is then used to assign the value of 49528 to the size property of the myObject object:

```
var mercury:Object = new Object();
mercury.diameter = 3030; // in miles
trace (mercury.diameter); // output: 3030
```

The following example builds upon the previous example by creating a variable named `merkur` (the German word for mercury) and assigning it the value of `mercury`. This creates two variables that reference the same object in memory, which means you can use either variable to access the object's properties. We can then change the diameter property to use kilometers instead of miles:

```
var merkur:Object = mercury;
merkur.diameter = 4878; // in kilometers
trace (mercury.diameter); // output: 4878
```

See also

== (equality)

-= (subtraction assignment)

Availability

Flash Player 4.

Usage

expression1 -= *expression2*

Parameters

expression1,expression2	A number or expression that evaluates to a number.

Returns

A number.

Description

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

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

String expressions must be converted to numbers; otherwise, `NaN` (not a number) is returned.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The following example uses the subtraction assignment (`-=`) operator to subtract 10 from 5 and assign the result to the variable `x`:

```
var x:Number = 5;
var y:Number = 10;
x -= y;
trace(x);  // output: -5
```

The following example shows how strings are converted to numbers:

```
var x:String = "5";
var y:String = "10";
x -= y;
trace(x);  // output: -5
```

See also

– (minus)

== (equality)

Availability

Flash Player 5.

Usage

expression1 == *expression2*

Parameters

expression1,*expression2* A number, string, Boolean value, variable, object, array, or function.

Returns

A Boolean value.

Description

Operator (equality); tests two expressions for equality. The result is `true` if the expressions are equal.

The definition of *equal* depends on the data type of the parameter:

- Numbers and Boolean values are compared by value and are considered equal if they have the same value.
- String expressions are equal if they have the same number of characters and the characters are identical.
- Variables representing objects, arrays, and functions are compared by reference. Two such variables are equal if they refer to the same object, array, or function. Two separate arrays are never considered equal, even if they have the same number of elements.

When comparing by value, if *expression1* and *expression2* are different data types, ActionScript will attempt to convert the data type of *expression2* to match that of *expression1*. For more information, see “Automatic data typing” and “Operator precedence and associativity” in *Using ActionScript in Flash*.

Example

The following example uses the equality (`==`) operator with an `if` statement:

```
var a:String = "David", b:String = "David";
if (a == b) {
  trace("David is David");
}
```

The following examples show the results of operations that compare mixed types:

```
var x:Number = 5;
var y:String = "5";
trace(x == y);  // output: true
var x:String = "5";
var y:String = "66";
trace(x == y);  // output: false
var x:String = "chris";
var y:String = "steve";
trace(x == y);  // output: false
```

The following examples show comparison by reference. The first example compares two arrays with identical length and elements. The equality operator will return `false` for these two arrays. Although the arrays appear equal, comparison by reference requires that they both refer to the same array. The second example creates the `thirdArray` variable, which points to the same array as the variable `firstArray`. The equality operator will return `true` for these two arrays because the two variables refer to the same array.

```
var firstArray:Array = new Array("one", "two", "three");
var secondArray:Array = new Array("one", "two", "three");
trace(firstArray == secondArray);  // will output false

// Arrays are only considered equal
// if the variables refer to the same array.
var thirdArray:Array = firstArray;
trace(firstArray == thirdArray);  // will output true
```

See also

! (logical NOT), != (inequality), !== (strict inequality), && (logical AND), || (logical OR), === (strict equality)

=== (strict equality)

Availability

Flash Player 6.

Usage

```
expression1 === expression2
```

Returns

A Boolean value.

Description

Operator; tests two expressions for equality; the strict equality (===)operator performs in the same way as the equality (==) operator, except that data types are not converted. (For more information, see "Automatic data typing" in *Using ActionScript in Flash*.) The result is `true` if both expressions, including their data types, are equal.

The definition of *equal* depends on the data type of the parameter:

- Numbers and Boolean values are compared by value and are considered equal if they have the same value.
- String expressions are equal if they have the same number of characters and the characters are identical.
- Variables representing objects, arrays, and functions are compared by reference. Two such variables are equal if they refer to the same object, array, or function. Two separate arrays are never considered equal, even if they have the same number of elements.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The comments in the following code show the returned value of operations that use the equality and strict equality operators:

```
// Both return true because no conversion is done
var string1:String = "5";
var string2:String = "5";
trace(string1 == string2); // true
trace(string1 === string2); // true

// Automatic data typing in this example converts 5 to "5"
var string1:String = "5";
var num:Number = 5;
trace(string1 == num);  // true
trace(string1 === num);  // false

// Automatic data typing in this example converts true to "1"
var string1:String = "1";
var bool1:Boolean = true;
trace(string1 == bool1);  // true
trace(string1 === bool1);  // false
```

```
// Automatic data typing in this example converts false to “0”
var string1:String = "0";
var bool2:Boolean = false;
trace(string1 == bool2);  // true
trace(string1 === bool2);  // false
```

The following examples show how strict equality treats variables that are references differently than it treats variables that contain literal values. This is one reason to consistently use String literals and to avoid the use of the `new` operator with the String class.

```
// Create a string variable using a literal value
var str:String = "asdf”;

// Create a variable that is a reference
var stringRef:String = new String("asdf");

// The equality operator does not distinguish among literals, variables,
// and references
trace(stringRef == "asdf"); // true
trace(stringRef == str); // true
trace("asdf" == str); // true

// The strict equality operator considers variables that are references
// distinct from literals and variables
trace(stringRef === "asdf"); // false
trace(stringRef === str); // false
```

See also

! (logical NOT), != (inequality), !== (strict inequality), && (logical AND), || (logical OR), == (equality), === (strict equality)

> (greater than)

Availability

Flash Player 4.

In Flash 5 or later, the greater-than (`>`) operator is a comparison operator capable of handling various data types. In Flash 4, `>` is a numeric operator. Flash 4 files that are brought into the Flash 5 or later authoring environment undergo a conversion process to maintain data type integrity. The following illustrates the conversion of a Flash 4 file containing a numeric quality comparison.

Flash 4 file:

```
x > y
```

Converted Flash 5 or later file:

```
Number(x) > Number(y)
```

Usage

`expression1 >expression2`

Parameters

`expression1,expression2` A number or string.

Returns

A Boolean value.

Description

Operator (comparison); compares two expressions and determines whether *`expression1`* is greater than *`expression2`*; if it is, the operator returns `true`. If *`expression1`* is less than or equal to *`expression2`*, the operator returns `false`. String expressions are evaluated using alphabetical order; all capital letters come before lowercase letters.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

In the following example, the greater than (>) operator is used to determine whether the value of the text field `score_txt` is greater than 90:

```
if (score_txt.text>90) {
  trace("Congratulations, you win!");
} else {
  trace("sorry, try again");
}
```

>= (greater than or equal to)

Availability

Flash Player 4.

In Flash 5 or later, the greater than or equal to (>=) operator is a comparison operator capable of handling various data types. In Flash 4, >= is a numeric operator. Flash 4 files that are brought into the Flash 5 or later authoring environment undergo a conversion process to maintain data type integrity. The following illustrates the conversion of a Flash 4 file containing a numeric quality comparison.

Flash 4 file:

```
x > y
```

Converted Flash 5 or later file:

```
Number(x) > Number(y)
```

Usage

```
expression1 >= expression2
```

Parameters

`expression1, expression2` A string, integer, or floating-point number.

Returns

A Boolean value.

Description

Operator (comparison); compares two expressions and determines whether *`expression1`* is greater than or equal to *`expression2`* (`true`) or *`expression1`* is less than *`expression2`* (`false`).

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

In the following example, the greater than or equal to (>=) operator is used to determine whether the current hour is greater than or equal to 12:

```
if (new Date().getHours()>=12) {
  trace("good afternoon");
} else {
  trace("good morning");
}
```

>> (bitwise right shift)

Availability

Flash Player 5.

Usage

`expression1 >> expression2`

Parameters

`expression1`	A number or expression to be shifted right.
`expression2`	A number or expression that converts to an integer from 0 to 31.

Returns

A 32-bit integer.

Description

Operator (bitwise); converts *`expression1`* and *`expression2`* to 32-bit integers, and shifts all the bits in *`expression1`* to the right by the number of places specified by the integer that results from the conversion of *`expression2`*. Bits that are shifted off the right end are discarded. To preserve the sign of the original *`expression`*, the bits on the left are filled in with 0 if the most significant bit (the bit farthest to the left) of *`expression1`* is 0, and filled in with 1 if the most significant bit is 1. Shifting a value right by one position is the equivalent of dividing by 2 and discarding the remainder.

Floating-point numbers are converted to integers by discarding any digits after the decimal point. Positive integers are converted to an unsigned hex value with a maximum value of 4294967295 or 0xFFFFFFFF; values larger than the maximum have their most significant digits discarded when they are converted so the value is still 32-bit. Negative numbers are converted to an unsigned hex value via the two's complement notation, with the minimum being -2147483648 or 0x800000000; numbers less than the minimum are converted to two's complement with greater precision and also have the most significant digits discarded.

The return value is interpreted as a two's complement number with sign, so the return value will be an integer in the range -2147483648 to 2147483647.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The following example converts 65535 to a 32-bit integer and shifts it 8 bits to the right:

```
var x:Number = 65535 >> 8;
trace(x); // outputs 255
```

The following example shows the result of the previous example:

```
var x:Number = 255
```

This is because 65535 decimal equals 1111111111111111 binary (sixteen 1's), 1111111111111111 binary shifted right by 8 bits is 11111111 binary, and 11111111 binary is 255 decimal. The most significant bit is 0 because the integers are 32-bit, so the fill bit is 0.

The following example converts -1 to a 32-bit integer and shifts it 1 bit to the right:

```
var x:Number = -1 >> 1;
trace(x); // outputs -1
```

The following example shows the result of the previous example:

```
var x:Number = -1
```

This is because -1 decimal equals 11111111111111111111111111111111 binary (thirty-two 1's), shifting right by one bit causes the least significant (bit farthest to the right) to be discarded and the most significant bit to be filled in with 1. The result is 11111111111111111111111111111111 (thirty-two 1's) binary, which represents the 32-bit integer -1.

See also

>>= (bitwise right shift and assignment)

>>= (bitwise right shift and assignment)

Availability

Flash Player 5.

Usage

`expression1 >>= expression2`

Parameters

`expression1`	A number or expression to be shifted right.
`expression2`	A number or expression that converts to an integer from 0 to 31.

Returns

A 32-bit integer.

Description

Operator (bitwise compound assignment); this operator performs a bitwise right-shift operation and stores the contents as a result in *`expression1`*.

The following two expressions are equivalent:

```
A >>= B
```

```
A = (A >> B)
```

For more information, see “Operator precedence and associativity” in *Using ActionScript in Flash*.

Example

The following commented code uses the bitwise right shift and assignment (`>>=`) operator.

```
function convertToBinary(numberToConvert:Number):String {
  var result:String = "";
  for (var i = 0; i<32; i++) {
    // Extract least significant bit using bitwise AND
    var lsb:Number = numberToConvert & 1;
    // Add this bit to the result string
    result = (lsb ? "1" : "0")+result;
    // Shift numberToConvert right by one bit, to see next bit
    numberToConvert >>= 1;
  }
  return result;
}
trace(convertToBinary(479));
// Returns the string 00000000000000000000000111011111
// This string is the binary representation of the decimal
// number 479
```

See also

>> (bitwise right shift)

>>> (bitwise unsigned right shift)

Availability

Flash Player 5.

Usage

expression1 >>> *expression2*

Parameters

expression1	A number or expression to be shifted right.
expression2	A number or expression that converts to an integer between 0 and 31.

Returns

A 32-bit unsigned integer.

Description

Operator (bitwise); the same as the bitwise right shift (>>) operator except that it does not preserve the sign of the original *expression* because the bits on the left are always filled with 0.

Floating-point numbers are converted to integers by discarding any digits after the decimal point. Positive integers are converted to an unsigned hex value with a maximum value of 4294967295 or 0xFFFFFFFF; values larger than the maximum have their most significant digits discarded when they are converted so the value is still 32-bit. Negative numbers are converted to an unsigned hex value via the two's complement notation, with the minimum being -2147483648 or 0x800000000; numbers less than the minimum are converted to two's complement with greater precision and also have the most significant digits discarded.

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

Example

The following example converts -1 to a 32-bit integer and shifts it 1 bit to the right:

```
var x:Number = -1 >>> 1;
trace(x);  // output: 2147483647
```

This is because -1 decimal is 11111111111111111111111111111111 binary (thirty-two 1's), and when you shift right (unsigned) by 1 bit, the least significant (rightmost) bit is discarded, and the most significant (leftmost) bit is filled with a 0. The result is 01111111111111111111111111111111 binary, which represents the 32-bit integer 2147483647.

See also

>>= (bitwise right shift and assignment)

>>>= (bitwise unsigned right shift and assignment)

Availability

Flash Player 5.

Usage

```
expression1 >>>= expression2
```

Parameters

`expression1`	A number or expression to be shifted left.
`expression2`	A number or expression that converts to an integer from 0 to 31.

Returns

A 32-bit integer.

Description

Operator (bitwise compound assignment); performs an unsigned bitwise right-shift operation and stores the contents as a result in `expression1`. The following two expressions are equivalent:

```
A >>>= B
A = (A >>> B)
```

For more information, see "Operator precedence and associativity" in *Using ActionScript in Flash*.

See also

>>> (bitwise unsigned right shift), >>= (bitwise right shift and assignment)

Accessibility class

Availability

Flash Player 6.

Description

The Accessibility class manages communication with screen readers. Screen readers are a type of assistive technology for visually impaired users that provides an audio version of screen content. The methods of the Accessibility class are static—that is, you don't have to create an instance of the class to use its methods.

To get and set accessible properties for a specific object, such as a button, movie clip, or text field, use the `_accProps` property. To determine whether the player is running in an environment that supports accessibility aids, use `System.capabilities.hasAccessibility`. For information on creating accessible content, see "Accessibility overview" in *Using Flash*.

Method summary for the Accessibility class

Method	Description
`Accessibility.isActive()`	Indicates whether a screen reader program is active.
`Accessibility.updateProperties()`	Updates the description of objects on the screen for screen readers.

Accessibility.isActive()

Availability

Flash Player 6.

Usage

`Accessibility.isActive() : Boolean`

Parameters

None.

Returns

A Boolean value: `true` if the Flash Player is communicating with an accessibility aid (usually a screen reader); `false` otherwise.

Description

Method; indicates whether an accessibility aid is currently active and the player is communicating with it. Use this method when you want your application to behave differently in the presence of a screen reader or other accessibility aid.

Note: If you call this method within one or two seconds of the first appearance of the Flash window in which your document is playing, you might get a return value of `false` even if there is an active Microsoft Active Accessibility (MSAA) client. This is because of an asynchronous communication mechanism between Flash and MSAA clients. You can work around this limitation by ensuring a delay of one to two seconds after loading your document before calling this method.

Example

The following example checks whether an accessibility aid is currently active:

```
if (Accessibility.isActive()) {
  trace ("An accessibility aid is currently active");
} else {
  trace ("There is currently no active accessibility aid");
}
```

See also

Accessibility.updateProperties(), `_accProps`, `System.capabilities.hasAccessibility`

Accessibility.updateProperties()

Availability

Flash Player 6 (6.0.65).

Usage

```
Accessibility.updateProperties() : Void
```

Parameters

None.

Returns

Nothing.

Description

Method; causes all changes to `_accProps` (accessibility properties) objects to take effect. For information on setting accessibility properties, see `_accProps`.

If you modify the accessibility properties for multiple objects, only one call to `Accessibility.updateProperties()` is necessary; multiple calls can result in reduced performance and unintelligible screen reader results.

Example

If you change an image and want to update its accessible description, you could use the following ActionScript code:

```
my_mc.gotoAndStop(2);

if (my_mc._accProps == undefined ) {
  my_mc._accProps = new Object();
}
my_mc._accProps.name = "Photo of Mount Rushmore";
Accessibility.updateProperties();
```

See also

Accessibility.isActive(), _accProps, `System.capabilities.hasAccessibility`

_accProps

Availability

Flash Player 6.65.

Usage

```
_accProps.propertyName
instanceName._accProps.propertyName
```

Parameters

propertyName An accessibility property name (see the following description for valid names).

instanceName The instance name assigned to an instance of a movie clip, button, dynamic text field, or input text field. To refer to the `_accProps` object that represents the entire Flash document, omit *instanceName*.

Description

Property; lets you control screen reader accessibility options for SWF files, movie clips, buttons, dynamic text fields, and input text fields at runtime. These properties override the corresponding settings available in the Accessibility panel during authoring. For changes to these properties to take effect, you must call `Accessibility.updateProperties()`. For information on the Accessibility panel, see "The Flash Accessibility panel" in *Using Flash*.

To determine whether the player is running in an environment that supports accessibility aids, use `System.capabilities.hasAccessibility`.

The following table lists the name and data type of each `_accProps` property, its equivalent setting in the Accessibility panel, and the kinds of objects to which the property can be applied. The term *inverse logic* means that the property setting is the inverse of the corresponding setting in the Accessibility panel. For example, setting the `silent` property to `true` is equivalent to deselecting the Make Movie Accessible or Make Object Accessible option.

Property	Data type	Equivalent in Accessibility panel	Applies to
`silent`	Boolean	Make Movie Accessible/ Make Object Accessible *(inverse logic)*	Whole SWF files Movie clips Buttons Dynamic text Input text
`forceSimple`	Boolean	Make Child Objects Accessible *(inverse logic)*	Whole SWF files Movie clips
`name`	String	Name	Whole SWF files Movie clips Buttons Input text

Property	Data type	Equivalent in Accessibility panel	Applies to
`description`	String	Description	Whole SWF files Movie clips Buttons Dynamic text Input text
`shortcut`	String	Shortcut*	Movie clips Buttons Input text

* For the Shortcut field, use names of the form Control+A. Adding a keyboard shortcut to the Accessibility panel doesn't create a keyboard shortcut; it merely advises screen readers of an existing shortcut. For information on assigning a keyboard shortcut to an accessible object, see `Key.addListener()`.

To specify settings that correspond to the Tab index setting in the Accessibility panel, use the `Button.tabIndex`, `MovieClip.tabIndex`, or `TextField.tabIndex` property.

There is no way to specify an Auto Label setting at runtime.

To refer to the `_accProps` object that represents the entire Flash document, omit the `instanceName` parameter. The value of `_accProps` must be an object. This means that if no `_accProps` object already exists, you must create one, as shown in the following example, before you can assign values to the properties of the `_accProps` object:

```
if ( _accProps == undefined ) {
  _accProps = new Object();
}
_accProps.name = "My SWF file";
```

When `_accProps` is used without the *`instanceName`* parameter, changes made to `_accProps` properties apply to the whole SWF file. For example, the following code sets the Accessibility `name` property for the whole SWF file to the string `"Pet Store"` and then calls `Accessibility.updateProperties()` to cause that change:

```
_accProps.name = "Pet Store";
Accessibility.updateProperties();
```

In contrast, the following code sets the `name` property for a movie clip with the instance name `price_mc` to the string `"Price"`:

```
price_mc._accProps.name = "Price";
Accessibility.updateProperties();
```

If you are specifying several accessibility properties, make as many changes as you can before calling `Accessibility.updateProperties()`, instead of calling it after each property statement, as shown in the following example:

```
_accProps.name = "Pet Store";
animal_mc._accProps.name = "Animal";
animal_mc._accProps.description = "Cat, dog, fish, etc.";
price_mc._accProps.name = "Price";
price_mc._accProps.description = "Cost of a single item";
Accessibility.updateProperties();
```

If you don't specify an accessibility property for a document or an object, any values set in the Accessibility panel are implemented.

After you specify an accessibility property, you can't revert its value to a value set in the Accessibility panel. However, you can set the property to its default value (`false` for Boolean values; empty strings for string values) by deleting the property from the `_accProps` object, as shown in the following example:

```
my_mc._accProps.silent = true; // set a property
// other code here
delete my_mc._accProps.silent; // revert to default value
```

The value of _accProps must be an object. This means that if no `_accProps` object already exists, you must create one before you can assign clues to the properties of the `_accProps` object.

```
if (_accProps == undefined) {
  _accProps = new Object();
}

_accProps.name = "My movie";
```

Example

If you change an image and want to update its accessibility description, you can use the following ActionScript code:

```
my_mc.gotoAndStop(2);

if (my_mc._accProps == undefined ) {
  my_mc._accProps = new Object();
}
my_mc._accProps.name = "Photo of Mount Rushmore";
Accessibility.updateProperties();
```

See also

`Accessibility.isActive()`, `Accessibility.updateProperties()`, `System.capabilities.hasAccessibility`

arguments object

Availability

Flash Player 5; property added in Flash Player 6.

Description

The arguments object is an array that contains the values that were passed as parameters to any function. Each time a function is called in ActionScript, an arguments object is automatically created for that function. A local variable, `arguments`, is also created and lets you refer to the arguments object.

Property summary for the arguments object

Property	Description
`arguments.callee`	A reference to the function being called.
`arguments.caller`	A reference to the calling function.
`arguments.length`	The number of parameters passed to a function.

arguments.callee

Availability

Flash Player 5.

Usage

```
arguments.callee:Function
```

Description

Property; refers to the function that is currently being called.

Example

You can use the `arguments.callee` property to make an anonymous function that is recursive, as shown in the following example:

```
factorial = function (x:Number) {
  if (x<=1) {
    return 1;
  } else {
    return x*arguments.callee(x-1);
  }
};
trace(factorial(3));  // output: 6
```

The following example is a named recursive function:

```
function factorial(x:Number):Number {
  if (x<=1) {
    return 1;
  } else {
    return x*factorial(x-1);
  }
}
trace(factorial(4));  // output: 24
```

arguments.caller

Availability

Flash Player 6.

Usage

`arguments.caller:`*`Function`*

Description

Property; refers to the calling function. The value of this property is `null` if the current function was not called by another function.

Example

The following example defines two functions—a caller function, named function1, which calls a another function, named function2:

```
// define the caller function, named function1
var function1:Function = function () {
  function2("hello from function1");
}

// define the callee function, named function2
var function2:Function = function (aString:String) {
  if (arguments.caller == function1) {
    trace("function2 was called by function1");
    trace(aString);
  }
}

// call function1
function1();

/*
Output:
function2 was called by function1
hello from function1
*/
```

arguments.length

Availability

Flash Player 5.

Usage

```
arguments.length:Number
```

Description

Property; the number of parameters actually passed to a function.

Example

The following ActionScript includes a function called `getArgLength`, which returns the number of arguments that are passed into the function. Although the method expects three arguments, you can pass as many arguments as you want.

```
function getArgLength(param_arg1:String, param_arg2:String, param_arg3:String)
  {
    return (arguments.length);
  }
trace(getArgLength("one", "two", "three")); // output: 3
trace(getArgLength("one", "two")); // output: 2
trace(getArgLength("one", "two", "three", "four")); // output: 4
```

In the following example, the function called `listSum` adds the values passed to it and returns the sum. The function uses a `for` loop to examine each argument passed. If the value is a number, the value is added to the `sum` variable. At the end of the function, the value of `sum` is returned.

```
function listSum():Number {
  var sum:Number = 0;
  for (var i = 0; i<arguments.length; i++) {
    if (!isNaN(Number(arguments[i]))) {
      sum += parseFloat(arguments[i]);
    }
  }
  return sum;
}
trace(listSum(100, 200, 300, 7)); // output: 607
```

Array()

Availability

Flash Player 6.

Usage

```
Array() : Array
```

```
Array(numElements:Number) : Array
```

```
Array( [element0:Object [, element1 , element2,...elementN ] ]) : Array
```

Parameters

element	One or more elements to place in the array.

Returns

An array.

Description

Conversion function; creates a new, empty array or converts specified elements to an array. Using this function is similar to creating an array with the Array constructor (see "Constructor for the Array class" on page 104).

Example

Usage 1: The following example adds items to an array by using the array's index and then by using the Array class's `push` method:

```
var myArray:Array = Array();
myArray.push(12);
trace(myArray); //traces 12
myArray[4] = 7;
trace(myArray); //traces 12,undefined,undefined,undefined,7
```

Usage 2: The following example creates an array of length 4 but with no elements defined:

```
var myArray:Array = Array(4);
trace(myArray.length);  // traces 4
trace(myArray); // traces undefined,undefined,undefined,undefined
```

Usage 3: The following example creates an array with three defined elements:

```
var myArray:Array = Array(["firstElement", "secondElement", "thirdElement"]);
trace (myArray); // traces firstElement,secondElement,thirdElement
```

Note: Unlike the Array class constructor, the `Array()` function does not use the keyword `new`.

See also

Array class

Array class

Availability

Flash Player 5 (became a native object in Flash Player 6, which improved performance significantly).

Description

The Array class lets you access and manipulate arrays. An array is an object whose properties are identified by a number representing their position in the array. This number is referred to as the *index*. All arrays are zero-based, which means that the first element in the array is [0], the second element is [1], and so on. To create an Array object, you use the constructor `new Array()`. To access the elements of an array, you use the array access (`[]`) operator.

In the following example, `my_array` contains four months of the year:

```
var my_array:Array = new Array();
my_array[0] = "January";
my_array[1] = "February";
my_array[2] = "March";
my_array[3] = "April";
```

Method summary for the Array class

Method	Description
`Array.concat()`	Concatenates the parameters and returns them as a new array.
`Array.join()`	Joins all elements of an array into a string.
`Array.pop()`	Removes the last element of an array and returns its value.
`Array.push()`	Adds one or more elements to the end of an array and returns the array's new length.
`Array.reverse()`	Reverses the direction of an array.
`Array.shift()`	Removes the first element from an array and returns its value.
`Array.slice()`	Extracts a section of an array and returns it as a new array.
`Array.sort()`	Sorts an array in place.
`Array.sortOn()`	Sorts an array according to a field in the array.
`Array.splice()`	Adds and removes elements from an array.
`Array.toString()`	Returns a string value representing the elements in the Array object.
`Array.unshift()`	Adds one or more elements to the beginning of an array and returns the array's new length.

Property summary for the Array class

Property	Description
`Array.length`	A non-negative integer specifying the number of elements in the array.

Constructor for the Array class

Availability

Flash Player 5.

Usage

```
new Array() : Array
new Array(length:Number) : Array
new Array(element0, element1, element2,...elementN) : Array
```

Parameters

length An integer that specifies the number of elements in the array.

element0...elementN A list of two or more arbitrary values. The values can be of type Boolean, Number, String, Object, or Array. The first element in an array always has an index or position of 0.

Note: If only a single numeric parameter is passed to the Array constructor, it is assumed to be `length` and it is converted to an integer using the `Integer()` function.

Returns

A reference to an Array object.

Description

Constructor; lets you create an array. You can use the constructor to create different types of arrays: an empty array, an array with a specific length but whose elements have undefined values, or an array whose elements have specific values.

Usage 1: If you don't specify any parameters, an array with a length of 0 is created.

Usage 2: If you specify only a length, an array is created with *length* number of elements. Each element's value is set to `undefined`.

Usage 3: If you use the *element* parameters to specify values, an array is created with specific values.

Example

Usage 1: The following example creates a new Array object with an initial length of 0:

```
var my_array:Array = new Array();
trace(my_array.length); // traces 0
```

Usage 2: The following example creates a new Array object with an initial length of 4:

```
var my_array:Array = new Array(4);
trace(my_array.length); // returns 4
trace(my_array[0]); // returns undefined
if (my_array[0] == undefined) { // no quotation marks around undefined
  trace("undefined is a special value, not a string");
} // traces: undefined is a special value, not a string
```

Usage 3: The following example creates the new Array object `go_gos_array` with an initial length of 5:

```
var go_gos_array:Array = new Array("Belinda", "Gina", "Kathy", "Charlotte",
  "Jane");
trace(go_gos_array.length); // returns 5
trace(go_gos_array.join(", ")); // displays elements
```

The initial elements of the `go_gos_array` array are identified, as shown in the following example:

```
go_gos_array[0] = "Belinda";
go_gos_array[1] = "Gina";
go_gos_array[2] = "Kathy";
go_gos_array[3] = "Charlotte";
go_gos_array[4] = "Jane";
```

The following code adds a sixth element to the `go_gos_array` array and changes the second element:

```
go_gos_array[5] = "Donna";
go_gos_array[1] = "Nina"
trace(go_gos_array.join(" + "));
// returns Belinda + Nina + Kathy + Charlotte + Jane + Donna
```

See also

Array.length, [] (array access)

Array.concat()

Availability

Flash Player 5.

Usage

my_array.concat([*value0:Object, value1,...valueN*]) *: Array*

Parameters

value0,...valueN Numbers, elements, or strings to be concatenated in a new array. If you don't pass any values, a duplicate of *my_array* is created.

Returns

An array.

Description

Method; concatenates the elements specified in the parameters with the elements in *my_array*, and creates a new array. If the *value* parameters specify an array, the elements of that array are concatenated, rather than the array itself. The array *my_array* is left unchanged.

Example

The following code concatenates two arrays:

```
var alpha_array:Array = new Array("a","b","c");
var numeric_array:Array = new Array(1,2,3);
var alphaNumeric_array:Array =alpha_array.concat(numeric_array);
trace(alphaNumeric_array);
// creates array [a,b,c,1,2,3]
```

The following code concatenates three arrays:

```
var num1_array:Array = [1,3,5];
var num2_array:Array = [2,4,6];
var num3_array:Array = [7,8,9];
var nums_array:Array=num1_array.concat(num2_array,num3_array)
trace(nums_array);
// creates array [1,3,5,2,4,6,7,8,9]
```

Nested arrays are not flattened in the same way as normal arrays. The elements in a nested array are not broken into separate elements in array `x_array`, as shown in the following example:

```
var a_array:Array = new Array ("a","b","c");

// 2 and 3 are elements in a nested array
var n_array:Array = new Array(1, [2, 3], 4);

var x_array:Array = a_array.concat(n_array);
trace(x_array[0]); // a
trace(x_array[1]); // b
trace(x_array[2]); // c
trace(x_array[3]); // 1
trace(x_array[4]); // 2, 3
trace(x_array[5]); // 4
```

Array.join()

Availability

Flash Player 5.

Usage

`my_array.join([separator:String]) : String`

Parameters

`separator` A character or string that separates array elements in the returned string. If you omit this parameter, a comma (,) is used as the default separator.

Returns

A string.

Description

Method; converts the elements in an array to strings, inserts the specified separator between the elements, concatenates them, and returns the resulting string. A nested array is always separated by a comma (,), not by the separator passed to the `join()` method.

Example

The following example creates an array with three elements: Earth, Moon, and Sun. It then joins the array three times—first using the default separator (a comma [,] and a space), then using a dash (-), and then using a plus sign (+).

```
var a_array:Array = new Array("Earth","Moon","Sun")
trace(a_array.join());
// displays Earth,Moon,Sun
trace(a_array.join(" - "));
// displays Earth - Moon - Sun
trace(a_array.join(" + "));
// displays Earth + Moon + Sun
```

The following example creates a nested array containing two arrays. The first array has three elements: Europa, Io, and Callisto. The second array has two elements: Titan and Rhea. It joins the array using a plus sign (+), but the elements within each nested array remain separated by a comma (,).

```
var a_nested_array:Array = new Array(["Europa", "Io", "Callisto"], ["Titan",
  "Rhea"]);
trace(a_nested_array.join(" + "));
// returns Europa,Io,Callisto + Titan,Rhea
```

See Also

`String.split()`

Array.length

Availability

Flash Player 5.

Usage

`my_array.length:Number`

Description

Property; a non-negative integer specifying the number of elements in the array. This property is automatically updated when new elements are added to the array. When you assign a value to an array element (for example, `my_array[index] = value`), if `index` is a number, and `index+1` is greater than the `length` property, the `length` property is updated to `index+1`.

Note: If you assign a value to the `length` property that is shorter than the existing length, the array will be truncated.

Example

The following code explains how the `length` property is updated:

```
var my_array:Array = new Array();
trace(my_array.length); // initial length is 0
my_array[0] = 'a';
trace(my_array.length); // my_array.length is updated to 1
my_array[1] = 'b';
trace(my_array.length); // my_array.length is updated to 2
my_array[9] = 'c';
trace(my_array.length); // my_array.length is updated to 10
trace(my_array);
// displays:
// a,b,undefined,undefined,undefined,undefined,undefined,undefined,undefined,c

// if the length property is now set to 5, the array will be truncated
my_array.length = 5;
trace(my_array.length); // my_array.length is updated to 5
trace(my_array); // outputs: a,b,undefined,undefined,undefined
```

Array.pop()

Availability

Flash Player 5.

Usage

```
my_array.pop() : Object
```

Parameters

None.

Returns

The value of the last element in the specified array.

Description

Method; removes the last element from an array and returns the value of that element.

Example

The following code creates the `myPets_array` array containing four elements, and then removes its last element:

```
var myPets_array:Array = new Array("cat", "dog", "bird", "fish");
var popped:String = myPets_array.pop();
trace(popped); // displays fish
trace(myPets_array); // displays cat,dog,bird
```

See Also

Array.push(), Array.shift(), Array.unshift()

Array.push()

Availability

Flash Player 5.

Usage

my_array.push(*value,...*) : *Number*

Parameters

value One or more values to append to the array.

Returns

An integer representing the length of the new array.

Description

Method; adds one or more elements to the end of an array and returns the array's new length.

Example

The following example creates the array myPets_array with two elements, cat and dog. The second line adds two elements to the array. Because the push() method returns the new length of the array, the trace() statement in the last line sends the new length of myPets_array (4) to the Output panel.

```
var myPets_array:Array = new Array("cat", "dog");
var pushed:Number = myPets_array.push("bird", "fish");
trace(pushed); // displays 4
```

See Also

Array.pop(), Array.shift(), Array.unshift()

Array.reverse()

Availability

Flash Player 5.

Usage

```
my_array.reverse() : Void
```

Parameters

None.

Returns

Nothing.

Description

Method; reverses the array in place.

Example

The following example uses this method to reverse the array `numbers_array`:

```
var numbers_array:Array = new Array(1, 2, 3, 4, 5, 6);
trace(numbers_array); // displays 1,2,3,4,5,6
numbers_array.reverse();
trace(numbers_array); // displays 6,5,4,3,2,1
```

Array.shift()

Availability

Flash Player 5.

Usage

```
my_array.shift() : Object
```

Parameters

None.

Returns

The first element in an array.

Description

Method; removes the first element from an array and returns that element.

Example

The following code creates the array `myPets_array` and then removes the first element from the array and assigns it to the variable `shifted`:

```
var myPets_array:Array = new Array("cat", "dog", "bird", "fish");
var shifted:String = myPets_array.shift();
trace(shifted); // displays "cat"
trace(myPets_array); // displays dog,bird,fish
```

See also

Array.pop(), Array.push(), Array.unshift()

Array.slice()

Availability

Flash Player 5.

Usage

my_array.slice([*start:Number* [, *end:Number*]]) *: Array*

Parameters

start A number specifying the index of the starting point for the slice. If *start* is a negative number, the starting point begins at the end of the array, where -1 is the last element.

end A number specifying the index of the ending point for the slice. If you omit this parameter, the slice includes all elements from the starting point to the end of the array. If *end* is a negative number, the ending point is specified from the end of the array, where -1 is the last element.

Returns

An array.

Description

Method; returns a new array that consists of a range of elements from the original array, without modifying the original array. The returned array includes the *start* element and all elements up to, but not including, the *end* element.

If you don't pass any parameters, a duplicate of *my_array* is created.

Example

The following example creates an array of five pets and uses `slice()` to populate a new array comprising only four-legged pets:

```
var myPets_array:Array = new Array("cat", "dog", "fish", "canary", "parrot");
var myFourLeggedPets_array:Array = new Array();
var myFourLeggedPets_array = myPets_array.slice(0, 2);
trace(myFourLeggedPets_array);  // returns cat,dog
trace(myPets_array);  // returns cat,dog,fish,canary,parrot
```

The following example creates an array of five pets, and then uses `slice()` with a negative `start` parameter to copy the last two elements from the array:

```
var myPets_array:Array = new Array("cat", "dog", "fish", "canary", "parrot");
var myFlyingPets_array:Array = myPets_array.slice(-2);
trace(myFlyingPets_array); // traces canary,parrot
```

The following example creates an array of five pets and uses `slice()` with a negative `end` parameter to copy the middle element from the array:

```
var myPets_array:Array = new Array("cat", "dog", "fish", "canary", "parrot");
var myAquaticPets_array:Array = myPets_array.slice(2,-2);
trace(myAquaticPets_array); // returns fish
```

Array.sort()

Availability

Flash Player 5; additional capabilities added in Flash Player 7.

Usage

```
my_array.sort() : Array
my_array.sort(compareFunction:Function) : Array
my_array.sort(option:Number | option |... ) : Array
my_array.sort(compareFunction:Function, option:Number | option |... ) : Array
```

Parameters

`compareFunction` A comparison function used to determine the sorting order of elements in an array. Given the elements A and B, the result of `compareFunction` can have one of the following three values:

- -1, if A should appear before B in the sorted sequence
- 0, if A equals B
- 1, if A should appear after B in the sorted sequence

`option` One or more numbers or names of defined constants, separated by the `| (bitwise OR)` operator, that change the behavior of the sort from the default. The following values are acceptable for `option`:

- 1 or `Array.CASEINSENSITIVE`
- 2 or `Array.DESCENDING`
- 4 or `Array.UNIQUESORT`
- 8 or `Array.RETURNINDEXEDARRAY`
- 16 or `Array.NUMERIC`

For information on this parameter, see Array.sortOn().

Note: `Array.sort()` is defined in ECMA-262, but the array sorting options introduced in Flash Player 7 are Flash-specific extensions to the ECMA-262 specification.

Returns

The return value depends on whether you pass any parameters, as described in the following list:

- If you specify a value of 4 or `Array.UNIQUESORT` for `option` and two or more elements being sorted have identical sort fields, Flash returns a value of 0 and does not modify the array.
- If you specify a value of 8 or `Array.RETURNINDEXEDARRAY` for `option`, Flash returns an array that reflects the results of the sort and does not modify the array.
- Otherwise, Flash returns nothing and modifies the array to reflect the sort order.

Description

Method; sorts the elements in an array. Flash sorts according to Unicode values. (ASCII is a subset of Unicode.)

By default, `Array.sort()` works as described in the following list:

- Sorting is case-sensitive (*Z* precedes *a*).
- Sorting is ascending (*a* precedes *b*).
- The array is modified to reflect the sort order; multiple elements that have identical sort fields are placed consecutively in the sorted array in no particular order.
- Numeric fields are sorted as if they were strings, so 100 precedes 99, because “1” is a lower string value than “9”.

If you want to sort in another way, create a function to do the sorting and pass its name as the *`compareFunction`* parameter. You might do this, for example, if you want to sort alphabetically by last name, ascending, and then by ZIP code, descending.

If you want to specify one or more fields on which to sort, using either the default sort or the *`options`* parameter, use Array.sortOn().

Example

Usage 1: The following example shows the use of `Array.sort()` with and without a value passed for *`option`*:

```
var fruits_array:Array = new Array("oranges", "apples", "strawberries",
  "pineapples", "cherries");
trace(fruits_array);  // displays
  oranges,apples,strawberries,pineapples,cherries
fruits_array.sort();
trace(fruits_array);  // displays
  apples,cherries,oranges,pineapples,strawberries
fruits_array.sort(Array.DESCENDING);
trace(fruits_array);  // displays
  strawberries,pineapples,oranges,cherries,apples
```

Usage 2: The following example uses `Array.sort()` with a compare function:

```
var passwords_array:Array = new Array("mom:glam", "ana:ring", "jay:mag",
  "anne:home", "regina:silly");
function order(a, b):Number {
  //Entries to be sorted are in form name:password
  //Sort using only the name part of the entry as a key.
  var name1:String = a.split(":")[0];
  var name2:String = b.split(":")[0];
  if (name1<name2) {
    return -1;
  } else if (name1>name2) {
    return 1;
  } else {
    return 0;
  }
}
trace("Unsorted:");
//displays Unsorted:
trace(passwords_array);
//displays mom:glam,ana:ring,jay:mag,anne:home,regina:silly
passwords_array.sort(order);
```

```
trace("Sorted:");
//displays Sorted:
trace(passwords_array);
//displays ana:ring,anne:home,jay:mag,mom:glam,regina:silly
```

See also

`| (bitwise OR)`, Array.sortOn()

Array.sortOn()

Availability

Flash Player 6; additional capabilities added in Flash Player 7.

Usage

```
my_array.sortOn("fieldName":String ) : Array
my_array.sortOn("fieldName":String, option:Number | option |... ) : Array
my_array.sortOn( [ "fieldName" , "fieldName" , ... ]:Array) : Array
my_array.sortOn( [ "fieldName" , "fieldName" , ... ]:Array , option:Number |
   option |... ) : Array
```

Note: Where brackets ([]) are shown, you must include them in the code; that is, the brackets don't represent optional parameters.

Parameters

fieldName A string that identifies a field (in an element of the Array) to be used as the sort value.

option One or more numbers or names of defined constants, separated by the `| (bitwise OR)` operator, that change the behavior of the sort from the default. The following values are acceptable for *option*:

- 1 or `Array.CASEINSENSITIVE`
- 2 or `Array.DESCENDING`
- 4 or `Array.UNIQUESORT`
- 8 or `Array.RETURNINDEXEDARRAY`
- 16 or `Array.NUMERIC`

For more information about each option, see "Description" on page 117.

Returns

The return value depends on whether you pass any parameters, as described in the following list:

- If you specify a value of 4 or `Array.UNIQUESORT` for *option*, and two or more elements being sorted have identical sort fields, Flash returns a value of 0 and does not modify the array.
- If you specify a value of 8 or `Array.RETURNINDEXEDARRAY` for *option*, Flash returns an Array that reflects the results of the sort and does not modify the array.
- Otherwise, Flash returns nothing and modifies the array to reflect the sort order.

Description

Method; sorts the elements in an array according to one or more fields in the array. If you pass multiple *fieldName* parameters, the first field represents the primary sort field, the second represents the next sort field, and so on. Flash sorts according to Unicode values. (ASCII is a subset of Unicode.) If either of the elements being compared does not contain the field specified in the *fieldName* parameter, the field is assumed to be `undefined`, and the elements are placed consecutively in the sorted array in no particular order.

By default, `Array.sortOn()` works as described in the following list:

- Sorting is case-sensitive (*Z* precedes *a*).
- Sorting is ascending (*a* precedes *b*).
- The array is modified to reflect the sort order; multiple elements that have identical sort fields are placed consecutively in the sorted array in no particular order.
- Numeric fields are sorted as if they were strings, so 100 precedes 99, because "1" is a lower string value than "9".

You can use the *`option`* flags to override these defaults. The following examples use different forms of the *`option`* flag for illustration purposes. If you want to sort a simple array (for example, an array with only one field), or if you want to specify a sort order that the *`options`* parameter doesn't support, use Array.sort().

To pass multiple flags in numeric format, separate them with the bitwise OR (`|`) operator or add the values of the flags together. The following code shows three ways to specify a numeric descending sort:

```
my_array.sortOn(someFieldName, 2 | 16);
my_array.sortOn(someFieldName, 18);
my_array.sortOn(someFieldName, Array.DESCENDING | Array.NUMERIC);
```

Code hinting (see "Using code hints" in *Using ActionScript in Flash*) is enabled if you use the string form of the flag (for example, `DESCENDING`) rather than the numeric form (2).

Consider the following array:

```
var my_array:Array = new Array();
my_array.push({password: "Bob", age:29});
my_array.push({password: "abcd", age:3});
my_array.push({password: "barb", age:35});
my_array.push({password: "catchy", age:4});
```

Performing a default sort on the password field produces the following results:

```
my_array.sortOn("password");
// Bob
// abcd
// barb
// catchy
```

Performing a case-insensitive sort on the password field produces the following results:

```
my_array.sortOn("password", Array.CASEINSENSITIVE);
// abcd
// barb
// Bob
// catchy
```

Performing a case-insensitive, descending sort on the password field produces the following results:

```
my_array.sortOn("password", 1|2);
// catchy
// Bob
// barb
// abcd
```

Performing a default sort on the age field produces the following results:

```
my_array.sortOn("age");
// 29
// 3
// 35
// 4
```

Performing a numeric sort on the age field produces the following results:

```
my_array.sortOn("age", 16);
// 3
// 4
// 29
// 35
```

Performing a descending numeric sort on the age field produces the following results:

```
my_array.sortOn("age", 18);
// 35
// 29
// 4
// 3
```

Performing a sort changes the elements in the array produces the following results:

```
// Before sorting
// my_array[0].age = 29;
// my_array[1].age = 3;
// my_array[2].age = 35;
// my_array[3].age = 4;

// After any sort that doesn't pass a value of 8 for option
my_array.sortOn("age", Array.NUMERIC);
// my_array[0].age = 3;
// my_array[1].age = 4;
// my_array[2].age = 29;
// my_array[3].age = 35;
```

Performing a sort that returns an index array doesn't change the elements in the array, as shown in the following example:

```
// Before sorting
// my_array[0].age = 29;
// my_array[1].age = 3;
// my_array[2].age = 35;
// my_array[3].age = 4;

// After a sort that returns an array containing index values
// Note that the original array is unchanged.
// You can then use the returned array to display sorted information
// without modifying the original array.
var indexArray:Array = my_array.sortOn("age", Array.RETURNINDEXEDARRAY);
// my_array[0].age = 29;
// my_array[1].age = 3;
// my_array[2].age = 35;
// my_array[3].age = 4;
```

Example

The following example creates a new array and sorts it according to the fields `name` and `city`: The first sort uses `name` as the first sort value and `city` as the second. The second sort uses `city` as the first sort value and `name` as the second.

```
var rec_array:Array = new Array();
rec_array.push( { name: "john", city: "omaha", zip: 68144 } );
rec_array.push( { name: "john", city: "kansas city", zip: 72345 } );
rec_array.push( { name: "bob", city: "omaha", zip: 94010 } );
for(i=0; i<rec_array.length; i++) {
 trace(rec_array[i].name + ", " + rec_array[i].city);
}
// results:
// john, omaha
// john, kansas city
// bob, omaha

rec_array.sortOn( [ "name", "city" ]);
for(i=0; i<rec_array.length; i++) {
 trace(rec_array[i].name + ", " + rec_array[i].city);
}
// results:
// bob, omaha
// john, kansas city
// john, omaha

rec_array.sortOn( ["city", "name" ]);
for(i=0; i<rec_array.length; i++) {
 trace(rec_array[i].name + ", " + rec_array[i].city);
}
// results:
// john, kansas city
// bob, omaha
// john, omaha
```

See also

`| (bitwise OR)`, Array.sort()

Array.splice()

Availability

Flash Player 5.

Usage

```
my_array.splice(start:Number, deleteCount:Number [, value0:Object,
  value1...valueN]) : Array
```

Parameters

`start` An integer that specifies the index of the element in the array where the insertion or deletion begins.

`deleteCount` An integer that specifies the number of elements to be deleted. This number includes the element specified in the `start` parameter. If no value is specified for `deleteCount`, the method deletes all the values from the `start` element to the last element in the array. If the value is 0, no elements are deleted.

`value` An optional parameter that specifies the values to insert into the array at the insertion point specified in the `start` parameter.

Returns

The elements that were removed from the array.

Description

Method; adds and removes elements from an array. This method modifies the array without making a copy.

Example

The following example creates an array and splices it using element index 1 for the `start` parameter. This removes all elements from the array starting with the second element, leaving only the element at index 0 in the original array:

```
var myPets_array:Array = new Array("cat", "dog", "bird", "fish");
trace( myPets_array.splice(1) ); // dog,bird,fish
trace( myPets_array ); // cat
```

The following example creates an array and splices it using element index 1 for the `start` parameter and the number 2 for the `deleteCount` parameter. This removes two elements from the array, starting with the second element, leaving the first and last elements in the original array:

```
var myFlowers_array:Array = new Array("roses", "tulips", "lilies", "orchids");
trace( myFlowers_array.splice(1,2 ) ); // tulips,lilies
trace( myFlowers_array ); // roses,orchids
```

The following example creates an array and splices it using element index 1 for the `start` parameter, the number 0 for the `deleteCount` parameter, and the string `chair` for the `value` parameter. This does not remove anything from the original array, and adds the string `chair` at index 1:

```
var myFurniture_array:Array = new Array("couch", "bed", "desk", "lamp");
trace( myFurniture_array.splice(1,0, "chair" ) ); // displays empty array
trace( myFurniture_array ); // displays couch,chair,bed,desk,lamp
```

Array.toString()

Availability

Flash Player 5.

Usage

my_array.toString() : *String*

Parameters

None.

Returns

A string.

Description

Method; returns a String value representing the elements in the specified Array object. Every element in the array, starting with index 0 and ending with index *my_array*.length-1, is converted to a concatenated string and separated by commas. To specify a custom separator, use Array.join().

Example

The following example creates my_array and converts it to a string:

The following example creates my_array, converts it to a string, and outputs 1,2,3,4,5 as a result of the trace statement:

```
my_array:Array = new Array();
my_array[0] = 1;
my_array[1] = 2;
my_array[2] = 3;
my_array[3] = 4;
my_array[4] = 5;
trace(my_array.toString()); // displays 1,2,3,4,5
```

See Also

Array.join(), String.split()

Array.unshift()

Availability

Flash Player 5.

Usage

`my_array.unshift(value1:Object,value2,...valueN) : Number`

Parameters

`value1,...valueN` One or more numbers, elements, or variables to be inserted at the beginning of the array.

Returns

An integer representing the new length of the array.

Description

Method; adds one or more elements to the beginning of an array and returns the array's new length.

Example

The following example shows the use of `Array.unshift()`:

```
var pets_array:Array = new Array("dog", "cat", "fish");
trace( pets_array ); // dog,cat,fish
pets_array.unshift("ferrets", "gophers", "engineers");
trace( pets_array ); // ferrets,gophers,engineers,dog,cat,fish
```

asfunction

Availability

Flash Player 5.

Usage

`asfunction:`*`function:Function`*`,`*`"parameter":String`*

Parameters

`function` An identifier for a function.

`parameter` A string that is passed to the function named in the *`function`* parameter.

Returns

Nothing.

Description

Protocol; a special protocol for URLs in HTML text fields. In HTML text fields, text can be linked using the HTML `A` tag. The `HREF` attribute of the `A` tag contains a URL that can be for a standard protocol such as HTTP, HTTPS, or FTP. The `asfunction` protocol is an additional protocol that is specific to Flash, which causes the link to invoke an ActionScript function.

Example

In the following example, the `playMP3()` function is defined. The TextField object `list_txt` is created and set so HTML text can be rendered. The text `Track 1` and `Track 2` are links inside the text field. The `playMP3()` function is called when the user clicks either link and plays the MP3 that is passed as a parameter of the asfunction call.

```
var myMP3:Sound = new Sound();
function playMP3(mp3:String) {
  myMP3.loadSound(mp3, true);
  myMP3.onLoad = function(success) {
    if (!success) {
      // code to handle errors here
    }
  };
}
this.createTextField("list_txt", this.getNextHighestDepth(), 0, 0, 200, 100);
list_txt.autoSize = true;
list_txt.html = true;
list_txt.multiline = true;
list_txt.htmlText = "<a href=\"asfunction:playMP3,track1.mp3\">Track 1</
  a><br>";
list_txt.htmlText += "<a href=\"asfunction:playMP3,track2.mp3\">Track 2</
  a><br>";
```

When you click a link, the MP3 sound file streams in Flash Player.

Boolean()

Availability

Flash Player 5; behavior changed in Flash Player 7.

Usage

`Boolean(`*`expression`*`) : `*`Boolean`*

Parameters

`expression`	An expression to convert to a Boolean value.

Returns

A Boolean value.

Description

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

- If *`expression`* is a Boolean value, the return value is *`expression`*.
- If *`expression`* is a number, the return value is `true` if the number is not zero; otherwise the return value is `false`.

If *`expression`* is a string, the return value is as follows:

- In files published for Flash Player 6 or earlier, the string is first converted to a number; the value is `true` if the number is not zero, `false` otherwise.
- In files published for Flash Player 7 or later, the result is `true` if the string has a length greater than zero; the value is `false` for an empty string.
- If *`expression`* is `undefined` or `NaN` (not a number), the return value is `false`.
- If *`expression`* is a movie clip or an object, the return value is `true`.

Note: Unlike the Boolean class constructor, the `Boolean()` function does not use the keyword `new`. Moreover, the Boolean class constructor initializes a Boolean object to `false` if no parameter is specified, while the `Boolean()` function returns `undefined` if no parameter is specified.

Example

In the following example, expressions are converted from numeric to Boolean values:

```
trace(Boolean(-1)); // output: true
trace(Boolean(0)); // output: false
trace(Boolean(1)); // output: true

trace(Boolean(true)); // output: true
trace(Boolean(false)); // output: false

trace(Boolean("true")); // output: true
trace(Boolean("false")); // output: true

trace(Boolean("Craiggers")); // output: true
trace(Boolean("")); // output: false
```

If files are published for Flash Player 6 or earlier, the results will differ for three of the preceding examples:

```
trace(Boolean("true")); // output: false
trace(Boolean("false")); // output: false
trace(Boolean("Craiggers")); // output: false
```

This example shows a string that will evaluate as `true` if the file is published for Flash Player 7, but will evaluate as `false` for Flash Player 6 or earlier:

```
trace(Boolean("0"));
```

This example shows a significant difference between use of the `Boolean()` function and the Boolean class. The `Boolean()` function creates a Boolean value, and the Boolean class creates a Boolean object. Boolean values are compared by value, and Boolean objects are compared by reference.

```
// Variables representing Boolean values are compared by value
var a:Boolean = Boolean("a"); // a is true
var b:Boolean = Boolean(1); // b is true
trace(a==b); // true

// Variables representing Boolean objects are compared by reference
var a:Boolean = new Boolean("a"); // a is true
var b:Boolean = new Boolean(1); // b is true
trace(a==b); // false
```

See also

Boolean class

Boolean class

Availability

Flash Player 5 (became a native object in Flash Player 6, which improved performance significantly).

Description

The Boolean class is a wrapper object with the same functionality as the standard JavaScript Boolean object. Use the Boolean class to retrieve the primitive data type or string representation of a Boolean object.

You must use the constructor `new Boolean()` to create a Boolean object before calling its methods.

Method summary for the Boolean class

Method	Description
`Boolean.toString()`	Returns the string representation (`"true"` or `"false"`) of the Boolean object.
`Boolean.valueOf()`	Returns the primitive value type of the specified Boolean object.

Constructor for the Boolean class

Availability

Flash Player 5.

Usage

```
new Boolean([x]) : Boolean
```

Parameters

x Any expression. This parameter is optional.

Returns

A reference to a Boolean object.

Description

Constructor; creates a Boolean object. If you omit the *x* parameter, the Boolean object is initialized with a value of `false`. If you specify a value for the *x* parameter, the method evaluates it and returns the result as a Boolean value according to the rules in the `Boolean()` function.

Example

The following code creates a new empty Boolean object called `myBoolean`:

```
var myBoolean:Boolean = new Boolean();
```

Boolean.toString()

Availability

Flash Player 5.

Usage

myBoolean.toString() *: String*

Parameters

None.

Returns

A string; "true" or "false".

Description

Method; returns the string representation ("true" or "false") of the Boolean object.

Example

This example creates a variable of type Boolean and uses toString() to convert the value to a string for use in the trace statement:

```
var myBool:Boolean = true;
trace("The value of the Boolean myBool is: " + myBool.toString());
myBool = false;
trace("The value of the Boolean myBool is: " + myBool.toString());
```

Boolean.valueOf()

Availability

Flash Player 5.

Usage

```
myBoolean.valueOf() : Boolean
```

Parameters

None.

Returns

A Boolean value.

Description

Method: returns `true` if the primitive value type of the specified Boolean object is true; `false` otherwise.

Example

The following example shows how this method works, and also shows that the primitive value type of a new Boolean object is `false`:

```
var x:Boolean = new Boolean();
trace(x.valueOf());   // false
x = (6==3+3);
trace(x.valueOf());   // true
```

break

Availability

Flash Player 4.

Usage

```
break
```

Parameters

None.

Returns

Nothing.

Description

Statement; appears within a loop (`for`, `for..in`, `do while` or `while`) or within a block of statements associated with a particular case within a `switch` statement. When used in a loop, the `break` statement instructs Flash to skip the rest of the loop body, stop the looping action, and execute the statement following the loop statement. When used in a `switch`, the `break` statement instructs Flash to skip the rest of the statements in that `case` block and jump to the first statement following the enclosing `switch` statement.

In nested loops, the `break` statement only skips the rest of the immediate loop and does not break out of the entire series of nested loops. For breaking out of an entire series of nested loops, see `try..catch..finally`.

Example

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

```
var i:Number = 0;
while (true) {
  trace(i);
  if (i>=10) {
    break;  // this will terminate/exit the loop
  }
  i++;
}
```

which traces the following output:

```
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
  10
```

See also

`for`, `for..in`, `do while`, `while`, `switch`, `case`, `continue`, `throw`, `try..catch..finally`

Button class

Availability

Flash Player 6.

Description

All button symbols in a SWF file are instances of the Button object. You can give a button an instance name in the Property inspector, and use the methods and properties of the Button class to manipulate buttons with ActionScript. Button instance names are displayed in the Movie Explorer and in the Insert Target Path dialog box in the Actions panel.

The Button class inherits from the Object class.

Method summary for the Button class

Method	Description
`Button.getDepth()`	Returns the depth of a button instance.

Property summary for the Button class

Property	Description
`Button._alpha`	The transparency value of a button instance.
`Button.enabled`	A Boolean value that indicates whether a button is active.
`Button._focusrect`	A Boolean value that indicates whether a button with focus has a yellow rectangle around it.
`Button._height`	The height of a button instance, in pixels.
`Button._quality`	The level of anti-aliasing applied to the current SWF file.
`Button.menu`	A reference to an associated ContextMenu object
`Button._name`	The instance name of a button instance.
`Button._parent`	A reference to the movie clip or object that contains the current movie clip or object.
`Button._quality`	A string that indicates the rendering quality of the SWF file.
`Button._rotation`	The degree of rotation of a button instance.
`Button._soundbuftime`	Number of seconds for a sound to preload.
`Button.tabEnabled`	A Boolean value that indicates whether a button is included in automatic tab ordering.
`Button.tabIndex`	A number that indicates the tab order of an object.
`Button._target`	Read-only; the target path of a button instance.
`Button.trackAsMenu`	A Boolean value that indicates whether other buttons can receive mouse release events.
`Button._url`	Read-only; the URL of the SWF file that created the button instance.

Property	Description
`Button.useHandCursor`	A Boolean value that indicates whether the pointing hand is displayed when the mouse passes over a button.
`Button._visible`	A Boolean value that indicates whether a button instance is hidden or visible.
`Button._width`	The width of a button instance, in pixels.
`Button._x`	The *x* coordinate of a button instance.
`Button._xmouse`	Read-only; the *x* coordinate of the mouse pointer relative to a button instance.
`Button._xscale`	The value specifying the percentage for horizontally scaling a button instance.
`Button._y`	The *y* coordinate of a button instance.
`Button._ymouse`	Read-only; the *y* coordinate of the mouse pointer relative to a button instance.
`Button._yscale`	The value specifying the percentage for vertically scaling a button instance.

Event handler summary for the Button class

Event handler	Description
`Button.onDragOut`	Invoked when the mouse button is pressed over the button and the pointer then rolls outside the button.
`Button.onDragOver`	Invoked when the user presses and drags the mouse button outside and then over the button.
`Button.onKeyUp`	Invoked when a key is released.
`Button.onKillFocus`	Invoked when focus is removed from a button.
`Button.onPress`	Invoked when the mouse button is pressed while the pointer is over a button.
`Button.onRelease`	Invoked when the mouse button is released while the pointer is over a button.
`Button.onReleaseOutside`	Invoked when the mouse button is released while the pointer is outside the button after the button is pressed while the pointer is inside the button.
`Button.onRollOut`	Invoked when the mouse pointer rolls outside of a button area.
`Button.onRollOver`	Invoked when the mouse pointer rolls over a button.
`Button.onSetFocus`	Invoked when a button has input focus and a key is released.

Button._alpha

Availability

Flash Player 6.

Usage

my_btn._alpha:*Number*

Description

Property; the alpha transparency value of the button specified by *my_btn*. Valid values are 0 (fully transparent) to 100 (fully opaque). The default value is 100. Objects in a button with _alpha set to 0 are active, even though they are invisible.

Example

The following code sets the _alpha property of a button named myBtn_btn to 50% when the user clicks the button:

```
// add a Button instance on the Stage
// give it an instance name of myBtn_btn
// with frame 1 selected, place the following code using the Actions panel
myBtn_btn.onRelease = function(){
  this._alpha = 50;
};
```

See also

MovieClip._alpha, TextField._alpha

Button.enabled

Availability

Flash Player 6.

Usage

my_btn.enabled:*Boolean*

Description

Property; a Boolean value that specifies whether a button is enabled. When a button is disabled (the enabled property is set to `false`), the button is visible but cannot be clicked. The default value is `true`. This property is useful if you want to disable part of your navigation; for example, you may want to disable a button in the currently displayed page so that it can't be clicked and the page cannot be reloaded.

Example

The following example demonstrates how you can disable and enable buttons from being clicked. Two buttons, `myBtn1_btn` and `myBtn2_btn`, are on the Stage and the following ActionScript is added so that the `myBtn2_btn` button cannot be clicked:

```
// add two button instances on the Stage
// give them instance names of myBtn1_btn and myBtn2_btn
// place the following code on frame 1
// to enable or disable buttons

myBtn1_btn.enabled = true;
myBtn2_btn.enabled = false;

//button code
// the following function will not get called
// because myBtn2_btn.enabled was set to false
myBtn1_btn.onRelease = function() {
  trace( "you clicked : " + this._name );
};
myBtn2_btn.onRelease = function() {
  trace( "you clicked : " + this._name );
};
```

Button._focusrect

Availability

Flash Player 6.

Usage

my_btn._focusrect:*Boolean*

Description

Property; a Boolean value that specifies whether a button has a yellow rectangle around it when it has keyboard focus. This property can override the global _focusrect property. By default, the `_focusrect` property of a button instance is null; meaning, the button instance does not override the global `_focusrect` property. If the `_focusrect` property of a button instance is set to `true` or `false`, it overrides the setting of the global `_focusrect` property for the single button instance.

In Flash Player 4 or Flash Player 5 SWF files, the `_focusrect` property controls the global `_focusrect` property. It is a Boolean value. This behavior was changed in Flash Player 6 and later to permit customizing the `_focusrect` property on an individual movie clip.

Example

This example demonstrates how to hide the yellow rectangle around a specified button instance in a SWF file when it has focus in a browser window. Create three buttons called `myBtn1_btn`, `myBtn2_btn`, and `myBtn3_btn`, and add the following ActionScript to Frame 1 of the Timeline:

```
myBtn2_btn._focusrect = false;
```

Change the publish settings to Flash Player 6, and test the SWF file in a browser window by selecting File > Publish Preview > HTML. Give the SWF focus by clicking it in the browser window, and use the Tab key to focus each instance. You will not be able to execute code for this button by pressing Enter or the Space key when `_focusrect` is disabled.

Button.getDepth()

Availability

Flash Player 6.

Usage

```
my_btn.getDepth() : Number
```

Returns

An integer.

Description

Method; returns the depth of a button instance.

Example

If you create `myBtn1_btn` and `myBtn2_btn` on the Stage, you can trace their depth using the following ActionScript:

```
trace(myBtn1_btn.getDepth());
trace(myBtn2_btn.getDepth());
```

If you load a SWF file called buttonMovie.swf into this document, you could trace the depth of a button, `myBtn4_btn`, inside that SWF file using another button in the main SWF.

```
this.createEmptyMovieClip("myClip_mc", 999);
myClip_mc.loadMovie("buttonMovie.swf");
myBtn3_btn.onRelease = function(){
  trace(myClip_mc.myBtn4_btn.getDepth());
};
```

You might notice that two of these buttons have the same depth value, one in the main SWF file and one in the loaded SWF file. This is misleading, because buttonMovie.swf was loaded at depth 999, which means that the button it contains will also have a depth of 999 relative to the buttons in the main SWF file. Keep in mind that each movie clip has its own internal z-order, which means that each movie clip has its own set of depth values. The two buttons may have the same depth value, but the values only have meaning in relation to other objects in the same z-order. In this case, the buttons have the same depth value, but the values relate to different movie clips: the depth value of the button in the main SWF file relates to the z-order of the main Timeline, while the depth value of the button in the loaded SWF file relates to the internal z-order of the myClip_mc movie clip.

Button._height

Availability

Flash Player 6.

Usage

my_btn._height:*Number*

Description

Property; the height of the button, in pixels.

Example

The following example sets the height and width of a button called `my_btn` to a specified width and height.

```
my_btn._width = 500;
my_btn._height = 200;
```

Button.menu

Availability

Flash Player 7.

Usage

my_btn.menu = *contextMenu*

Parameters

contextMenu A ContextMenu object.

Description

Property; associates the ContextMenu object *contextMenu* with the button object *my_button*. The ContextMenu class lets you modify the context menu that appears when the user right-clicks (Windows) or Control-clicks (Macintosh) in Flash Player.

Example

The following example assigns a ContextMenu object to a button instance named myBtn_btn. The ContextMenu object contains a single menu item (labeled "Save...") with an associated callback handler function named doSave.

Add the button instance to the Stage and name it myBtn_btn.

```
var menu_cm:ContextMenu = new ContextMenu();
menu_cm.customItems.push(new ContextMenuItem("Save...", doSave));
function doSave(menu:Object, obj:Object):Void {
  trace( " You selected the 'Save...' menu item ");
}
myBtn_btn.menu = menu_cm;
```

Select Control > Test Movie to test the SWF file. With the pointer over myBtn_btn, right-click or Control-click. The context menu appears with Save in the menu. When you select Save from the menu, the Output panel appears.

See also

ContextMenu class, ContextMenuItem class, MovieClip.menu, TextField.menu

Button._name

Availability

Flash Player 6.

Usage

my_btn._name:*String*

Description

Property; instance name of the button specified by *my_btn*.

Example

The following example traces all instance names of any Button instances within the current Timeline of a SWF file.

```
for (i in this) {
  if (this[i] instanceof Button) {
    trace(this[i]._name);
  }
}
```

Button.onDragOut

Availability

Flash Player 6.

Usage

```
my_btn.onDragOut = function() : Void {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the mouse button is clicked over the button and the pointer then dragged outside of the button.

You must define a function that executes when the event handler is invoked.

Example

The following example demonstrates how you can execute statements when the pointer is dragged off a button. Create a button called my_btn on the Stage and enter the following ActionScript in a frame on the Timeline:

```
my_btn.onDragOut = function() {
  trace("onDragOut: "+this._name);
};
my_btn.onDragOver = function() {
  trace("onDragOver: "+this._name);
};
```

Button.onDragOver

Availability

Flash Player 6.

Usage

```
my_btn.onDragOver = function() : Void {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the user presses and drags the mouse button outside and then over the button.

You must define a function that executes when the event handler is invoked.

Example

The following example defines a function for the `onDragOver` handler that sends a trace() statement to the Output panel. Create a button called `my_btn` on the Stage and enter the following ActionScript on the Timeline:

```
my_btn.onDragOut = function() {
  trace("onDragOut: "+this._name);
};
my_btn.onDragOver = function() {
  trace("onDragOver: "+this._name);
};
```

When you test the SWF file, drag the pointer off the button instance. Then, while holding the mouse button, drag onto the button instance again. Notice that the Output panel tracks your movements.

See also

Button.onDragOut

Button.onKeyDown

Availability

Flash Player 6.

Usage

```
my_btn.onKeyDown = function() : Void {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when a button has keyboard focus and a key is pressed. The `onKeyDown` event handler is invoked with no parameters. You can use the Key.getAscii() and Key.getCode() methods to determine which key was pressed.

You must define a function that executes when the event handler is invoked.

Example

In the following example, a function that sends text to the Output panel is defined for the `onKeyDown` handler. Create a button called `my_btn` on the Stage, and enter the following ActionScript in a frame on the Timeline:

```
my_btn.onKeyDown = function() {
  trace("onKeyDown: "+this._name+" (Key: "+getKeyPressed()+")");
};
function getKeyPressed():String {
  var theKey:String;
  switch (Key.getAscii()) {
  case Key.BACKSPACE :
    theKey = "<BACKSPACE>";
    break;
  case Key.SPACE :
    theKey = "<SPACE>";
    break;
  default :
    theKey = chr(Key.getAscii());
  }
  return theKey;
}
```

Select Control > Test Movie to test the SWF file. Make sure you select Control > Disable Keyboard Shortcuts in the test environment. Then press the Tab key until the button has focus (a yellow rectangle appears around the `my_btn` instance) and start pressing keys on your keyboard. When you press keys, they are displayed in the Output panel.

See also

Button.onKeyUp

Button.onKeyUp

Availability

Flash Player 6.

Usage

```
my_btn.onKeyUp = function() : Void {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when a button has input focus and a key is released. The `onKeyUp` event handler is invoked with no parameters. You can use the Key.getAscii() and Key.getCode() methods to determine which key was pressed.

You must define a function that executes when the event handler is invoked.

Example

In the following example, a function that sends text to the Output panel is defined for the `onKeyDown` handler. Create a button called `my_btn` on the Stage, and enter the following ActionScript in a frame on the Timeline:

```
my_btn.onKeyDown = function() {
  trace("onKeyDown: "+this._name+" (Key: "+getKeyPressed()+")");
};
my_btn.onKeyUp = function() {
  trace("onKeyUp: "+this._name+" (Key: "+getKeyPressed()+")");
};
function getKeyPressed():String {
  var theKey:String;
  switch (Key.getAscii()) {
  case Key.BACKSPACE :
    theKey = "<BACKSPACE>";
    break;
  case Key.SPACE :
    theKey = "<SPACE>";
    break;
  default :
    theKey = chr(Key.getAscii());
  }
  return theKey;
}
```

Press Control+Enter to test the SWF file. Make sure you select Control > Disable Keyboard Shortcuts in the test environment. Then press the Tab key until the button has focus (a yellow rectangle appears around the `my_btn` instance) and start pressing keys on your keyboard. When you press keys, they are displayed in the Output panel.

Button.onKillFocus

Availability

Flash Player 6.

Usage

```
my_btn.onKillFocus = function (newFocus:Object) : Void {
  // your statements here
}
```

Parameters

newFocus	The object that is receiving the focus.

Returns

Nothing.

Description

Event handler; invoked when a button loses keyboard focus. The `onKillFocus` handler receives one parameter, *newFocus*, which is an object representing the new object receiving the focus. If no object receives the focus, *newFocus* contains the value `null`.

Example

The following example demonstrates how statements can be executed when a button loses focus. Create a button instance on the Stage called my_btn and add the following ActionScript to Frame 1 of the Timeline:

```
this.createTextField("output_txt", this.getNextHighestDepth(), 0, 0, 300,
  200);
output_txt.wordWrap = true;
output_txt.multiline = true;
output_txt.border = true;
my_btn.onKillFocus = function() {
  output_txt.text = "onKillFocus: "+this._name+newline+output_txt.text;
};
```

Test the SWF file in a browser window, and try using the Tab key to move through the elements in the window. When the button instance loses focus, text is sent to the `output_txt` text field.

Button.onPress

Availability

Flash Player 6.

Usage

```
my_btn.onPress = function() : Void {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when a button is pressed. You must define a function that executes when the event handler is invoked.

Example

In the following example, a function that sends a trace() statement to the Output panel is defined for the `onPress` handler.

```
my_btn.onPress = function () {
  trace ("onPress called");
};
```

Button.onRelease

Availability

Flash Player 6.

Usage

```
my_btn.onRelease = function() : Void {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when a button is released. You must define a function that executes when the event handler is invoked.

Example

In the following example, a function that sends a trace() statement to the Output panel is defined for the `onRelease` handler.

```
my_btn.onRelease = function () {
  trace ("onRelease called");
};
```

Button.onReleaseOutside

Availability

Flash Player 6.

Usage

```
my_btn.onReleaseOutside = function() : Void {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the mouse is released while the pointer is outside the button after the button is pressed while the pointer is inside the button.

You must define a function that executes when the event handler is invoked.

Example

In the following example, a function that sends a trace() statement to the Output panel is defined for the `onReleaseOutside` handler.

```
my_btn.onReleaseOutside = function () {
  trace ("onReleaseOutside called");
};
```

Button.onRollOut

Availability

Flash Player 6.

Usage

```
my_btn.onRollOut = function() : Void {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the pointer moves outside a button area. You must define a function that executes when the event handler is invoked.

Example

In the following example, a function that sends a trace() statement to the Output panel is defined for the `onRollOut` handler.

```
my_btn.onRollOut = function () {
  trace ("onRollOut called");
};
```

Button.onRollOver

Availability

Flash Player 6.

Usage

```
my_btn.onRollOver = function() : Void {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the pointer moves over a button area. You must define a function that executes when the event handler is invoked.

Example

In the following example, a function that sends a trace() statement to the Output panel is defined for the `onRollOver` handler.

```
my_btn.onRollOver = function () {
  trace ("onRollOver called");
};
```

Button.onSetFocus

Availability

Flash Player 6.

Usage

```
my_btn.onSetFocus = function(oldFocus:Object) : Void {
  // your statements here
}
```

Parameters

oldFocus The object to lose keyboard focus.

Returns

Nothing.

Description

Event handler; invoked when a button receives keyboard focus. The *oldFocus* parameter is the object that loses the focus. For example, if the user presses the Tab key to move the input focus from a text field to a button, *oldFocus* contains the text field instance.

If there is no previously focused object, *oldFocus* contains a `null` value.

Example

The following example demonstrates how you can execute statements when the user of a SWF file moves focus from one button to another. Create two buttons, `btn1_btn` and `btn2_btn`, and enter the following ActionScript in Frame 1 of the Timeline:

```
Selection.setFocus(btn1_btn);
trace(Selection.getFocus());
btn2_btn.onSetFocus = function(oldFocus) {
  trace(oldFocus._name + " lost focus");
};
```

Test the SWF file by pressing Control+Enter. Make sure you select Control > Disable Keyboard Shortcuts if it is not already selected. Focus is set on `btn1_btn`. When `btn1_btn` loses focus and `btn2_btn` gains focus, information is displayed in the Output panel.

Button._parent

Availability

Flash Player 6.

Usage

```
my_btn._parent:MovieClip.property
```

```
_parent:MovieClip.property
```

Description

Property; a reference to the movie clip or object that contains the current movie clip or object. The current object is the one containing the ActionScript code that references `_parent`.

Use `_parent` to specify a relative path to movie clips or objects that are above the current movie clip or object. You can use `_parent` to move up multiple levels in the display list as in the following:

```
this._parent._parent._alpha = 20;
```

Example

In the following example, a button named `my_btn` is placed inside a movie clip called `my_mc`. The following code shows how to use the `_parent` property to get a reference to the movie clip `my_mc`:

```
trace(my_mc.my_btn._parent);
```

The Output panel displays the following:

```
_level0.my_mc
```

See also

MovieClip._parent, _root, targetPath()

Button._quality

Availability

Flash Player 6.

Usage

`my_btn._quality:String`

Description

Property (global); sets or retrieves the rendering quality used for a SWF file. Device fonts are always aliased and therefore are unaffected by the `_quality` property.

The `_quality` property can be set to the following values:

- `"LOW"` Low rendering quality. Graphics are not anti-aliased, and bitmaps are not smoothed.
- `"MEDIUM"` Medium rendering quality. Graphics are anti-aliased using a 2 x 2 pixel grid, but bitmaps are not smoothed. This is suitable for movies that do not contain text.
- `"HIGH"` High rendering quality. Graphics are anti-aliased using a 4 x 4 pixel grid, and bitmaps are smoothed if the movie is static. This is the default rendering quality setting used by Flash.
- `"BEST"` Very high rendering quality. Graphics are anti-aliased using a 4 x 4 pixel grid and bitmaps are always smoothed.

Note: Although you can specify this property for a Button object, it is actually a global property, and you can specify its value simply as `_quality`. For more information, see _quality.

Example

This example sets the rendering quality of a button named `my_btn` to `LOW`:

```
my_btn._quality = "LOW";
```

Button._rotation

Availability

Flash Player 6.

Usage

my_btn._rotation:*Number*

Description

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

Example

The following example rotates two buttons on the Stage. Create two buttons on the Stage called `control_btn` and `my_btn`. Make sure that `my_btn` is not perfectly round, so you can see it rotating. Then enter the following ActionScript in Frame 1 of the Timeline:

```
var control_btn:Button;
var my_btn:Button;
control_btn.onRelease = function() {
  my_btn._rotation += 10;
};
```

Now create another button on the Stage called `myOther_btn`, making sure it isn't perfectly round (so you can see it rotate). Enter the following ActionScript in Frame 1 of the Timeline.

```
var myOther_btn:Button;
this.createEmptyMovieClip("rotater_mc", this.getNextHighestDepth());
rotater_mc.onEnterFrame = function() {
  myOther_btn._rotation += 2;
};
```

See also

MovieClip._rotation, TextField._rotation

Button._soundbuftime

Availability

Flash Player 6.

Usage

`my_btn._soundbuftime:Number`

Description

Property (global); an integer that specifies the number of seconds a sound prebuffers before it starts to stream.

Note: Although you can specify this property for a Button object, it is actually a global property, and you can specify its value simply as `_soundbuftime`. For more information and an example, see _soundbuftime.

Button.tabEnabled

Availability

Flash Player 6.

Usage

`my_btn.tabEnabled:Boolean`

Description

Property; specifies whether `my_btn` is included in automatic tab ordering. It is `undefined` by default.

If the `tabEnabled` property is `undefined` or `true`, the object is included in automatic tab ordering. If the `tabIndex` property is also set to a value, the object is included in custom tab ordering as well. If `tabEnabled` is `false`, the object is not included in automatic or custom tab ordering, even if the `tabIndex` property is set.

Example

The following ActionScript is used to set the `tabEnabled` property for one of four buttons to `false`. However, all four buttons (`one_btn`, `two_btn`, `three_btn`, and `four_btn`) are placed in a custom tab order using `tabIndex`. Although `tabIndex` is set for `three_btn`, `three_btn` is not included in a custom or automatic tab order because `tabEnabled` is set to `false` for that instance. To set the tab ordering for the four buttons, add the following ActionScript to Frame 1 of the Timeline:

```
three_btn.tabEnabled = false;
two_btn.tabIndex = 1;
four_btn.tabIndex = 2;
three_btn.tabIndex = 3;
one_btn.tabIndex = 4;
```

Make sure that you disable keyboard shortcuts when you test the SWF file by selecting Control > Disable Keyboard Shortcuts in the test environment.

See also

`Button.tabIndex`, `MovieClip.tabEnabled`, `TextField.tabEnabled`

Button.tabIndex

Availability

Flash Player 6.

Usage

my_btn.tabIndex:*Number*

Description

Property; lets you customize the tab ordering of objects in a SWF file. You can set the tabIndex property on a button, movie clip, or text field instance; it is undefined by default.

If any currently displayed object in the SWF file contains a tabIndex property, automatic tab ordering is disabled, and the tab ordering is calculated from the tabIndex properties of objects in the SWF file. The custom tab ordering only includes objects that have tabIndex properties.

The tabIndex property may be a non-negative integer. The objects are ordered according to their tabIndex properties, in ascending order. An object with a tabIndex value of 1 precedes an object with a tabIndex value of 2. If two objects have the same tabIndex value, the one that precedes the other in the tab ordering is undefined.

The custom tab ordering defined by the tabIndex property is *flat*. This means that no attention is paid to the hierarchical relationships of objects in the SWF file. All objects in the SWF file with tabIndex properties are placed in the tab order, and the tab order is determined by the order of the tabIndex values. If two objects have the same tabIndex value, the one that goes first is undefined. You shouldn't use the same tabIndex value for multiple objects.

Example

The following ActionScript is used to set the tabEnabled property for one of four buttons to false. However, all four buttons (one_btn, two_btn, three_btn, and four_btn) are placed in a custom tab order using tabIndex. Although tabIndex is set for three_btn, three_btn is not included in a custom or automatic tab order because tabEnabled is set to false for that instance. To set the tab ordering for the four buttons, add the following ActionScript to Frame 1 of the Timeline:

```
three_btn.tabEnabled = false;
two_btn.tabIndex = 1;
four_btn.tabIndex = 2;
three_btn.tabIndex = 3;
one_btn.tabIndex = 4;
```

Make sure that you disable keyboard shortcuts when you test the SWF file by selecting Control > Disable Keyboard Shortcuts in the test environment.

See also

Button.tabEnabled, MovieClip.tabChildren, MovieClip.tabEnabled, MovieClip.tabIndex, TextField.tabIndex

Button._target

Availability

Flash Player 6.

Usage

my_btn._target:*String*

Description

Read-only property; returns the target path of the button instance specified by *my_btn*.

Example

Add a button instance to the Stage with an instance name my_btn and add the following code to Frame 1 of the Timeline:

```
trace(my_btn._target); //displays /my_btn
```

Select my_btn and convert it to a movie clip. Give the new movie clip an instance name my_mc. Delete the existing ActionScript in Frame 1 of the Timeline and replace it with:

```
my_mc.my_btn.onRelease = function(){
  trace(this._target); //displays /my_mc/my_btn
};
```

To convert the notation from slash notation to dot notation, modify the previous code example to the following:

```
my_mc.my_btn.onRelease = function(){
  trace(eval(this._target)); //displays _level0.my_mc.my_btn
};
```

This lets you access methods and parameters of the target object, such as:

```
my_mc.my_btn.onRelease = function(){
  var target_btn:Button = eval(this._target);
trace(target_btn._name); //displays my_btn
};
```

See also

targetPath()

Button.trackAsMenu

Availability

Flash Player 6.

Usage

my_btn.trackAsMenu:*Boolean*

Description

Property; a Boolean value that indicates whether other buttons or movie clips can receive mouse release events. If you drag a button and then release on a second button, the `onRelease` event is registered for the second button. This allows you to create menus. You can set the `trackAsMenu` property on any button or movie clip object. If the `trackAsMenu` property has not been defined, the default behavior is `false`.

You can change the `trackAsMenu` property at any time; the modified button immediately takes on the new behavior.

Example

The following example demonstrates how to track two buttons as a menu. Place two button instances on the Stage called `one_btn` and `two_btn`. Enter the following ActionScript in the Timeline:

```
var one_btn:Button;
var two_btn:Button;
one_btn.trackAsMenu = true;
two_btn.trackAsMenu = true
one_btn.onRelease = function() {
  trace("clicked one_btn");
};
two_btn.onRelease = function() {
  trace("clicked two_btn");
};
```

Test the SWF file by clicking the Stage over `one_btn`, holding the mouse button down and releasing it over `two_btn`. Then try commenting out the two lines of ActionScript that contain `trackAsMenu` and test the SWF file again to see the difference in button behavior.

See also

MovieClip.trackAsMenu

Button._url

Availability

Flash Player 6.

Usage

my_btn._url*:String*

Description

Read-only property; retrieves the URL of the SWF file that created the button.

Example

Create two button instances on the Stage called `one_btn` and `two_btn`. Enter the following ActionScript in Frame 1 of the Timeline:

```
var one_btn:Button;
var two_btn:Button;
this.createTextField("output_txt", 999, 0, 0, 100, 22);
output_txt.autoSize = true;
one_btn.onRelease = function() {
  trace("clicked one_btn");
  trace(this._url);
};
two_btn.onRelease = function() {
  trace("clicked "+this._name);
  var url_array:Array = this._url.split("/");
  var my_str:String = String(url_array.pop());
  output_txt.text = unescape(my_str);
};
```

When you click each button, the file name of the SWF containing the buttons displays in the Output panel.

Button.useHandCursor

Availability

Flash Player 6.

Usage

my_btn.useHandCursor:*Boolean*

Description

Property; a Boolean value that, when set to true (the default), indicates whether a pointing hand (hand cursor) displays when the mouse rolls over a button. If this property is set to false, the arrow pointer is used instead.

You can change the useHandCursor property at any time; the modified button immediately takes on the new cursor behavior. The useHandCursor property can be read out of a prototype object.

Example

Create two buttons on the Stage with the instance names myBtn1_btn and myBtn2_btn. Enter the following ActionScript in Frame 1 of the Timeline:

```
myBtn1_btn.useHandCursor = false;
myBtn1_btn.onRelease = buttonClick;
myBtn2_btn.onRelease = buttonClick;
function buttonClick() {
  trace(this._name);
}
```

When the mouse is over and clicks myBtn1_btn, there is no pointing hand. However, you see the pointing hand when the button is over and clicks myBtn2_btn.

Button._visible

Availability

Flash Player 6.

Usage

my_btn._visible:*Boolean*

Description

Property; a Boolean value that indicates whether the button specified by *my_btn* is visible. Buttons that are not visible (_visible property set to false) are disabled.

Example

Create two buttons on the Stage with the instance names myBtn1_btn and myBtn2_btn. Enter the following ActionScript in Frame 1 of the Timeline:

```
myBtn1_btn.onRelease = function() {
  this._visible = false;
  trace("clicked "+this._name);
};
myBtn2_btn.onRelease = function() {
  this._alpha = 0;
  trace("clicked "+this._name);
};
```

Notice how you can still click myBtn2_btn after the alpha is set to 0.

See also

MovieClip._visible, TextField._visible

Button._width

Availability

Flash Player 6.

Usage

my_btn._width:*Number*

Description

Property; the width of the button, in pixels.

Example

The following example increases the width property of a button called `my_btn`, and displays the width in the Output panel. Enter the following ActionScript in Frame 1 of the Timeline:

```
my_btn.onRelease = function() {
  trace(this._width);
  this._width *= 1.1;
};
```

See also

MovieClip._width

Button._x

Availability

Flash Player 6.

Usage

my_btn._x:Number

Description

Property; an integer that sets the *x* coordinate of a button relative to the local coordinates of the parent movie clip. If a button is on the main Timeline, then its coordinate system refers to the upper left corner of the Stage as (0, 0). If the button is inside a movie clip that has transformations, the button is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90° counterclockwise, the enclosed button inherits a coordinate system that is rotated 90° counterclockwise. The button's coordinates refer to the registration point position.

Example

The following example sets the coordinates of `my_btn` to 0 on the Stage. Create a button called `my_btn` and enter the following ActionScript in Frame 1 of the Timeline:

```
my_btn._x = 0;
my_btn._y = 0;
```

See also

Button._xscale, Button._y, Button._yscale

Button._xmouse

Availability

Flash Player 6.

Usage

```
my_btn._xmouse:Number
```

Description

Read-only property; returns the *x* coordinate of the mouse position relative to the button.

Example

The following example displays the xmouse position for the Stage and a button called `my_btn` that is placed on the Stage. Enter the following ActionScript in Frame 1 of the Timeline:

```
this.createTextField("mouse_txt", 999, 5, 5, 150, 40);
mouse_txt.html = true;
mouse_txt.wordWrap = true;
mouse_txt.border = true;
mouse_txt.autoSize = true;
mouse_txt.selectable = false;
//
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
  var table_str:String = "<textformat tabstops='[50,100]'>";
  table_str += "<b>Stage</b>\t"+"x:"+_xmouse+"\t"+"y:"+_ymouse+newline;
  table_str += "<b>Button</
  b>\t"+"x:"+my_btn._xmouse+"\t"+"y:"+my_btn._ymouse+newline;
  table_str += "</textformat>";
  mouse_txt.htmlText = table_str;
};
Mouse.addListener(mouseListener);
```

See also

Button._ymouse

Button._xscale

Availability

Flash Player 6.

Usage

`my_btn._xscale:Number`

Description

Property; the horizontal scale of the button as applied from the registration point of the button, expressed as a percentage. The default registration point is (0,0).

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

Example

The following example scales a button called my_btn. When you click and release the button, it grows 10% on the *x* and *y* axis. Enter the following ActionScript in Frame 1 of the Timeline:

```
my_btn.onRelease = function(){
  this._xscale *= 1.1;
  this._yscale *= 1.1;
};
```

See also

Button._x, Button._y, Button._yscale

Button._y

Availability

Flash Player 6.

Usage

`my_btn._y:Number`

Description

Property; the *y* coordinate of the button relative to the local coordinates of the parent movie clip. If a button is in the main Timeline, its coordinate system refers to the upper left corner of the Stage as (0, 0). If the button is inside another movie clip that has transformations, the button is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90° counterclockwise, the enclosed button inherits a coordinate system that is rotated 90° counterclockwise. The button's coordinates refer to the registration point position.

Example

The following example sets the coordinates of `my_btn` to 0 on the Stage. Create a button called `my_btn` and enter the following ActionScript in Frame 1 of the Timeline:

```
my_btn._x = 0;
my_btn._y = 0;
```

See also

Button._x, Button._xscale, Button._yscale

Button._ymouse

Availability

Flash Player 6.

Usage

my_btn._ymouse:*Number*

Description

Read-only property; indicates the *y* coordinate of the mouse position relative to the button.

Example

The following example displays the xmouse position for the Stage and a button called `my_btn` that is placed on the Stage. Enter the following ActionScript in Frame 1 of the Timeline:

```
this.createTextField("mouse_txt", 999, 5, 5, 150, 40);
mouse_txt.html = true;
mouse_txt.wordWrap = true;
mouse_txt.border = true;
mouse_txt.autoSize = true;
mouse_txt.selectable = false;
//
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
  var table_str:String = "<textformat tabstops='[50,100]'>";
  table_str += "<b>Stage</b>\t"+"x:"+_xmouse+"\t"+"y:"+_ymouse+newline;
  table_str += "<b>Button</
  b>\t"+"x:"+my_btn._xmouse+"\t"+"y:"+my_btn._ymouse+newline;
  table_str += "</textformat>";
  mouse_txt.htmlText = table_str;
};
Mouse.addListener(mouseListener);
```

See also

Button._xmouse

Button._yscale

Availability

Flash Player 6.

Usage

my_btn._yscale:*Number*

Description

Property; the vertical scale of the button as applied from the registration point of the button, expressed as a percentage. The default registration point is (0,0).

Example

The following example scales a button called my_btn. When you click and release the button, it grows 10% on the *x* and *y* axis. Enter the following ActionScript in Frame 1 of the Timeline:

```
my_btn.onRelease = function(){
  this._xscale *= 1.1;
  this._yscale *= 1.1;
};
```

See also

Button._y, Button._x, Button._xscale

Camera class

Availability

Flash Player 6.

Description

The Camera class is primarily for use with Macromedia Flash Communication Server, but can be used in a limited way without the server.

The Camera class lets you capture video from a video camera attached to the computer that is running Macromedia Flash Player—for example, to monitor a video feed from a web camera attached to your local system. (Flash provides similar audio capabilities; for more information, see the Microphone class entry.)

Warning: When a SWF file tries to access the camera returned by Camera.get(), Flash Player displays a Privacy dialog box that lets the user choose whether to allow or deny access to the camera. (Make sure your Stage size is at least 215 x 138 pixels for the Camera class examples; this is the minimum size Flash requires to display the dialog box.)

To create or reference a Camera object, use Camera.get().

Method summary for the Camera class

Method	Description
`Camera.get()`	Returns a default or specified Camera object, or `null` if the camera is not available.
`Camera.setMode()`	Sets aspects of the camera capture mode, including height, width, and frames per second.
`Camera.setMotionLevel()`	Specifies how much motion is required to invoke `Camera.onActivity(true)` and how much time should elapse without motion before `Camera.onActivity(false)` is invoked.
`Camera.setQuality()`	Specifies the maximum amount of bandwidth that the current outgoing video feed can use, in bytes per second.

Property summary for the Camera class

Property (read-only)	Description
`Camera.activityLevel`	The amount of motion the camera is detecting.
`Camera.bandwidth`	The maximum amount of bandwidth the current outgoing video feed can use, in bytes.
`Camera.currentFps`	The rate at which the camera is capturing data, in frames per second.
`Camera.fps`	The rate at which you would like the camera to capture data, in frames per second.
`Camera.height`	The current capture height, in pixels.
`Camera.index`	The index of the camera, as reflected in the array returned by Camera.names.

Property (read-only)	Description
`Camera.motionLevel`	The amount of motion required to invoke `Camera.onActivity(true)`.
`Camera.motionTimeOut`	The number of milliseconds between the time when the camera stops detecting motion and the time `Camera.onActivity(false)` is invoked.
`Camera.muted`	A Boolean value that specifies whether the user has allowed or denied access to the camera.
`Camera.name`	The name of the camera as specified by the camera hardware.
`Camera.names`	Class property in an array of strings reflecting the names of all available video capture devices, including video cards and cameras.
`Camera.quality`	An integer specifying the required level of picture quality, as determined by the amount of compression being applied to each video frame.
`Camera.width`	The current capture width, in pixels.

Event handler summary for the Camera class

Event handler	Description
`Camera.onActivity`	Invoked when the camera starts or stops detecting motion.
`Camera.onStatus`	Invoked when the user allows or denies access to the camera.

Constructor for the Camera class

See `Camera.get()`.

Camera.activityLevel

Availability

Flash Player 6.

Usage

active_cam.activityLevel*:Number*

Description

Read-only property; a numeric value that specifies the amount of motion the camera is detecting. Values range from 0 (no motion is being detected) to 100 (a large amount of motion is being detected). The value of this property can help you determine if you need to pass a setting to `Camera.setMotionLevel()`.

If the camera is available but is not yet being used because Video.attachVideo() has not been called, this property is set to -1.

If you are streaming only uncompressed local video, this property is set only if you have assigned a function to the `Camera.onActivity` event handler. Otherwise, it is undefined.

Example

The following example detects the amount of motion the camera detects using the `activityLevel` property:

The following example detects the amount of motion the camera detects using the `activityLevel` property and a ProgressBar instance. Create a new video instance by selecting New Video from the Library options menu. Add an instance to the Stage and give it the instance name `my_video`. Add a ProgressBar component instance to the Stage and give it the instance name `activity_pb`. Then add the following ActionScript to Frame 1 of the Timeline:

```
// video instance on the Stage.
var my_video:Video;
var activity_pb:mx.controls.ProgressBar;
var my_cam:Camera = Camera.get();
my_video.attachVideo(my_cam);
activity_pb.mode = "manual";
activity_pb.label = "Activity %3%%";

this.onEnterFrame = function() {
  activity_pb.setProgress(my_cam.activityLevel, 100);
};
my_cam.onActivity = function(isActive:Boolean) {
  var themeColor:String = (isActive) ? "haloGreen" : "haloOrange";
  activity_pb.setStyle("themeColor", themeColor);
};
```

See also

Camera.motionLevel, Camera.setMotionLevel()

Camera.bandwidth

Availability

Flash Player 6.

Usage

active_cam.bandwidth:*Number*

Description

Read-only property; an integer that specifies the maximum amount of bandwidth the current outgoing video feed can use, in bytes. A value of 0 means that Flash video can use as much bandwidth as needed to maintain the desired frame quality.

To set this property, use `Camera.setQuality()`.

Example

The following example changes the maximum amount of bandwidth used by the camera feed. Create a new video instance by selecting New Video from the Library options menu. Add an instance to the Stage and give it the instance name `my_video`. Add a NumericStepper component instance to the Stage and give it the instance name `bandwidth_nstep`. Then add the following ActionScript to Frame 1 of the Timeline:

```
var bandwidth_nstep:mx.controls.NumericStepper;
var my_video:Video;
var my_cam:Camera = Camera.get();
my_video.attachVideo(my_cam);
this.createTextField("bandwidth_txt", this.getNextHighestDepth(), 0, 0, 100,
  22);
bandwidth_txt.autoSize = true;
this.onEnterFrame = function() {
  bandwidth_txt.text = "Camera is currently using "+my_cam.bandwidth+" bytes
  ("+Math.round(my_cam.bandwidth/1024)+" KB) bandwidth.";
};
//
bandwidth_nstep.minimum = 0;
bandwidth_nstep.maximum = 128;
bandwidth_nstep.stepSize = 16;
bandwidth_nstep.value = my_cam.bandwidth/1024;
function changeBandwidth(evt:Object) {
  my_cam.setQuality(evt.target.value*1024, 0);
}
bandwidth_nstep.addEventListener("change", changeBandwidth);
```

See also

Camera.setQuality()

Camera.currentFps

Availability

Flash Player 6.

Usage

active_cam.currentFps:*Number*

Description

Read-only property; the rate at which the camera is capturing data, in frames per second. This property cannot be set; however, you can use the `Camera.setMode()` method to set a related property—`Camera.fps`—which specifies the maximum frame rate at which you would like the camera to capture data.

Example

The following example detects the rate in frames per second that the camera captures data, using the `currentFps` property:

The following example detects the rate in frames per second that the camera captures data, using the `currentFps` property and a ProgressBar instance. Create a new video instance by selecting New Video from the Library options menu. Add an instance to the Stage and give it the instance name `my_video`. Add a ProgressBar component instance to the Stage and give it the instance name `fps_pb`. Then add the following ActionScript to Frame 1 of the Timeline:

```
var my_video:Video;
var fps_pb:mx.controls.ProgressBar;
var my_cam:Camera = Camera.get();
my_video.attachVideo(my_cam);
this.onEnterFrame = function() {
  fps_pb.setProgress(my_cam.fps-my_cam.currentFps, my_cam.fps);
};

fps_pb.setStyle("fontSize", 10);
fps_pb.setStyle("themeColor", "haloOrange");
fps_pb.labelPlacement = "top";
fps_pb.mode = "manual";
fps_pb.label = "FPS: %2 (%3%% dropped)";
```

See also

Camera.fps, Camera.setMode()

Camera.fps

Availability

Flash Player 6.

Usage

active_cam.fps:*Number*

Description

Read-only property; the maximum rate at which you want the camera to capture data, in frames per second. The maximum rate possible depends on the capabilities of the camera; that is, if the camera doesn't support the value you set here, this frame rate will not be achieved.

- To set a desired value for this property, use `Camera.setMode()`.
- To determine the rate at which the camera is currently capturing data, use the `Camera.currentFps` property.

Example

The following example detects the rate in frames per second that the camera captures data, using the `currentFps` property:

The following example detects the rate in frames per second that the camera captures data, using the `currentFps` property and a ProgressBar instance. Create a new video instance by selecting New Video from the Library options menu. Add an instance to the Stage and give it the instance name `my_video`. Add a ProgressBar component instance to the Stage and give it the instance name `fps_pb`. Then add the following ActionScript to Frame 1 of the Timeline:

```
var my_video:Video;
var fps_pb:mx.controls.ProgressBar;
var my_cam:Camera = Camera.get();
my_video.attachVideo(my_cam);
this.onEnterFrame = function() {
  fps_pb.setProgress(my_cam.fps-my_cam.currentFps, my_cam.fps);
};

fps_pb.setStyle("fontSize", 10);
fps_pb.setStyle("themeColor", "haloOrange");
fps_pb.labelPlacement = "top";
fps_pb.mode = "manual";
fps_pb.label = "FPS: %2 (%3%% dropped)";
```

Note: The `setMode()` function does not guarantee the requested fps setting; it sets the fps you requested or the fastest fps available.

See also

`Camera.currentFps`, `Camera.setMode()`

Camera.get()

Availability

Flash Player 6.

Usage

`Camera.get([index:Number]) : Camera`

Note: The correct syntax is `Camera.get()`. To assign the Camera object to a variable, use syntax like `active_cam = Camera.get()`.

Parameters

`index` An optional zero-based integer that specifies which camera to get, as determined from the array returned by the `Camera.names` property. To get the default camera (which is recommended for most applications), omit this parameter.

Returns

- If *`index`* is not specified, this method returns a reference to the default camera or, if it is in use by another application, to the first available camera. (If there is more than one camera installed, the user may specify the default camera in the Flash Player Camera Settings panel.) If no cameras are available or installed, the method returns `null`.
- If *`index`* is specified, this method returns a reference to the requested camera, or `null` if it is not available.

Description

Method; returns a reference to a Camera object for capturing video. To actually begin capturing the video, you must attach the Camera object to a Video object (see `Video.attachVideo()`).

Unlike objects that you create using the `new` constructor, multiple calls to `Camera.get()` reference the same camera. Thus, if your script contains the lines `first_cam = Camera.get()` and `second_cam = Camera.get()`, both `first_cam` and `second_cam` reference the same (default) camera.

In general, you shouldn't pass a value for *`index`*; simply use `Camera.get()` to return a reference to the default camera. By means of the Camera settings panel (discussed later in this section), the user can specify the default camera Flash should use. If you pass a value for *`index`*, you might be trying to reference a camera other than the one the user prefers. You might use *`index`* in rare cases—for example, if your application is capturing video from two cameras at the same time.

When a SWF file tries to access the camera returned by `Camera.get()`, Flash Player displays a Privacy dialog box that lets the user choose whether to allow or deny access to the camera. (Make sure your Stage size is at least 215 x 138 pixels; this is the minimum size Flash requires to display the dialog box.)

When the user responds to this dialog box, the `Camera.onStatus` event handler returns an information object that indicates the user's response. To determine whether the user has denied or allowed access to the camera without processing this event handler, use the `Camera.muted` property.

The user can also specify permanent privacy settings for a particular domain by right-clicking (Windows) or Control-clicking (Macintosh) while a SWF file is playing, selecting Settings, opening the Privacy panel, and selecting Remember.

You can't use ActionScript to set the Allow or Deny value for a user, but you can display the Privacy panel for the user by using `System.showSettings(0)`. If the user selects Remember, Flash Player no longer displays the Privacy dialog box for SWF files from this domain.

If `Camera.get` returns `null`, either the camera is in use by another application, or there are no cameras installed on the system. To determine whether any cameras are installed, use `Camera.names.length`. To display the Flash Player Camera Settings panel, which lets the user choose the camera to be referenced by `Camera.get()`, use `System.showSettings(3)`.

Scanning the hardware for cameras takes time. When Flash finds at least one camera, the hardware is not scanned again for the lifetime of the player instance. However, if Flash doesn't find any cameras, it will scan each time `Camera.get` is called. This is helpful if a user has forgotten to connect the camera; if your SWF file provides a Try Again button that calls `Camera.get`, Flash can find the camera without the user having to restart the SWF file.

Example

The following example lets you select an active camera to use from a ComboBox instance. The current active camera is displayed in a Label instance. Create a new video instance by selecting New Video from the Library options menu. Add an instance to the Stage and give it the instance name `my_video`. Add a Label component instance to the Stage and give it the instance name `camera_lbl`, and a ComboBox component instance and give it the instance name `cameras_cb`. Then add the following ActionScript to Frame 1 of the Timeline:

```
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);
var camera_lbl:mx.controls.Label;
var cameras_cb:mx.controls.ComboBox;
camera_lbl.text = my_cam.name;
cameras_cb.dataProvider = Camera.names;
function changeCamera():Void {
  my_cam = Camera.get(cameras_cb.selectedIndex);
  my_video.attachVideo(my_cam);
  camera_lbl.text = my_cam.name;
}
cameras_cb.addEventListener("change", changeCamera);
camera_lbl.setStyle("fontSize", 9);
cameras_cb.setStyle("fontSize", 9);
```

See also

Camera.index, Camera.muted, Camera.names, Camera.onStatus, Camera.setMode(), System.showSettings(), Video.attachVideo()

Camera.height

Availability

Flash Player 6.

Usage

active_cam.height:*Number*

Description

Read-only property; the current capture height, in pixels. To set a value for this property, use `Camera.setMode()`.

Example

The following code displays the current width, height and FPS of a video instance in a Label component instance on the Stage. Create a new video instance by selecting New Video from the Library options menu. Add an instance to the Stage and give it the instance name `my_video`. Add a Label component instance to the Stage and give it the instance name `dimensions_lbl`. Then add the following ActionScript to Frame 1 of the Timeline:

```
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);
var dimensions_lbl:mx.controls.Label;
dimensions_lbl.setStyle("fontSize", 9);
dimensions_lbl.setStyle("fontWeight", "bold");
dimensions_lbl.setStyle("textAlign", "center");
dimensions_lbl.text = "width: "+my_cam.width+", height: "+my_cam.height+",
  FPS: "+my_cam.fps;
```

See also the example for `Camera.setMode()`.

See also

Camera.setMode(), Camera.width

Camera.index

Availability

Flash Player 6.

Usage

active_cam.index:*Number*

Description

Read-only property; a zero-based integer that specifies the index of the camera, as reflected in the array returned by Camera.names.

Example

The following example displays an array of cameras in a text field that is created at runtime, and tells you which camera you are currently using. Create a new video instance by selecting New Video from the Library options menu. Add an instance to the Stage and give it the instance name my_video. Add a Label component instance to the Stage and give it the instance name camera_lbl. Then add the following ActionScript to Frame 1 of the Timeline:

```
var camera_lbl:mx.controls.Label;
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);

camera_lbl.text = my_cam.index+". "+my_cam.name;
this.createTextField("cameras_txt", this.getNextHighestDepth(), 25, 160, 160,
  80);
cameras_txt.html = true;
cameras_txt.border = true;
cameras_txt.wordWrap = true;
cameras_txt.multiline = true;
for (var i = 0; i<Camera.names.length; i++) {
  cameras_txt.htmlText += "<li><u><a
  href=\"asfunction:changeCamera,"+i+"\">"+Camera.names[i]+"</a></u></li>";
}
function changeCamera(index:Number) {
  my_cam = Camera.get(index);
  my_video.attachVideo(my_cam);
  camera_lbl.text = my_cam.index+". "+my_cam.name;
}
```

See also

Camera.get(), Camera.names

Camera.motionLevel

Availability

Flash Player 6.

Usage

active_cam.motionLevel:*Number*

Description

Read-only property; a numeric value that specifies the amount of motion required to invoke `Camera.onActivity(true)`. Acceptable values range from 0 to 100. The default value is 50.

Video can be displayed regardless of the value of the `motionLevel` property. For more information, see `Camera.setMotionLevel()`.

Example

The following example continually detects the motion level of a camera feed. Create a new video instance by selecting New Video from the Library options menu. Add an instance to the Stage and give it the instance name `my_video`. Add a Label component instance to the Stage and give it the instance name `motionLevel_lbl`, a NumericStepper with the instance name `motionLevel_nstep`, and a ProgressBar with the instance name `motion_pb`. Then add the following ActionScript to Frame 1 of the Timeline:

```
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);

// configure the ProgressBar component instance
var motion_pb:mx.controls.ProgressBar;
motion_pb.mode = "manual";
motion_pb.label = "Motion: %3%%";

var motionLevel_lbl:mx.controls.Label;
// configure the NumericStepper component instance
var motionLevel_nstep:mx.controls.NumericStepper;
motionLevel_nstep.minimum = 0;
motionLevel_nstep.maximum = 100;
motionLevel_nstep.stepSize = 5;
motionLevel_nstep.value = my_cam.motionLevel;

/* Continuously update the progress of the ProgressBar component instance to
  the activityLevel of the current Camera instance, which is defined in my_cam
  */
this.onEnterFrame = function() {
  motion_pb.setProgress(my_cam.activityLevel, 100);
};
```

```
/* When the level of activity changes goes above or below the number defined in
  Camera.motionLevel, trigger the onActivity event handler. If the
  activityLevel is above the value defined in the motionLevel property
  isActive will be set to true. Otherwise, the activityLevel has fallen below
  the value defined in the motionLevel property (and stayed below that level
  for 2 seconds), so set the isActive parameter to false. */
my_cam.onActivity = function(isActive:Boolean) {
  /* If isActive equals true, set the themeColor variable to "haloGreen".
     Otherwise set the themeColor to "haloOrange". */
  var themeColor:String = (isActive) ? "haloGreen" : "haloOrange";
  motion_pb.setStyle("themeColor", themeColor);
};

function changeMotionLevel() {
  /* Set the motionLevel property for my_cam Camera instance to the value of
  the NumericStepper component instance. Maintain the current motionTimeOut
  value of the my_cam Camera instance. */
  my_cam.setMotionLevel(motionLevel_nstep.value, my_cam.motionTimeOut);
}
motionLevel_nstep.addEventListener("change", changeMotionLevel);
```

See also

Camera.activityLevel, Camera.onActivity, Camera.onStatus, Camera.setMotionLevel()

Camera.motionTimeOut

Availability

Flash Player 6.

Usage

active_cam.motionTimeOut:*Number*

Description

Read-only property; the number of milliseconds between the time the camera stops detecting motion and the time Camera.onActivity(false) is invoked. The default value is 2000 (2 seconds).

To set this value, use Camera.setMotionLevel().

Example

In the following example, the ProgressBar instance changes its halo theme color when the activity level falls below the motion level. You can set the number of seconds for the motionTimeout property using a NumericStepper instance. Create a new video instance by selecting New Video from the Library options menu. Add an instance to the Stage and give it the instance name my_video. Add a Label component instance to the Stage and give it the instance name motionLevel_lbl, a NumericStepper with the instance name motionTimeOut_nstep, and a ProgressBar with the instance name motion_pb. Then add the following ActionScript to Frame 1 of the Timeline:

```
var motionLevel_lbl:mx.controls.Label;
var motion_pb:mx.controls.ProgressBar;
var motionTimeOut_nstep:mx.controls.NumericStepper;
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);

this.onEnterFrame = function() {
  motionLevel_lbl.text = "activityLevel: "+my_cam.activityLevel;
};

motion_pb.indeterminate = true;
my_cam.onActivity = function(isActive:Boolean) {
  if (isActive) {
    motion_pb.setStyle("themeColor", "haloGreen");
    motion_pb.label = "Motion is above "+my_cam.motionLevel;
  } else {
    motion_pb.setStyle("themeColor", "haloOrange");
    motion_pb.label = "Motion is below "+my_cam.motionLevel;
  }
};
function changeMotionTimeOut() {
  my_cam.setMotionLevel(my_cam.motionLevel, motionTimeOut_nstep.value*1000);
}
motionTimeOut_nstep.addEventListener("change", changeMotionTimeOut);
motionTimeOut_nstep.value = my_cam.motionTimeOut/1000;
```

See also

Camera.onActivity, Camera.setMotionLevel()

Camera.muted

Availability

Flash Player 6.

Usage

active_cam.muted:*Boolean*

Description

Read-only property; a Boolean value that specifies whether the user has denied access to the camera (`true`) or allowed access (`false`) in the Flash Player Privacy Settings panel. When this value changes, `Camera.onStatus` is invoked. For more information, see `Camera.get()`.

Example

In the following example, an error message could be displayed if `my_cam.muted` evaluates to true. Create a new video instance by selecting New Video from the Library options menu. Add an instance to the Stage and give it the instance name `my_video`. Then add the following ActionScript to Frame 1 of the Timeline:

```
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);
my_cam.onStatus = function(infoObj:Object) {
  if (my_cam.muted) {
    /* If user is denied access to their Camera, you can display an error
  message here. You can display the user's Camera/Privacy settings again using
  System.showSettings(0); */
    trace("User denied access to Camera");
    System.showSettings(0);
  }
};
```

See also

Camera.get(), Camera.onStatus

Camera.name

Availability

Flash Player 6.

Usage

active_cam.name:*String*

Description

Read-only property; a string that specifies the name of the current camera, as returned by the camera hardware.

Example

The following example displays the name of the default camera in a text field. In Windows, this name is the same as the device name listed in the Scanners and Cameras Control Panel. Create a new video instance by selecting New Video from the Library options menu. Add an instance to the Stage and give it the instance name my_video. Then add the following ActionScript to Frame 1 of the Timeline:

```
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);

this.createTextField("name_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
name_txt.autoSize = true;
name_txt.text = my_cam.name;
```

See also

Camera.get(), Camera.names

Camera.names

Availability

Flash Player 6.

Usage

```
Camera.names:Array
```

Note: The correct syntax is `Camera.names`. To assign the return value to a variable, use syntax like *`cam_array`* `= Camera.names`. To determine the name of the current camera, use *`active_cam`*`.name`.

Description

Read-only class property; retrieves an array of strings reflecting the names of all available cameras without displaying the Flash Player Privacy Settings panel. This array behaves in the same way as any other ActionScript array, implicitly providing the zero-based index of each camera and the number of cameras on the system (by means of `Camera.names.length`). For more information, see the Array class entry.

Calling the `Camera.names` property requires an extensive examination of the hardware, and it may take several seconds to build the array. In most cases, you can just use the default camera.

Example

The following example uses the default camera unless more than one camera is available, in which case the user can choose which camera to set as the default camera. If only one camera is present, then the default camera is used. Create a new video instance by selecting New Video from the Library options menu. Add an instance to the Stage and give it the instance name `my_video`. Then add the following ActionScript to Frame 1 of the Timeline:

```
var my_video:Video;
var cam_array:Array = Camera.names;
if (cam_array.length>1) {
  System.showSettings(3);
}
var my_cam:Camera = Camera.get();
my_video.attachVideo(my_cam);
```

See also

Camera.get(), Camera.index, Camera.name

Camera.onActivity

Availability

Flash Player 6.

Usage

```
active_cam.onActivity = function(activity:Boolean) : Void {
  // your statements here
}
```

Parameters

`activity` A Boolean value set to `true` when the camera starts detecting motion, `false` when it stops.

Returns

Nothing.

Description

Event handler; invoked when the camera starts or stops detecting motion. If you want to respond to this event handler, you must create a function to process its `activity` value.

To specify the amount of motion required to invoke `Camera.onActivity(true)` and the amount of time that must elapse without activity before invoking `Camera.onActivity(false)`, use `Camera.setMotionLevel()`.

Example

The following example displays `true` or `false` in the Output panel when the camera starts or stops detecting motion:

```
// Assumes a Video object named "myVideoObject" is on the Stage
active_cam = Camera.get();
myVideoObject.attachVideo(active_cam);
active_cam.setMotionLevel(10, 500);
active_cam.onActivity = function(mode)
{
  trace(mode);
}
```

See also

Camera.onActivity, Camera.setMotionLevel()

Camera.onStatus

Availability

Flash Player 6.

Usage

```
active_cam.onStatus = function(infoObject:Object) : Void {
  // your statements here
}
```

Parameters

`infoObject`	A parameter defined according to the status message.

Returns

Nothing.

Description

Event handler; invoked when the user allows or denies access to the camera. If you want to respond to this event handler, you must create a function to process the information object generated by the camera.

When a SWF file tries to access the camera, Flash Player displays a Privacy dialog box that lets the user choose whether to allow or deny access.

- If the user allows access, the `Camera.muted` property is set to `false`, and this handler is invoked with an information object whose `code` property is `"Camera.Unmuted"` and whose `level` property is `"Status"`.
- If the user denies access, the `Camera.muted` property is set to `true`, and this handler is invoked with an information object whose `code` property is `"Camera.Muted"` and whose `level` property is `"Status"`.

To determine whether the user has denied or allowed access to the camera without processing this event handler, use the `Camera.muted` property.

Note: If the user chooses to permanently allow or deny access for all SWF files from a specified domain, this handler is not invoked for SWF files from that domain unless the user later changes the privacy setting. For more information, see `Camera.get()`.

Example

The following ActionScript is used to display a message whenever the user allows or denies access to the camera:

```
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);
my_cam.onStatus = function(infoObj:Object) {
  switch (infoObj.code) {
  case 'Camera.Muted' :
    trace("Camera access is denied");
    break;
  case 'Camera.Unmuted' :
      trace("Camera access granted");
      break;
    }
  };
```

See also

Camera.get(), Camera.muted, System.showSettings(); System.onStatus

Camera.quality

Availability

Flash Player 6.

Usage

active_cam.quality:*Number*

Description

Read-only property; an integer specifying the required level of picture quality, as determined by the amount of compression being applied to each video frame. Acceptable quality values range from 1 (lowest quality, maximum compression) to 100 (highest quality, no compression). The default value is 0, which means that picture quality can vary as needed to avoid exceeding available bandwidth.

Example

The following example uses a NumericStepper instance to specify the amount of compression applied to the camera feed. Create a new video instance by selecting New Video from the Library options menu. Add an instance to the Stage and give it the instance name `my_video`. Add a NumericStepper with the instance name `quality_nstep`. Then add the following ActionScript to Frame 1 of the Timeline:

```
var quality_nstep:mx.controls.NumericStepper;

var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);

quality_nstep.minimum = 0;
quality_nstep.maximum = 100;
quality_nstep.stepSize = 5;
quality_nstep.value = my_cam.quality;

function changeQuality() {
  my_cam.setQuality(my_cam.bandwidth, quality_nstep.value);
}
quality_nstep.addEventListener("change", changeQuality);
```

See also

Camera.setQuality()

Camera.setMode()

Availability

Flash Player 6.

Usage

```
active_cam.setMode(width:Number, height:Number, fps:Number
  [,favorSize:Boolean]) : Void
```

Parameters

`width` The requested capture width, in pixels. The default value is 160.

`height` The requested capture height, in pixels. The default value is 120.

`fps` The requested rate at which the camera should capture data, in frames per second. The default value is 15.

`favorSize` A Boolean value that specifies how to manipulate the width, height, and frame rate if the camera does not have a native mode that meets the specified requirements. The default value is `true`, which means that maintaining capture size is favored; using this parameter selects the mode that most closely matches `width` and `height` values, even if doing so adversely affects performance by reducing the frame rate. To maximize frame rate at the expense of camera height and width, pass `false` for the `favorSize` parameter. This parameter is optional.

Returns

Nothing.

Description

Method; sets the camera capture mode to the native mode that best meets the specified requirements. If the camera does not have a native mode that matches all the parameters you pass, Flash selects a capture mode that most closely synthesizes the requested mode. This manipulation may involve cropping the image and dropping frames.

By default, Flash drops frames as needed to maintain image size. To minimize the number of dropped frames, even if this means reducing the size of the image, pass `false` for the `favorSize` parameter.

When choosing a native mode, Flash tries to maintain the requested aspect ratio whenever possible. For example, if you issue the command `active_cam.setMode(400, 400, 30)`, and the maximum width and height values available on the camera are 320 and 288, Flash sets both the width and height at 288; by setting these properties to the same value, Flash maintains the 1:1 aspect ratio you requested.

To determine the values assigned to these properties after Flash selects the mode that most closely matches your requested values, use `Camera.width`, `Camera.height`, and `Camera.fps`.

Example

The following example sets the camera capture mode. You can type a frame rate into a TextInput instance and press Enter or Return to apply the frame rate. Create a new video instance by selecting New Video from the Library options menu. Add an instance to the Stage and give it the instance name `my_video`. Add a TextInput component instance with the instance name `fps_ti`. Then add the following ActionScript to Frame 1 of the Timeline:

```
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);

fps_ti.maxChars = 2;
fps_ti.restrict = [0-9];
fps_lbl.text = "Current: "+my_cam.fps+" fps";

function changeFps():Void {
  my_cam.setMode(my_cam.width, my_cam.height, fps_ti.text);
  fps_lbl.text = "Current: "+my_cam.fps+" fps";
  fps_ti.text = my_cam.fps;
  Selection.setSelection(0,2);
}
fps_ti.addEventListener("enter", changeFps);
```

See also

Camera.currentFps, Camera.fps, Camera.height, Camera.width

Camera.setMotionLevel()

Availability

Flash Player 6.

Usage

```
active_cam.setMotionLevel(sensitivity:Number [, timeout:Number]) : Void
```

Parameters

`sensitivity` A numeric value that specifies the amount of motion required to invoke `Camera.onActivity(true)`. Acceptable values range from 0 to 100. The default value is 50.

`timeout` An optional numeric parameter that specifies how many milliseconds must elapse without activity before Flash considers activity to have stopped and invokes the `Camera.onActivity(false)` event handler. The default value is 2000 (2 seconds).

Returns

Nothing.

Description

Method; specifies how much motion is required to invoke `Camera.onActivity(true)`. Optionally sets the number of milliseconds that must elapse without activity before Flash considers motion to have stopped and invokes `Camera.onActivity(false)`.

Note: Video can be displayed regardless of the value of the *`sensitivity`* parameter. This parameter determines only when and under what circumstances Camera.onActivity is invoked–not whether video is actually being captured or displayed.

- To prevent the camera from detecting motion at all, pass a value of 100 for *`sensitivity`*; `Camera.onActivity` is never invoked. (You would probably use this value only for testing purposes—for example, to temporarily disable any actions set to occur when `Camera.onActivity` is invoked.)
- To determine the amount of motion the camera is currently detecting, use the Camera.activityLevel property.

Motion sensitivity values correspond directly to activity values. Complete lack of motion is an activity value of 0. Constant motion is an activity value of 100. Your activity value is less than your motion sensitivity value when you're not moving; when you are moving, activity values frequently exceed your motion sensitivity value.

This method is similar in purpose to `Microphone.setSilenceLevel()`; both methods are used to specify when the `onActivity` event handler should be invoked. However, these methods have a significantly different impact on publishing streams:

- `Microphone.setSilenceLevel()` is designed to optimize bandwidth. When an audio stream is considered silent, no audio data is sent. Instead, a single message is sent, indicating that silence has started.
- `Camera.setMotionLevel()` is designed to detect motion and does not affect bandwidth usage. Even if a video stream does not detect motion, video is still sent.

Example

The following example sends messages to the Output panel when video activity starts or stops. Change the motion sensitivity value of 30 to a higher or lower number to see how different values affect motion detection.

```
// Assumes a Video object named "myVideoObject" is on the Stage
active_cam = Camera.get();
x = 0;
function motion(mode) {
  trace(x + ": " + mode);
  x++;
}
active_cam.onActivity = function(mode) {
  motion(mode);
}
active_cam.setMotionLevel(30, 500);
myVideoObject.attachVideo(active_cam);
```

See also

Camera.activityLevel, Camera.motionLevel, Camera.motionTimeOut, Camera.onActivity

Camera.setQuality()

Availability

Flash Player 6.

Usage

```
active_cam.setQuality(bandwidth:Number, frameQuality:Number) : Void
```

Parameters

`bandwidth` An integer that specifies the maximum amount of bandwidth that the current outgoing video feed can use, in bytes per second. To specify that Flash video can use as much bandwidth as needed to maintain the value of `frameQuality`, pass 0 for `bandwidth`. The default value is 16384.

`frameQuality` An integer that specifies the required level of picture quality, as determined by the amount of compression being applied to each video frame. Acceptable values range from 1 (lowest quality, maximum compression) to 100 (highest quality, no compression). To specify that picture quality can vary as needed to avoid exceeding bandwidth, pass 0 for `frameQuality`. The default value is 0.

Returns

Nothing.

Description

Method; sets the maximum amount of bandwidth per second or the required picture quality of the current outgoing video feed. This method is generally applicable only if you are transmitting video using Flash Communication Server.

Use this method to specify which element of the outgoing video feed is more important to your application—bandwidth use or picture quality.

- To indicate that bandwidth use takes precedence, pass a value for `bandwidth` and 0 for `frameQuality`. Flash will transmit video at the highest quality possible within the specified bandwidth. If necessary, Flash will reduce picture quality to avoid exceeding the specified bandwidth. In general, as motion increases, quality decreases.
- To indicate that quality takes precedence, pass 0 for `bandwidth` and a numeric value for `frameQuality`. Flash will use as much bandwidth as required to maintain the specified quality. If necessary, Flash will reduce the frame rate to maintain picture quality. In general, as motion increases, bandwidth use also increases.
- To specify that both bandwidth and quality are equally important, pass numeric values for both parameters. Flash will transmit video that achieves the specified quality and that doesn't exceed the specified bandwidth. If necessary, Flash will reduce the frame rate to maintain picture quality without exceeding the specified bandwidth.

Example

The following examples illustrate how to use this method to control bandwidth use and picture quality.

```
// Ensure that no more than 8192 (8K/second) is used to send video
active_cam.setQuality(8192,0);

// Ensure that no more than 8192 (8K/second) is used to send video
// with a minimum quality of 50
active_cam.setQuality(8192,50);

// Ensure a minimum quality of 50, no matter how much bandwidth it takes
active_cam.setQuality(0,50);
```

See also

Camera.bandwidth, `Camera.get()`, Camera.quality

Camera.width

Availability

Flash Player 6.

Usage

active_cam.width:*Number*

Description

Read-only property; the current capture width, in pixels. To set a desired value for this property, use `Camera.setMode()`.

Example

The following code displays the current width, height and FPS of a video instance in a Label component instance on the Stage. Create a new video instance by selecting New Video from the Library options menu. Add an instance to the Stage and give it the instance name `my_video`. Add a Label component instance to the Stage and give it the instance name `dimensions_lbl`. Then add the following ActionScript to Frame 1 of the Timeline:

```
var my_cam:Camera = Camera.get();
var my_video:Video;
my_video.attachVideo(my_cam);
var dimensions_lbl:mx.controls.Label;
dimensions_lbl.setStyle("fontSize", 9);
dimensions_lbl.setStyle("fontWeight", "bold");
dimensions_lbl.setStyle("textAlign", "center");
dimensions_lbl.text = "width: "+my_cam.width+", height: "+my_cam.height+",
  FPS: "+my_cam.fps;
```

See also the example for `Camera.setMode()`.

See also

Camera.height, `Camera.setMode()`

case

Availability

Flash Player 4.

Usage

`case` *`expression`*`:` *`statement(s)`*

Parameters

`expression`	Any expression.
`statement(s)`	Any statement or sequence of statements.

Returns

Nothing.

Description

Statement; defines a condition for the `switch` statement. If the *`expression`* parameter equals the *`expression`* parameter of the `switch` statement using strict equality (===), then Flash Player will execute statements in the *`statement(s)`* parameter until it encounters a `break` statement or the end of the `switch` statement.

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

Note: You should always end the *`statement(s)`* parameter with a `break` statement. If you omit the `break` statement from the *`statement(s)`* parameter, it continues executing with the next `case` statement instead of exiting the `switch` statement.

Example

The following example defines conditions for the switch statement `thisMonth`. If `thisMonth` equals the expression in the case statement, the statement executes.

```
var thisMonth:Number = new Date().getMonth();
switch (thisMonth) {
case 0 :
  trace("January");
  break;
case 1 :
  trace("February");
  break;
case 5 :
case 6 :
case 7 :
  trace("Some summer month");
  break;
case 8 :
  trace("September");
  break;
default :
  trace("some other month");
}
```

See also

`break`, `default`, `=== (strict equality)`, `switch`

class

Availability

Flash Player 6.

Usage

```
[dynamic] class className  [ extends superClass ]
                [ implements interfaceName [, interfaceName... ] ]
{
  // class definition here
}
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files–not in scripts written in the Actions panel.

Parameters

`className` The fully qualified name of the class.

`superClass` The name of the class that `className` extends (inherits from). This parameter is optional.

`interfaceName` The name of the interface whose methods `className` must implement. This parameter is optional.

Description

Statement; defines a custom class, which lets you instantiate objects that share methods and properties that you define. For example, if you are developing an invoice-tracking system, you could create an invoice class that defines all the methods and properties that each invoice should have. You would then use the `new invoice()` command to create invoice objects.

The name of the class must match the name of the external file that contains the class. The name of the external file must be the name of the class with the file extension .as appended. For example, if you name a class Student, the file that defines the class must be named Student.as.

If a class is within a package, the class declaration must use the fully qualified class name of the form base.sub1.sub2.MyClass (for more information, see "Using packages" in *Using ActionScript in Flash*). Also, the class's AS file must be stored within the path in a directory structure that reflects the package structure, such as base/sub1/sub2/MyClass.as (for more information, see "Understanding the classpath" in *Using ActionsScript in Flash*). If a class definition is of the form "class MyClass," it is in the default package and the MyClass.as file should be in the top level of some directory in the path.

For this reason, it's good practice to plan your directory structure before you begin creating classes. Otherwise, if you decide to move class files after you create them, you have to modify the class declaration statements to reflect their new location.

You cannot nest class definitions; that is, you cannot define additional classes within a class definition.

To indicate that objects can add and access dynamic properties at runtime, precede the class statement with the `dynamic` keyword. To declare that a class implements an interface, use the `implements` keyword. To create subclasses of a class, use the `extends` keyword. (A class can extend only one class, but can implement several interfaces.) You can use `implements` and `extends` in a single statement. The following examples show typical uses of the `implements` and `extends` keywords:

```
class C implements Interface_i, Interface_j    // OK
class C extends Class_d implements Interface_i, Interface_j    // OK
class C extends Class_d, Class_e    // not OK
```

For more information, see in *Using ActionScript in Flash*.

Example

The following example creates a class called Plant. The Plant constructor takes two parameters.

```
// Filename Plant.as
class Plant {
  // Define property names and types
  var leafType:String;
  var bloomSeason:String;
  // Following line is constructor
  // because it has the same name as the class
  function Plant(param_leafType:String, param_bloomSeason:String) {
    // Assign passed values to properties when new Plant object is created
    this.leafType = param_leafType;
    this.bloomSeason = param_bloomSeason;
  }
  // Create methods to return property values, because best practice
  // recommends against directly referencing a property of a class
  function getLeafType():String {
    return leafType;
  }
  function getBloomSeason():String {
    return bloomSeason;
  }
}
```

In an external script file or in the Actions panel, use the `new` operator to create a Plant object.

```
var pineTree:Plant = new Plant("Evergreen", "N/A");
// Confirm parameters were passed correctly
trace(pineTree.getLeafType());
trace(pineTree.getBloomSeason());
```

The following example creates a class called ImageLoader. The `ImageLoader` constructor takes three parameters.

```
// Filename ImageLoader.as
class ImageLoader extends MovieClip {
  function ImageLoader(image:String, target_mc:MovieClip, init:Object) {
    var listenerObject:Object = new Object();
    listenerObject.onLoadInit = function(target) {
      for (var i in init) {
        target[i] = init[i];
      }
    };
    var JPEG_mcl:MovieClipLoader = new MovieClipLoader();
    JPEG_mcl.addListener(listenerObject);
    JPEG_mcl.loadClip(image, target_mc);
  }
}
```

In an external script file or in the Actions panel, use the `new` operator to create a ImageLoader object.

```
var jakob_mc:MovieClip = this.createEmptyMovieClip("jakob_mc",
  this.getNextHighestDepth());
var jakob:ImageLoader = new ImageLoader("http://www.macromedia.com/devnet/mx/
  blueprint/articles/nielsen/spotlight_jnielsen.jpg", jakob_mc, {_x:10, _y:10,
  _alpha:70, _rotation:-5});
```

See also

`dynamic`, `extends`, `implements`, `import`, `interface`, `new`, `Object.registerClass()`

clearInterval()

Availability

Flash Player 6.

Usage

```
clearInterval( intervalID:Number ) : Void
```

Parameters

intervalID A numeric (integer) identifier returned from a call to `setInterval()`.

Returns

Nothing.

Description

Function; cancels an interval created by a call to `setInterval()`.

Example

The following example first sets and then clears an interval call:

```
function callback() {
  trace("interval called: "+getTimer()+" ms.");
}
var intervalID:Number = setInterval(callback, 1000);
```

You need to clear the interval when you have finished using the function. Create a button called `clearInt_btn` and use the following ActionScript to clear `setInterval()`:

```
clearInt_btn.onRelease = function(){
  clearInterval( intervalID );
  trace("cleared interval");
};
```

See also

`setInterval()`

Color class

Availability

Flash Player 5.

Description

The Color class lets you set the RGB color value and color transform of movie clips and retrieve those values once they have been set.

You must use the constructor `new Color()` to create a Color object before calling its methods.

Method summary for the Color class

Method	Description
`Color.getRGB()`	Returns the numeric RGB value set by the last `setRGB()` call.
`Color.getTransform()`	Returns the transform information set by the last `setTransform()` call.
`Color.setRGB()`	Sets the hexadecimal representation of the RGB value for a Color object.
`Color.setTransform()`	Sets the color transform for a Color object.

Constructor for the Color class

Availability

Flash Player 5.

Usage

```
new Color(target_mc:Object) : Color
```

Parameters

`target_mc` The instance name of a movie clip.

Returns

A reference to a Color object.

Description

Constructor; creates a Color object for the movie clip specified by the *`target_mc`* parameter. You can then use the methods of that Color object to change the color of the entire target movie clip.

Example

The following example creates a Color object called `my_color` for the movie clip `my_mc` and sets its RGB value to orange:

```
var my_color:Color = new Color(my_mc);
my_color.setRGB(0xff9933);
```

Color.getRGB()

Availability

Flash Player 5.

Usage

```
my_color.getRGB() : Number
```

Parameters

None.

Returns

A number that represents the RGB numeric value for the color specified.

Description

Method; returns the numeric values set by the last `setRGB()` call.

Example

The following code retrieves the RGB value for the Color object `my_color`, converts the value to a hexadecimal string, and assigns it to the `myValue` variable. To see this code work, add a movie clip instance to the Stage, and give it the instance name `my_mc`:

```
var my_color:Color = new Color(my_mc);
// set the color
my_color.setRGB(0xff9933);
var myValue:String = my_color.getRGB().toString(16);
// trace the color value
trace(myValue);  // traces ff9933
```

See also

Color.setRGB()

Color.getTransform()

Availability

Flash Player 5.

Usage

```
my_color.getTransform() : Object
```

Parameters

None.

Returns

An object whose properties contain the current offset and percentage values for the specified color.

Description

Method; returns the transform value set by the last `Color.setTransform()` call.

Example

The following example gets the transform object, and then sets new percentages for colors and alpha of `my_mc` relative to their current values. To see this code work, place a multicolored movie clip on the Stage with the instance name `my_mc`. Then place the following code on Frame 1 in the main Timeline and select Control > Test Movie:

```
var my_color:Color = new Color(my_mc);
var myTransform:Object = my_color.getTransform();
myTransform = { ra: 50, ba: 50, aa: 30};
my_color.setTransform(myTransform);
```

For descriptions of the parameters for a color transform object, see Color.setTransform().

See also

Color.setTransform()

Color.setRGB()

Availability

Flash Player 5.

Usage

```
my_color.setRGB(0xRRGGBB:Number) : Void
```

Parameters

`0xRRGGBB` The hexadecimal or RGB color to be set. `RR`, `GG`, and `BB` each consist of two hexadecimal digits that specify the offset of each color component. The `0x` tells the ActionScript compiler that the number is a hexadecimal value.

Description

Method; specifies an RGB color for a Color object. Calling this method overrides any previous `Color.setTransform()` settings.

Returns

Nothing.

Example

This example sets the RGB color value for the movie clip `my_mc`. To see this code work, place a movie clip on the Stage with the instance name `my_mc`. Then place the following code on Frame 1 in the main Timeline and select Control > Test Movie:

```
var my_color:Color = new Color(my_mc);
my_color.setRGB(0xFF0000); // my_mc turns red
```

See also

Color.setTransform()

Color.setTransform()

Availability

Flash Player 5.

Usage

```
my_color.setTransform(colorTransformObject:Object) : Void
```

Parameters

`colorTransformObject` An object created with the `new Object` constructor. This instance of the Object class must have the following properties that specify color transform values: `ra`, `rb`, `ga`, `gb`, `ba`, `bb`, `aa`, `ab`. These properties are explained below.

Returns

Nothing.

Description

Method; sets color transform information for a Color object. The `colorTransformObject` parameter is a generic object that you create from the `new Object` constructor. It has parameters specifying the percentage and offset values for the red, green, blue, and alpha (transparency) components of a color, entered in the format *0xRRGGBBAA*.

The parameters for a color transform object correspond to the settings in the Advanced Effect dialog box and are defined as follows:

- `ra` is the percentage for the red component (-100 to 100).
- `rb` is the offset for the red component (-255 to 255).
- `ga` is the percentage for the green component (-100 to 100).
- `gb` is the offset for the green component (-255 to 255).
- `ba` is the percentage for the blue component (-100 to 100).
- `bb` is the offset for the blue component (-255 to 255).
- `aa` is the percentage for alpha (-100 to 100).
- `ab` is the offset for alpha (-255 to 255).

You create a `colorTransformObject` parameter as follows:

```
var myColorTransform:Object = new Object();
myColorTransform.ra = 50;
myColorTransform.rb = 244;
myColorTransform.ga = 40;
myColorTransform.gb = 112;
myColorTransform.ba = 12;
myColorTransform.bb = 90;
myColorTransform.aa = 40;
myColorTransform.ab = 70;
```

You can also use the following syntax to create a `colorTransformObject` parameter:

```
var myColorTransform:Object = { ra: 50, rb: 244, ga: 40, gb: 112, ba: 12, bb:
  90, aa: 40, ab: 70}
```

Example

This example creates a new Color object for a target SWF file, creates a generic object called `myColorTransform` with the properties defined above, and uses the `setTransform()` method to pass the *`colorTransformObject`* to a Color object. To use this code in a Flash (FLA) document, place it on Frame 1 on the main Timeline and place a movie clip on the Stage with the instance name `my_mc`, as in the following code:

```
// Create a color object called my_color for the target my_mc
var my_color:Color = new Color(my_mc);
// Create a color transform object called myColorTransform using
// Set the values for myColorTransform
var myColorTransform:Object = { ra: 50, rb: 244, ga: 40, gb: 112, ba: 12, bb:
  90, aa: 40, ab: 70};
// Associate the color transform object with the Color object
// created for my_mc
my_color.setTransform(myColorTransform);
```

ContextMenu class

Availability

Flash Player 7.

Description

The ContextMenu class provides runtime control over the items in the Flash Player context menu, which appears when a user right-clicks (Windows) or Control-clicks (Macintosh) on Flash Player. You can use the methods and properties of the ContextMenu class to add custom menu items, control the display of the built-in context menu items (for example, Zoom In and Print), or create copies of menus.

You can attach a ContextMenu object to a specific button, movie clip, or text field object, or to an entire movie level. You use the `menu` property of the Button, MovieClip, or TextField classes to do this. For more information about the `menu` property, see `Button.menu`, `MovieClip.menu`, and `TextField.menu`.

To add new items to a ContextMenu object, you create a ContextMenuItem object, and then add that object to the `ContextMenu.customItems` array. For more information about creating context menu items, see the ContextMenuItem class entry.

Flash Player has three types of context menus: the standard menu (which appears when you right-click in Flash Player), the edit menu (which appears when you right-click over a selectable or editable text field), and an error menu (which appears when a SWF file has failed to load into Flash Player.) Only the standard and edit menus can be modified with the ContextMenu class.

Custom menu items always appear at the top of the Flash Player context menu, above any visible built-in menu items; a separator bar distinguishes built-in and custom menu items. You can add no more than 15 custom items to a context menu. You cannot remove the Settings menu item from the context menu. The Settings menu item is required in Flash so users can access the settings that affect privacy and storage on their computers. You also cannot remove the About menu item from the context menu, which is required so users can find out what version of Flash Player they are using.

You must use the constructor `new ContextMenu()` to create a ContextMenu object before calling its methods.

Method summary for the ContextMenu class

Method	Description
`ContextMenu.copy()`	Returns a copy of the specified ContextMenu object.
`ContextMenu.hideBuiltInItems()`	Hides most built-in items in the Flash Player context menu.

Property summary for the ContextMenu class

Property	Description
`ContextMenu.builtInItems`	An object whose members correspond to built-in context menu items.
`ContextMenu.customItems`	An array, undefined by default, that contains ContextMenuItem objects.

Event handler summary for the ContextMenu class

Property	Description
`ContextMenu.onSelect`	Invoked before the menu is displayed.

Constructor for the ContextMenu class

Availability

Flash Player 7.

Usage

```
new ContextMenu ([callBackFunction])
```

Parameters

`callBackFunction` A reference to a function that is called when the user right-clicks or Control-clicks, before the menu is displayed. This parameter is optional.

Returns

A reference to a ContextMenu object.

Description

Constructor; creates a new ContextMenu object. You can optionally specify an identifier for an event handler when you create the object. The specified function is called when the user invokes the context menu, but *before* the menu is actually displayed. This is useful for customizing menu contents based on application state or based on the type of object (movie clip, text field, or button) or the Timeline that the user right-clicks or Control-clicks. (For an example of creating an event handler, see `ContextMenu.onSelect`.)

Example

The following example hides all the built-in objects in the Context menu. (However, the Settings and About items still appear, because they cannot be disabled.)

```
var newMenu:ContextMenu = new ContextMenu();
newMenu.hideBuiltInItems();
this.menu = newMenu;
```

In this example, the specified event handler, `menuHandler`, enables or disables a custom menu item (using the `ContextMenu.customItems` array) based on the value of a Boolean variable named `showItem`. If `false`, the custom menu item is disabled; otherwise, it's enabled.

```
var showItem = true;  // Change this to false to remove
var my_cm:ContextMenu = new ContextMenu(menuHandler);
my_cm.customItems.push(new ContextMenuItem("Hello", itemHandler));
function menuHandler(obj, menuObj) {
  if (showItem == false) {
    menuObj.customItems[0].enabled = false;
  } else {
    menuObj.customItems[0].enabled = true;
  }
}
function itemHandler(obj, item) {
  //...put code here...
  trace("selected!");
}
this.menu = my_cm;
```

When the user right-clicks or Control-clicks the Stage, the custom menu is displayed.

See also

`Button.menu`, `ContextMenu.onSelect`, `ContextMenu.customItems`, `ContextMenu.hideBuiltInItems()`, `MovieClip.menu`, `TextField.menu`

ContextMenu.builtInItems

Availability

Flash Player 7.

Usage

my_cm.builtInItems*:Object*

Description

Property; an object that has the following Boolean properties: `save`, `zoom`, `quality`, `play`, `loop`, `rewind`, `forward_back`, and `print`. Setting these variables to `false` removes the corresponding menu items from the specified ContextMenu object. These properties are enumerable and are set to `true` by default.

Example

In this example, the built-in Quality and Print menu items are disabled for the ContextMenu object `my_cm`, which is attached to the current Timeline of the SWF file.

```
var my_cm:ContextMenu = new ContextMenu ();
my_cm.builtInItems.quality=false;
my_cm.builtInItems.print=false;
this.menu = my_cm;
```

Note: You cannot disable the Settings or About menu items from the context menu.

In the next example, a `for..in` loop enumerates through all names and values of the built-in menu items of the ContextMenu object, `my_cm`.

```
var my_cm:ContextMenu = new ContextMenu();
for(eachProp in my_cm.builtInItems) {
  var propName = eachProp;
  var propValue = my_cm.builtInItems[propName];
  trace(propName + ": " + propValue);
}
```

ContextMenu.copy()

Availability

Flash Player 7.

Usage

```
my_cm.copy() : ContextMenu
```

Parameters

None.

Returns

A ContextMenu object.

Description

Method; creates a copy of the specified ContextMenu object. The copy inherits all the properties of the original menu object.

Example

This example creates a copy of the ContextMenu object named `my_cm` whose built-in menu items are hidden, and adds a menu item with the text "Save...". It then creates a copy of `my_cm` and assigns it to the variable `clone_cm`, which inherits all the properties of the original menu.

```
var my_cm:ContextMenu = new ContextMenu();
my_cm.hideBuiltInItems();
var menuItem_cmi:ContextMenuItem = new ContextMenuItem("Save...",
  saveHandler);
my_cm.customItems.push(menuItem_cmi);
function saveHandler(obj, menuItem) {
  // saveDocument();
  // custom function (not shown)
  trace("something");
}
clone_cm = my_cm.copy();
this.menu = my_cm;
for (var i in clone_cm.customItems) {
  trace("clone_cm-> "+clone_cm.customItems[i].caption);
}
for (var i in my_cm.customItems) {
  trace("my_cm-> "+my_cm.customItems[i].caption);
}
```

ContextMenu.customItems

Availability

Flash Player 7.

Usage

`my_cm.customItems:Array`

Description

Property; an array of ContextMenuItem objects. Each object in the array represents a context menu item that you have defined. Use this property to add, remove, or modify these custom menu items.

To add new menu items, you first create a new ContextMenuItem object, and then add it to the `menu_mc.customItems` array (for example, using Array.push()). For more information about creating new menu items, see the ContextMenuItem class entry.

Example

The following example creates a new custom menu item called `menuItem_cm` with a caption of "Send e-mail" and a callback handler named `emailHandler`. The new menu item is then added to the ContextMenu object, `my_cm`, using the `customItems` array. Finally, the new menu is attached to a movie clip named `email_mc`.

```
var my_cm:ContextMenu = new ContextMenu();
var menuItem_cmi:ContextMenuItem = new ContextMenuItem("Send e-mail",
  emailHandler);
my_cm.customItems.push(menuItem_cmi);
email_mc.menu = my_cm;
function emailHandler() {
  trace("sending email");
}
```

See also

Button.menu, ContextMenu class, MovieClip.menu, TextField.menu

ContextMenu.hideBuiltInItems()

Availability

Flash Player 7.

Usage

my_cm.hideBuiltInItems() *: Void*

Parameters

None.

Returns

Nothing.

Description

Method; hides all built-in menu items (except Settings) in the specified ContextMenu object. If the Flash Debug Player is running, the Debugging menu item shows, although it is dimmed for SWF files that don't have remote debugging enabled.

This method hides only menu items that appear in the standard context menu; it does not affect items that appear in the edit or error menus. For more information about the different menu types, see the ContextMenu class entry.

This method works by setting all the Boolean members of *my_cm*.builtInItems to false. You can selectively make a built-in item visible by setting its corresponding member in *my_cm*.builtInItems to true (as demonstrated in the following example).

Example

The following example creates a new ContextMenu object named my_cm whose built-in menu items are hidden, except for Print. The menu object is attached to the current Timeline.

```
var my_cm:ContextMenu = new ContextMenu();
my_cm.hideBuiltInItems();
my_cm.builtInItems.print = true;
this.menu = my_cm;
```

ContextMenu.onSelect

Availability

Flash Player 7.

Usage

```
my_cm.onSelect = function (item:Object, item_menu:ContextMenu) : Void{
  // your code here
}
```

Parameters

`item` A reference to the object (movie clip, button, or selectable text field) that was under the mouse pointer when the Flash Player context menu was invoked and whose `menu` property is set to a valid ContextMenu object.

`item_menu` A reference to the ContextMenu object assigned to the `menu` property of *`object`*.

Returns

Nothing.

Description

Event handler; called when a user invokes the Flash Player context menu, but before the menu is actually displayed. This lets you customize the contents of the context menu based on the current application state.

You can also specify the callback handler for a ContextMenu object when you construct a new ContextMenu object. For more information, see the ContextMenu class entry.

Example

The following example determines over what type of object the context menu was invoked:

```
my_cm = new ContextMenu();
function menuHandler(obj:Object, menu:ContextMenu) {
  if(obj instanceof MovieClip) {
    trace("Movie clip: " + obj);
  }
  if(obj instanceof TextField) {
    trace("Text field: " + obj);
  }
  if(obj instanceof Button) {
    trace("Button: " + obj);
  }
}
my_cm.onSelect = menuHandler;
my_mc.menu = my_cm;
my_btn.menu = my_cm;
```

ContextMenuItem class

Availability

Flash Player 7.

Description

You use the ContextMenuItem class to create custom menu items to display in the Flash Player context menu. Each ContextMenuItem object has a caption (text) that is displayed in the context menu, and a callback handler (a function) that is invoked when the menu item is selected. To add a new context menu item to a context menu, you add it to the `customItems` array of a ContextMenu object.

You can enable or disable specific menu items, make items visible or invisible, or change the caption or callback handler associated with a menu item.

Custom menu items appear at the top of the context menu, above any built-in items. A separator bar always divides custom menu items from built-in items. You can add no more than 15 custom items to a context menu. Each item must contain at least one visible character— control characters, newlines, and other white space characters are ignored. No item can be more than 100 characters long. Items that are identical to any built-in menu item, or to another custom item, are ignored, whether the matching item is visible or not. Menu items are compared without regard to case, punctuation, or white space.

None of the following words can appear in a custom item: *Macromedia, Flash Player,* or *Settings.*

Method summary for the ContextMenuItem class

Method	Description
`ContextMenuItem.copy()`	Returns a copy of the specified ContextMenuItem object.

Property summary for the ContextMenuItem class

Property	Description
`ContextMenuItem.caption`	Specifies the text displayed in the menu item.
`ContextMenuItem.enabled`	Specifies whether the menu item is enabled or disabled.
`ContextMenuItem.separatorBefore`	Specifies whether a separator bar should appear above the menu item.
`ContextMenuItem.visible`	Specifies whether the menu item is visible.

Event handler summary for the ContextMenuItem class

Event handler	Description
`ContextMenuItem.onSelect`	Invoked when the menu item is selected.

Constructor for the ContextMenuItem class

Availability

Flash Player 7.

Usage

```
new ContextMenuItem(caption:String,
                    callbackFunction:Function,
                    [ separatorBefore:Boolean,
                    [ enabled:Boolean,
                    [ visible:Boolean] ] ] ) : ContextMenuItem
```

Parameters

`caption` A string that specifies the text associated with the menu item.

`callbackFunction` A function that you define, which is called when the menu item is selected.

`separatorBefore` A Boolean value that indicates whether a separator bar should appear above the menu item in the context menu. The default value is `false`. This parameter is optional.

`enabled` A Boolean value that indicates whether the menu item is enabled or disabled in the context menu. The default value is `true`. This parameter is optional.

`visible` A Boolean value that indicates whether the menu item is visible or invisible. The default value is `true`. This parameter is optional.

Returns

A reference to a ContextMenuItem object.

Description

Constructor; creates a new ContextMenuItem object that can be added to the `ContextMenu.customItems` array.

Example

This example adds Start and Stop menu items, separated by a bar, to the ContextMenu object `my_cm`. The `startHandler()` function is called when Start is selected from the context menu; `stopHandler()` is called when Stop is selected. The ContextMenu object is applied to the current Timeline.

```
var my_cm:ContextMenu = new ContextMenu();
my_cm.customItems.push(new ContextMenuItem("Start", startHandler));
my_cm.customItems.push(new ContextMenuItem("Stop", stopHandler, true));
function stopHandler(obj, item) {
  trace("Stopping...");
}
function startHandler(obj, item) {
  trace("Starting...");
}
this.menu = my_cm;
```

ContextMenuItem.caption

Availability

Flash Player 7.

Usage

menuItem_cmi.caption:*String*

Description

Property; a string that specifies the menu item caption (text) displayed in the context menu.

Example

The following example displays the caption for the selected menu item (Pause Game) in the Output panel:

```
var my_cm:ContextMenu = new ContextMenu();
var menuItem_cmi:ContextMenuItem = new ContextMenuItem("Pause Game", onPause);
my_cm.customItems.push(menuItem_cmi);
function onPause(obj, menuItem) {
  trace("You chose: " + menuItem.caption);
}
this.menu = my_cm;
```

ContextMenuItem.copy()

Availability

Flash Player 7.

Usage

```
menuItem_cmi.copy() : ContextMenuItem
```

Returns

A ContextMenuItem object.

Description

Method; creates and returns a copy of the specified ContextMenuItem object. The copy includes all properties of the original object.

Example

This example creates a new ContextMenuItem object named `original_cmi` with the caption Pause and a callback handler set to the function `onPause`. The example then creates a copy of the ContextMenuItem object and assigns it to the variable `copy_cmi`.

```
var original_cmi:ContextMenuItem = new ContextMenuItem("Pause", onPause);
function onPause(obj:Object, menu:ContextMenu) {
  trace("pause me");
}

var copy_cmi:ContextMenuItem = original_cmi.copy();

var my_cm:ContextMenu = new ContextMenu();
my_cm.customItems.push(original_cmi);
my_cm.customItems.push(copy_cmi);

my_mc.menu = my_cm;
```

ContextMenuItem.enabled

Availability

Flash Player 7.

Usage

menuItem_cmi.enabled:*Boolean*

Description

Property; a Boolean value that indicates whether the specified menu item is enabled or disabled. By default, this property is `true`.

Example

The following example creates two new context menu items: Start and Stop. When the user selects Start, the number of milliseconds from when the SWF file started is traced. Start is then disabled in the menu. When Stop is selected, the number of milliseconds that have elapsed since the SWF file started is traced. The Start menu item is re-enabled and the Stop menu item is disabled.

```
var my_cm:ContextMenu = new ContextMenu();
var startMenuItem:ContextMenuItem = new ContextMenuItem("Start",
  startHandler);
startMenuItem.enabled = true;
my_cm.customItems.push(startMenuItem);
var stopMenuItem:ContextMenuItem = new ContextMenuItem("Stop", stopHandler,
  true);
stopMenuItem.enabled = false;
my_cm.customItems.push(stopMenuItem);
function stopHandler(obj, item) {
  trace("Stopping... "+getTimer()+"ms");
  startMenuItem.enabled = true;
  stopMenuItem.enabled = false;
}
function startHandler(obj, item) {
  trace("Starting... "+getTimer()+"ms");
  startMenuItem.enabled = false;
  stopMenuItem.enabled = true;
}
this.menu = my_cm;
```

ContextMenuItem.onSelect

Availability

Flash Player 7.

Usage

```
menuItem_cmi.onSelect = function (obj:Object, menuItem:ContextMenuItem) : Void
  {
  // your statements here
}
```

Parameters

obj A reference to the movie clip (or Timeline), button, or selectable text field that the user right-clicked or Control-clicked.

menuItem A reference to the selected ContextMenuItem object.

Returns

Nothing.

Description

Event handler; invoked when the specified menu item is selected from the Flash Player context menu. The specified callback handler receives two parameters: *obj*, a reference to the object under the mouse when the user invoked the Flash Player context menu, and *item*, a reference to the ContextMenuItem object that represents the selected menu item.

Example

The following example assigns a function to the `onSelect` handler for a ContextMenuItem object named `my_cmi`. The function displays the caption of the selected menu item.

```
var my_cmi:ContextMenu = new ContextMenu();
var start_cmi:ContextMenuItem = new ContextMenuItem("Start");
start_cmi.onSelect = function(obj, item) {
  trace("You chose: "+item.caption);
};
my_cmi.customItems.push(start_cmi);
my_cmi.customItems.push(new ContextMenuItem("Stop", stopHandler, true));
function stopHandler(obj, item) {
  trace("Stopping...");
}
this.menu = my_cmi;
```

See also

`ContextMenu.onSelect`

ContextMenuItem.separatorBefore

Availability

Flash Player 7.

Usage

menuItem_cmi.separatorBefore:*Boolean*

Description

Property; a Boolean value that indicates whether a separator bar should appear above the specified menu item. By default, this property is `false`.

Note: A separator bar always appears between any custom menu items and the built-in menu items.

Example

This example creates three menu items, labeled Open, Save, and Print. A separator bar divides the Save and Print items. The menu items are then added to the ContextMenu object's `customItems` array. Finally, the menu is attached to the current Timeline of the SWF file.

```
var my_cm:ContextMenu = new ContextMenu();
var open_cmi:ContextMenuItem = new ContextMenuItem("Open", itemHandler);
var save_cmi:ContextMenuItem = new ContextMenuItem("Save", itemHandler);
var print_cmi:ContextMenuItem = new ContextMenuItem("Print", itemHandler);
print_cmi.separatorBefore = true;
my_cm.customItems.push(open_cmi, save_cmi, print_cmi);
function itemHandler(obj, menuItem) {
    trace("You chose: " + menuItem.caption);
};
this.menu = my_cm;
```

See also

`ContextMenu.onSelect`

ContextMenuItem.visible

Availability

Flash Player 7.

Usage

```
menuItem_cmi.visible:Boolean
```

Description

Property; a Boolean value that indicates whether the specified menu item is visible when the Flash Player context menu is displayed. By default, this property is `true`.

Example

The following example creates two new context menu items: Start and Stop. When the user selects Start, the number of milliseconds from when the SWF file started is displayed. Start is then made invisible in the menu. When Stop is selected, the number of milliseconds that have elapsed since the SWF file started is displayed. The Start menu item is made visible and the Stop menu item is made invisible.

```
var my_cm:ContextMenu = new ContextMenu();
var startMenuItem:ContextMenuItem = new ContextMenuItem("Start",
  startHandler);
startMenuItem.visible = true;
my_cm.customItems.push(startMenuItem);
var stopMenuItem:ContextMenuItem = new ContextMenuItem("Stop", stopHandler,
  true);
stopMenuItem.visible = false;
my_cm.customItems.push(stopMenuItem);
function stopHandler(obj, item) {
  trace("Stopping... "+getTimer()+"ms");
  startMenuItem.visible = true;
  stopMenuItem.visible = false;
}
function startHandler(obj, item) {
  trace("Starting... "+getTimer()+"ms");
  startMenuItem.visible = false;
  stopMenuItem.visible = true;
}
this.menu = my_cm;
```

continue

Availability

Flash Player 4.

Usage

```
continue
```

Parameters

None.

Returns

Nothing.

Description

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

Example

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

```
trace("example 1");
var i:Number = 0;
while (i<10) {
  if (i%3 == 0) {
    i++;
    continue;
  }
  trace(i);
  i++;
}
```

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

```
trace("example 2");
var i:Number = 0;
do {
  if (i%3 == 0) {
    i++;
    continue;
  }
  trace(i);
  i++;
} while (i<10);
```

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

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

In the following `for..in` loop, `continue` causes the Flash interpreter to skip the rest of the loop body and jump back to the top of the loop, where the next value in the enumeration is processed:

```
for (i in _root) {
  if (i == "$version") {
    continue;
  }
  trace(i);
}
```

See also

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

CustomActions class

Availability

Flash Player 6.

Description

The methods of the CustomActions class allow a SWF file playing in the Flash authoring tool to manage any custom actions that are registered with the authoring tool. A SWF file can install and uninstall custom actions, retrieve the XML definition of a custom action, and retrieve the list of registered custom actions.

You can use these methods to build SWF files that are extensions of the Flash authoring tool. Such an extension could, for example, use the Flash Application Protocol to navigate a UDDI repository and download web services into the Actions toolbox.

Method summary for the CustomActions class

Method	Description
`CustomActions.get()`	Reads the contents of a custom action XML definition file.
`CustomActions.install()`	Installs a new custom action XML definition file.
`CustomActions.list()`	Returns a list of all registered custom actions.
`CustomActions.uninstall()`	Removes a custom action XML definition file.

CustomActions.get()

Availability

Flash Player 6.

Usage

`CustomActions.get(customName:String) : String`

Parameters

customName The name of the custom action definition to retrieve.

Returns

If the custom action XML definition is located, returns a string; otherwise, returns `undefined`.

Description

Method; reads the contents of the custom action XML definition file named *customName*.

The name of the definition file must be a simple filename, without the .xml file extension, and without any directory separators (':', '/' or '\').

If the definition file specified by the *customName* cannot be found, a value of `undefined` is returned. If the custom action XML definition specified by the *customName* parameter is located, it is read in its entirety and returned as a string.

Example

The following example lists the custom actions in a ComboBox instance, and gets the custom action when a Button instance is clicked. Drag an instance of a ComboBox, Button, and TextArea onto the Stage. Give the ComboBox an instance name of `customActionName_cb`, the TextArea an instance name of `customActionXml_ta`, and the Button an instance name of `view_button`. Enter the following ActionScript on Frame 1 of the Timeline:

```
import mx.controls.*;

var customActionName_cb:ComboBox;
var customActionXml_ta:TextArea;
var view_button:Button;

customActionName_cb.dataProvider = CustomActions.list();

customActionXml_ta.editable = false;

var viewListener:Object = new Object();
viewListener.click = function(evt:Object) {
  var caName:String = String(customActionName_cb.selectedItem);
  customActionXml_ta.text = CustomActions.get(caName);
};
view_button.addEventListener("click", viewListener);
```

CustomActions.install()

Availability

Flash Player 6.

Usage

```
CustomActions.install(customName:String, customXML:String) : Boolean
```

Parameters

`customName` The name of the custom action definition to install.

`customXML` The text of the XML definition to install.

Returns

A Boolean value of `false` if an error occurs during installation; otherwise, a value of `true` is returned to indicate that the custom action has been successfully installed.

Description

Method; installs a new custom action XML definition file indicated by the *`customName`* parameter. The contents of the file is specified by the string *`customXML`*.

The name of the definition file must be a simple filename, without the .xml file extension, and without any directory separators (':', '/' or '\').

If a custom actions file already exists with the name *`customName`*, it is overwritten.

If the Configuration/ActionsPanel/CustomActions directory does not exist when this method is invoked, the directory is created.

Example

The following example installs information into the Actions panel from an XML file. Open a text editor and save a new document called dogclass.xml. Enter the following code:

```
<?xml version="1.0"?>
<customactions>
  <actionspanel>
    <folder version="7" id="DogClass" index="true" name="Dog" tiptext="Dog
  Class">
      <string version="7" id="getFleas" name="getFleas" tiptext="gets number
  of fleas" text=".getFleas(% fleas %)" />
    </folder>
  </actionspanel>
  <colorsyntax>
      <identifier text=".getFleas" />
  </colorsyntax>
  <codehints>
      <typeinfo pattern="*_dog" object="Dog"/>
  </codehints>
</customactions>
```

Then open a new FLA file in the same directory and select Frame 1 of the Timeline. Enter the following code into the Actions panel:

```
var my_xml:XML = new XML();
my_xml.ignoreWhite = true;
my_xml.onLoad = function(success:Boolean) {
  trace(success);
  CustomActions.install("dogclass", this.firstChild);
  trace(CustomActions.list());
};
my_xml.load("dogclass.xml");
```

Select Control > Test Movie, and if the XML loads successfully, you will see `true`, and an array containing the names of all the custom actions that are registered with the Flash authoring tool in the Output panel. Close the SWF file, and open the Actions panel. You will see a new item in the Actions toolbox called Dog, and inside that folder you see getFleas.

CustomActions.list()

Availability

Flash Player 6.

Usage

`CustomActions.list() : Array`

Parameters

None.

Returns

An array.

Description

Method; returns an Array object containing the names of all the custom actions that are registered with the Flash authoring tool. The elements of the array are simple names, without the .xml file extension, and without any directory separators (for example, ":", "/", or "\"). If there are no registered custom actions, `list()` returns a zero-length array. If an error occurs, `list()` returns the value `undefined`.

Example

The following example lists the custom actions in a ComboBox instance, and gets the custom action when a Button instance is clicked. Drag an instance of a ComboBox, Button, and TextArea onto the Stage. Give the ComboBox an instance name of `customActionName_cb`, the TextArea an instance name of `customActionXml_ta`, and the Button an instance name of `view_button`. Enter the following ActionScript on Frame 1 of the Timeline:

```
import mx.controls.*;

var customActionName_cb:ComboBox;
var customActionXml_ta:TextArea;
var view_button:Button;

customActionName_cb.dataProvider = CustomActions.list();

customActionXml_ta.editable = false;

var viewListener:Object = new Object();
viewListener.click = function(evt:Object) {
  var caName:String = String(customActionName_cb.selectedItem);
  customActionXml_ta.text = CustomActions.get(caName);
};
view_button.addEventListener("click", viewListener);
```

CustomActions.uninstall()

Availability

Flash Player 6.

Usage

`CustomActions.uninstall(customName:String) : Boolean`

Parameters

customName	The name of the custom action definition to uninstall.

Returns

A Boolean value of `false` if no custom actions are found with the name *customName*. If the custom actions were successfully removed, a value of `true` is returned.

Description

Method; removes the Custom Actions XML definition file named *customName*.

The name of the definition file must be a simple filename, without the .xml file extension, and without any directory separators (':', '/' or '\').

Example

The following example installs a new custom action and displays an array containing the names of all the custom actions that are registered with the Flash authoring tool in the Output panel. When the uninstall_btn is clicked, the custom action is then uninstalled. An array containing names of the custom actions installed is displayed, and dogclass should then be removed from the array. Create a button called `uninstall_btn` and then enter the following ActionScript onto Frame 1 of the Timeline:

```
var my_xml:XML = new XML();
my_xml.ignoreWhite = true;
my_xml.onLoad = function(success:Boolean) {
  trace(success);
  CustomActions.install("dogclass", this.firstChild);
  trace(CustomActions.list());
};
my_xml.load("dogclass.xml");

uninstall_btn.onRelease = function() {
  CustomActions.uninstall("dogclass");
  trace(CustomActions.list());
};
```

For information on creating dogclass.xml, see `CustomActions.install()`.

Date class

Availability

Flash Player 5.

Description

The Date class lets you retrieve date and time values relative to universal time (Greenwich Mean Time, now called universal time or UTC) or relative to the operating system on which Flash Player is running. The methods of the Date class are not static but apply only to the individual Date object specified when the method is called. The Date.UTC() method is an exception; it is a static method.

The Date class handles daylight saving time differently, depending on the operating system and Flash Player version. Flash Player 6 and later versions handle daylight saving time on the following operating systems in these ways:

- Windows—the Date object automatically adjusts its output for daylight saving time. The Date object detects whether daylight saving time is employed in the current locale, and if so, it detects the standard-to-daylight saving time transition date and times. However, the transition dates currently in effect are applied to dates in the past and the future, so the daylight saving time bias might calculate incorrectly for dates in the past when the locale had different transition dates.
- Mac OS X—the Date object automatically adjusts its output for daylight saving time. The time zone information database in Mac OS X is used to determine whether any date or time in the present or past should have a daylight saving time bias applied.
- Mac OS 9—the operating system provides only enough information to determine whether the current date and time should have a daylight saving time bias applied. Accordingly, the date object assumes that the current daylight saving time bias applies to all dates and times in the past or future.

Flash Player 5 handles daylight saving time on the following operating systems as follows:

- Windows—the U.S. rules for daylight saving time are always applied, which leads to incorrect transitions in Europe and other areas that employ daylight saving time but have different transition times than the U.S. Flash correctly detects whether daylight saving time is used in the current locale.

To call the methods of the Date class, you must first create a Date object using the constructor for the Date class, described later in this section.

Method summary for the Date class

Method	Description
`Date.getDate()`	Returns the day of the month according to local time.
`Date.getDay()`	Returns the day of the week according to local time.
`Date.getFullYear()`	Returns the four-digit year according to local time.
`Date.getHours()`	Returns the hour according to local time.

Method	Description
`Date.getMilliseconds()`	Returns the milliseconds according to local time.
`Date.getMinutes()`	Returns the minutes according to local time.
`Date.getMonth()`	Returns the month according to local time.
`Date.getSeconds()`	Returns the seconds according to local time.
`Date.getTime()`	Returns the number of milliseconds since midnight January 1, 1970, universal time.
`Date.getTimezoneOffset()`	Returns the difference, in minutes, between the computer's local time and universal time.
`Date.getUTCDate()`	Returns the day of the month according to universal time.
`Date.getUTCDay()`	Returns the day of the week according to universal time.
`Date.getUTCFullYear()`	Returns the four-digit year according to universal time.
`Date.getUTCHours()`	Returns the hour according to universal time.
`Date.getUTCMilliseconds()`	Returns the milliseconds according to universal time.
`Date.getUTCMinutes()`	Returns the minutes according to universal time.
`Date.getUTCMonth()`	Returns the month according to universal time.
`Date.getUTCSeconds()`	Returns the seconds according to universal time.
`Date.getYear()`	Returns the year according to local time.
`Date.setDate()`	Sets the day of the month according to local time. Returns the new time in milliseconds.
`Date.setFullYear()`	Sets the full year according to local time. Returns the new time in milliseconds.
`Date.setHours()`	Sets the hour according to local time. Returns the new time in milliseconds.
`Date.setMilliseconds()`	Sets the milliseconds according to local time. Returns the new time in milliseconds.
`Date.setMinutes()`	Sets the minutes according to local time. Returns the new time in milliseconds.
`Date.setMonth()`	Sets the month according to local time. Returns the new time in milliseconds.
`Date.setSeconds()`	Sets the seconds according to local time. Returns the new time in milliseconds.
`Date.setTime()`	Sets the date in milliseconds. Returns the new time in milliseconds.
`Date.setUTCDate()`	Sets the date according to universal time. Returns the new time in milliseconds.
`Date.setUTCFullYear()`	Sets the year according to universal time. Returns the new time in milliseconds.

Method	Description
`Date.setUTCHours()`	Sets the hour according to universal time. Returns the new time in milliseconds.
`Date.setUTCMilliseconds()`	Sets the milliseconds according to universal time. Returns the new time in milliseconds.
`Date.setUTCMinutes()`	Sets the minutes according to universal time. Returns the new time in milliseconds.
`Date.setUTCMonth()`	Sets the month according to universal time. Returns the new time in milliseconds.
`Date.setUTCSeconds()`	Sets the seconds according to universal time. Returns the new time in milliseconds.
`Date.setYear()`	Sets the year according to local time.
`Date.toString()`	Returns a string value representing the date and time stored in the specified Date object.
`Date.UTC()`	Returns the number of milliseconds between midnight on January 1, 1970, universal time, and the specified time.

Constructor for the Date class

Availability

Flash Player 5.

Usage

```
new Date() : Date
new Date(timeValue:Number) : Date
new Date(year:Number, month:Number [, date:Number [, hour:Number [,
  minute:Number [, second:Number [, millisecond:Number ]]]]]) : Date
```

Parameters

year A value of 0 to 99 indicates 1900 through 1999; otherwise all four digits of the year must be specified.

month An integer from 0 (January) to 11 (December).

date An integer from 1 to 31. This parameter is optional.

hour An integer from 0 (midnight) to 23 (11 p.m.).

minute An integer from 0 to 59. This parameter is optional.

second An integer from 0 to 59. This parameter is optional.

millisecond An integer from 0 to 999. This parameter is optional.

Returns

A reference to a Date object.

Description

Constructor; constructs a new Date object that holds the specified date or the current date and time.

Example

The following example retrieves the current date and time:

```
var now_date:Date = new Date();
```

The following example creates a new Date object for Mary's birthday, August 12, 1974 (because the month parameter is zero-based, the example uses 7 for the month, not 8):

```
var maryBirthday:Date = new Date (74, 7, 12);
```

The following example creates a new Date object and concatenates the returned values of Date.getMonth(), Date.getDate(), and Date.getFullYear():

```
var today_date:Date = new Date();
var date_str:String = ((today_date.getMonth()+1)+"/"+today_date.getDate()+"/
  "+today_date.getFullYear());
trace(date_str); // displays current date in United States date format
```

Date.getDate()

Availability

Flash Player 5.

Usage

```
my_date.getDate() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the day of the month (an integer from 1 to 31) of the specified Date object according to local time. Local time is determined by the operating system on which Flash Player is running.

Example

The following example creates a new Date object and concatenates the returned values of Date.getMonth(), `Date.getDate()`, and Date.getFullYear():

```
var today_date:Date = new Date();
var date_str:String = (today_date.getDate()+"/"+(today_date.getMonth()+1)+"/
  "+today_date.getFullYear());
trace(date_str); // displays current date in United States date format
```

Date.getDay()

Availability

Flash Player 5.

Usage

```
my_date.getDay() : Number
```

Parameters

None.

Returns

An integer representing the day of the week.

Description

Method; returns the day of the week (0 for Sunday, 1 for Monday, and so on) of the specified Date object according to local time. Local time is determined by the operating system on which Flash Player is running.

Example

The following example creates a new Date object and uses `getDay()` to determine the current day of the week:

```
var dayOfWeek_array:Array = new Array("Sunday", "Monday", "Tuesday",
  "Wednesday", "Thursday", "Friday", "Saturday");
var today_date:Date = new Date();
var day_str:String = dayOfWeek_array[today_date.getDay()];
trace("Today is "+day_str);
```

Date.getFullYear()

Availability

Flash Player 5.

Usage

my_date.getFullYear() *: Number*

Parameters

None.

Returns

An integer representing the year.

Description

Method; returns the full year (a four-digit number, such as 2000) of the specified Date object, according to local time. Local time is determined by the operating system on which Flash Player is running.

Example

The following example uses the constructor to create a Date object. The `trace` statement shows the value returned by the `getFullYear()` method.

```
var my_date:Date = new Date();
trace(my_date.getYear()); // displays 104
trace(my_date.getFullYear()); // displays current year
```

Date.getHours()

Availability

Flash Player 5.

Usage

my_date.getHours() *: Number*

Parameters

None.

Returns

An integer.

Description

Method; returns the hour (an integer from 0 to 23) of the specified Date object, according to local time. Local time is determined by the operating system on which Flash Player is running.

Example

The following example uses the constructor to create a Date object based on the current time and uses the `getHours()` method to display hour values from that object:

```
var my_date:Date = new Date();
trace(my_date.getHours());

var my_date:Date = new Date();
var hourObj:Object = getHoursAmPm(my_date.getHours());
trace(hourObj.hours);
trace(hourObj.ampm);

function getHoursAmPm(hour24:Number):Object {
  var returnObj:Object = new Object();
  returnObj.ampm = (hour24<12) ? "AM" : "PM";
  var hour12:Number = hour24%12;
  if (hour12 == 0) {
    hour12 = 12;
  }
  returnObj.hours = hour12;
  return returnObj;
}
```

Date.getMilliseconds()

Availability

Flash Player 5.

Usage

```
my_date.getMilliseconds() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the milliseconds (an integer from 0 to 999) of the specified Date object, according to local time. Local time is determined by the operating system on which Flash Player is running.

Example

The following example uses the constructor to create a Date object based on the current time and uses the `getMilliseconds()` method to return the milliseconds value from that object:

```
var my_date:Date = new Date();
trace(my_date.getMilliseconds());
```

Date.getMinutes()

Availability

Flash Player 5.

Usage

```
my_date.getMinutes() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the minutes (an integer from 0 to 59) of the specified Date object, according to local time. Local time is determined by the operating system on which Flash Player is running.

Example

The following example uses the constructor to create a Date object based on the current time, and uses the `getMinutes()` method to return the minutes value from that object:

```
var my_date:Date = new Date();
trace(my_date.getMinutes());
```

Date.getMonth()

Availability

Flash Player 5.

Usage

```
my_date.getMonth() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the month (0 for January, 1 for February, and so on) of the specified Date object, according to local time. Local time is determined by the operating system on which Flash Player is running.

Example

The following example uses the constructor to create a Date object based on the current time and uses the `getMonth()` method to return the month value from that object:

```
var my_date:Date = new Date();
trace(my_date.getMonth());
```

The following example uses the constructor to create a Date object based on the current time and uses the `getMonth()` method to display the current month as a numeric value, and display the name of the month.

```
var my_date:Date = new Date();
trace(my_date.getMonth());
trace(getMonthAsString(my_date.getMonth()));
function getMonthAsString(month:Number):String {
  var monthNames_array:Array = new Array("January", "February", "March",
  "April", "May", "June", "July", "August", "September", "October",
  "November", "December");
  return monthNames_array[month];
}
```

Date.getSeconds()

Availability

Flash Player 5.

Usage

```
my_date.getSeconds() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the seconds (an integer from 0 to 59) of the specified Date object, according to local time. Local time is determined by the operating system on which Flash Player is running.

Example

The following example uses the constructor to create a Date object based on the current time and uses the `getSeconds()` method to return the seconds value from that object:

```
var my_date:Date = new Date();
trace(my_date.getSeconds());
```

Date.getTime()

Availability

Flash Player 5.

Usage

`my_date``.getTime()` *`: Number`*

Parameters

None.

Returns

An integer.

Description

Method; returns the number of milliseconds since midnight January 1, 1970, universal time, for the specified Date object. Use this method to represent a specific instant in time when comparing two or more Date objects.

Example

The following example uses the constructor to create a Date object based on the current time, and uses the `getTime()` method to return the number of milliseconds since midnight January 1, 1970:

```
var my_date:Date = new Date();
trace(my_date.getTime());
```

Date.getTimezoneOffset()

Availability

Flash Player 5.

Usage

```
my_date.getTimezoneOffset() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the difference, in minutes, between the computer's local time and universal time.

Example

The following example returns the difference between the local daylight saving time for San Francisco and universal time. Daylight saving time is factored into the returned result only if the date defined in the Date object occurs during daylight saving time.

```
var my_date:Date = new Date();
trace(my_date.getTimezoneOffset());
// 420 is displayed in the Output panel
// (7 hours * 60 minutes/hour = 420 minutes).
// This example is Pacific Daylight Time (PDT, GMT-0700).
// Result will vary depending on locale and time of year.
```

Date.getUTCDate()

Availability

Flash Player 5.

Usage

```
my_date.getUTCDate() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the day of the month (an integer from 1 to 31) in the specified Date object, according to universal time.

Example

The following example creates a new Date object and uses `Date.getUTCDate()` and Date.getDate(). The value returned by `Date.getUTCDate()` can differ from the value returned by `Date.getDate()`, depending on the relationship between your local time zone and universal time.

```
var my_date:Date = new Date(2004,8,25);
trace(my_date.getUTCDate()); // output: 25
```

Date.getUTCDay()

Availability

Flash Player 5.

Usage

my_date.getUTCDay() : *Number*

Parameters

None.

Returns

An integer.

Description

Method; returns the day of the week (0 for Sunday, 1 for Monday, and so on) of the specified Date object, according to universal time.

Example

The following example creates a new Date object and uses `Date.getUTCDay()` and Date.getDay(). The value returned by `Date.getUTCDay()` can differ from the value returned by `Date.getDay()`, depending on the relationship between your local time zone and universal time.

```
var today_date:Date = new Date();
trace(today_date.getDay()); // output will be based on local timezone
trace(today_date.getUTCDay()); // output will equal getDay() plus or minus one
```

Date.getUTCFullYear()

Availability

Flash Player 5.

Usage

```
my_date.getUTCFullYear() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the four-digit year of the specified Date object, according to universal time.

Example

The following example creates a new Date object and uses `Date.getUTCFullYear()` and Date.getFullYear(). The value returned by `Date.getUTCFullYear()` may differ from the value returned by `Date.getFullYear()` if today's date is December 31 or January 1 , depending on the relationship between your local time zone and universal time.

```
var today_date:Date = new Date();
trace(today_date.getFullYear()); // display based on local timezone
trace(today_date.getUTCFullYear()); // displays getYear() plus or minus 1
```

Date.getUTCHours()

Availability

Flash Player 5.

Usage

my_date.getUTCHours() : *Number*

Parameters

None.

Returns

An integer.

Description

Method; returns the hour (an integer from 0 to 23) of the specified Date object, according to universal time.

Example

The following example creates a new Date object and uses `Date.getUTCHours()` and Date.getHours(). The value returned by `Date.getUTCHours()` may differ from the value returned by `Date.getHours()`, depending on the relationship between your local time zone and universal time.

```
var today_date:Date = new Date();
trace(today_date.getHours()); // display based on local timezone
trace(today_date.getUTCHours()); // display equals getHours() plus or minus 12
```

Date.getUTCMilliseconds()

Availability

Flash Player 5.

Usage

```
my_date.getUTCMilliseconds() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the milliseconds (an integer from 0 to 999) of the specified Date object, according to universal time.

Example

The following example creates a new Date object and uses `getUTCMilliseconds()` to return the milliseconds value from the Date object.

```
var today_date:Date = new Date();
trace(today_date.getUTCMilliseconds());
```

Date.getUTCMinutes()

Availability

Flash Player 5.

Usage

```
my_date.getUTCMinutes() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the minutes (an integer from 0 to 59) of the specified Date object, according to universal time.

Example

The following example creates a new Date object and uses `getUTCMinutes()` to return the minutes value from the Date object:

```
var today_date:Date = new Date();
trace(today_date.getUTCMinutes());
```

Date.getUTCMonth()

Availability

Flash Player 5.

Usage

my_date.getUTCMonth() : *Number*

Parameters

None.

Returns

An integer.

Description

Method; returns the month (0 [January] to 11 [December]) of the specified Date object, according to universal time.

Example

The following example creates a new Date object and uses `Date.getUTCMonth()` and Date.getMonth(). The value returned by `Date.getUTCMonth()` can differ from the value returned by `Date.getMonth()` if today's date is the first or last day of a month, depending on the relationship between your local time zone and universal time.

```
var today_date:Date = new Date();
trace(today_date.getMonth()); // output based on local timezone
trace(today_date.getUTCMonth()); // output equals getMonth() plus or minus 1
```

Date.getUTCSeconds()

Availability

Flash Player 5.

Usage

my_date.getUTCSeconds() *: Number*

Parameters

None.

Returns

An integer.

Description

Method; returns the seconds (an integer from 0 to 59) of the specified Date object, according to universal time.

Example

The following example creates a new Date object and uses `getUTCSeconds()` to return the seconds value from the Date object:

```
var today_date:Date = new Date();
trace(today_date.getUTCSeconds());
```

Date.getYear()

Availability

Flash Player 5.

Usage

my_date.getYear() : *Number*

Parameters

None.

Returns

An integer.

Description

Method; returns the year of the specified Date object, according to local time. Local time is determined by the operating system on which Flash Player is running. The year is the full year minus 1900. For example, the year 2000 is represented as 100.

Example

The following example creates a Date object with the month and year set to May 2004. The `Date.getYear()` method returns 104, and Date.getFullYear() returns 2004:

```
var today_date:Date = new Date(2004,4);
trace(today_date.getYear()); // output: 104
trace(today_date.getFullYear()); // output: 2004
```

See also

`Date.getFullYear()`

Date.setDate()

Availability

Flash Player 5.

Usage

```
my_date.setDate(date) : Number
```

Parameters

`date` An integer from 1 to 31.

Returns

An integer.

Description

Method; sets the day of the month for the specified Date object, according to local time, and returns the new time in milliseconds. Local time is determined by the operating system on which Flash Player is running.

Example

The following example initially creates a new Date object, setting the date to May 15, 2004, and uses `Date.setDate()` to change the date to May 25, 2004:

```
var today_date:Date = new Date(2004,4,15);
trace(today_date.getDate()); //displays 15
today_date.setDate(25);
trace(today_date.getDate()); //displays 25
```

Date.setFullYear()

Availability

Flash Player 5.

Usage

```
my_date.setFullYear(year:Number [, month:Number [, date:Number]] ) : Number
```

Parameters

`year` A four-digit number specifying a year. Two-digit numbers do not represent four-digit years; for example, 99 is not the year 1999, but the year 99.

`month` An integer from 0 (January) to 11 (December). This parameter is optional.

`date` A number from 1 to 31. This parameter is optional.

Returns

An integer.

Description

Method; sets the year of the specified Date object, according to local time and returns the new time in milliseconds. If the `month` and `date` parameters are specified, they are set to local time. Local time is determined by the operating system on which Flash Player is running.

Calling this method does not modify the other fields of the specified Date object but Date.getUTCDay() and Date.getDay() can report a new value if the day of the week changes as a result of calling this method.

Example

The following example initially creates a new Date object, setting the date to May 15, 2004, and uses `Date.setFullYear()` to change the date to May 15, 2002:

```
var my_date:Date = new Date(2004,4,15);
trace(my_date.getFullYear()); //output: 2004
my_date.setFullYear(2002);
trace(my_date.getFullYear()); //output: 2002
```

Date.setHours()

Availability

Flash Player 5.

Usage

my_date.setHours(*hour*) *: Number*

Parameters

hour An integer from 0 (midnight) to 23 (11 p.m.).

Returns

An integer.

Description

Method; sets the hours for the specified Date object according to local time and returns the new time in milliseconds. Local time is determined by the operating system on which Flash Player is running.

Example

The following example initially creates a new Date object, setting the time and date to 8:00 a.m. on May 15, 2004, and uses `Date.setHours()` to change the time to 4:00 p.m.:

```
var my_date:Date = new Date(2004,4,15,8);
trace(my_date.getHours()); // output: 8
my_date.setHours(16);
trace(my_date.getHours()); // output: 16
```

Date.setMilliseconds()

Availability

Flash Player 5.

Usage

my_date.setMilliseconds(*millisecond:Number*) : *Number*

Parameters

millisecond An integer from 0 to 999.

Returns

An integer.

Description

Method; sets the milliseconds for the specified Date object according to local time and returns the new time in milliseconds. Local time is determined by the operating system on which Flash Player is running.

Example

The following example initially creates a new Date object, setting the date to 8:30 a.m. on May 15, 2004 with the milliseconds value set to 250, and then uses `Date.setMilliseconds()` to change the milliseconds value to 575:

```
var my_date:Date = new Date(2004,4,15,8,30,0,250);
trace(my_date.getMilliseconds()); // output: 250
my_date.setMilliseconds(575);
trace(my_date.getMilliseconds()); // output: 575
```

Date.setMinutes()

Availability

Flash Player 5.

Usage

my_date.`setMinutes(`*minute:Number*`)` *: Number*

Parameters

minute An integer from 0 to 59.

Returns

An integer.

Description

Method; sets the minutes for a specified Date object according to local time and returns the new time in milliseconds. Local time is determined by the operating system on which Flash Player is running.

Example

The following example initially creates a new Date object, setting the time and date to 8:00 a.m. on May 15, 2004, and then uses `Date.setMinutes()` to change the time to 8:30 a.m.:

```
var my_date:Date = new Date(2004,4,15,8,0);
trace(my_date.getMinutes()); // output: 0
my_date.setMinutes(30);
trace(my_date.getMinutes()); // output: 30
```

Date.setMonth()

Availability

Flash Player 5.

Usage

my_date.setMonth(*month:Number* [, *date:Number*]) *: Number*

Parameters

month An integer from 0 (January) to 11 (December).

date An integer from 1 to 31. This parameter is optional.

Returns

An integer.

Description

Method; sets the month for the specified Date object in local time and returns the new time in milliseconds. Local time is determined by the operating system on which Flash Player is running.

Example

The following example initially creates a new Date object, setting the date to May 15, 2004, and uses `Date.setMonth()` to change the date to June 15, 2004:

```
var my_date:Date = new Date(2004,4,15);
trace(my_date.getMonth()); //output: 4
my_date.setMonth(5);
trace(my_date.getMonth()); //output: 5
```

Date.setSeconds()

Availability

Flash Player 5.

Usage

`my_date.setSeconds(second:Number) : Number`

Parameters

`second` An integer from 0 to 59.

Returns

An integer.

Description

Method; sets the seconds for the specified Date object in local time and returns the new time in milliseconds. Local time is determined by the operating system on which Flash Player is running.

Example

The following example initially creates a new Date object, setting the time and date to 8:00:00 a.m. on May 15, 2004, and uses `Date.setSeconds()` to change the time to 8:00:45 a.m.:

```
var my_date:Date = new Date(2004,4,15,8,0,0);
trace(my_date.getSeconds()); // output: 0
my_date.setSeconds(45);
trace(my_date.getSeconds()); // output: 45
```

Date.setTime()

Availability

Flash Player 5.

Usage

`my_date``.setTime(`*`milliseconds:Number`*`) : Number`

Parameters

`milliseconds` A number; an integer value where 0 is midnight on January 1, universal time.

Returns

An integer.

Description

Method; sets the date for the specified Date object in milliseconds since midnight on January 1, 1970, and returns the new time in milliseconds.

Example

The following example initially creates a new Date object, setting the time and date to 8:00 a.m. on May 15, 2004, and uses `Date.setTime()` to change the time to 8:30 a.m.:

```
var my_date:Date = new Date(2004,4,15,8,0,0);
var myDate_num:Number = my_date.getTime(); // convert my_date to milliseconds
myDate_num += 30 * 60 * 1000; // add 30 minutes in milliseconds
my_date.setTime(myDate_num); // set my_date Date object 30 minutes forward
trace(my_date.getFullYear()); // output: 2004
trace(my_date.getMonth()); // output: 4
trace(my_date.getDate()); // output: 15
trace(my_date.getHours()); // output: 8
trace(my_date.getMinutes()); // output: 30
```

Date.setUTCDate()

Availability

Flash Player 5.

Usage

my_date.setUTCDate(*date:Number*) : *Number*

Parameters

date A number; an integer from 1 to 31.

Returns

An integer.

Description

Method; sets the date for the specified Date object in universal time and returns the new time in milliseconds. Calling this method does not modify the other fields of the specified Date object, but Date.getUTCDay() and Date.getDay() can report a new value if the day of the week changes as a result of calling this method.

Example

The following example initially creates a new Date object with today's date, uses `Date.setUTCDate()` to change the date value to 10, and changes it again to 25:

```
var my_date:Date = new Date();
my_date.setUTCDate(10);
trace(my_date.getUTCDate()); // output: 10
my_date.setUTCDate(25);
trace(my_date.getUTCDate()); // output: 25
```

Date.setUTCFullYear()

Availability

Flash Player 5.

Usage

`my_date``.setUTCFullYear(`*`year:Number`*` [, `*`month:Number`*` [, `*`date:Number`*`]]) : `*`Number`*

Parameters

`year` An integer that represents the year specified as a full four-digit year, such as 2000.

`month` An integer from 0 (January) to 11 (December). This parameter is optional.

`date` An integer from 1 to 31. This parameter is optional.

Returns

An integer.

Description

Method; sets the year for the specified Date object (*`my_date`*) in universal time and returns the new time in milliseconds.

Optionally, this method can also set the month and date represented by the specified Date object. Calling this method does not modify the other fields of the specified Date object, but Date.getUTCDay() and Date.getDay() can report a new value if the day of the week changes as a result of calling this method.

Example

The following example initially creates a new Date object with today's date, uses `Date.setUTCFullYear()` to change the year value to 2001, and changes the date to May 25, 1995:

```
var my_date:Date = new Date();
my_date.setUTCFullYear(2001);
trace(my_date.getUTCFullYear()); // output: 2001
my_date.setUTCFullYear(1995, 4, 25);
trace(my_date.getUTCFullYear()); // output: 1995
trace(my_date.getUTCMonth()); // output: 4
trace(my_date.getUTCDate()); // output: 25
```

Date.setUTCHours()

Availability

Flash Player 5.

Usage

```
my_date.setUTCHours(hour:Number [, minute:Number [, second:Number [,
  millisecond:Number]]]) : Number
```

Parameters

`hour` A number; an integer from 0 (midnight) to 23 (11 p.m.).

`minute` A number; an integer from 0 to 59. This parameter is optional.

`second` A number; an integer from 0 to 59. This parameter is optional.

`millisecond` A number; an integer from 0 to 999. This parameter is optional.

Returns

An integer.

Description

Method; sets the hour for the specified Date object in universal time and returns the new time in milliseconds.

Example

The following example initially creates a new Date object with today's date, uses `Date.setUTCHours()` to change the time to 8:30 a.m., and changes the time again to 5:30:47 p.m.:

```
var my_date:Date = new Date();
my_date.setUTCHours(8,30);
trace(my_date.getUTCHours()); // output: 8
trace(my_date.getUTCMinutes()); // output: 30
my_date.setUTCHours(17,30,47);
trace(my_date.getUTCHours()); // output: 17
trace(my_date.getUTCMinutes()); // output: 30
trace(my_date.getUTCSeconds()); // output: 47
```

Date.setUTCMilliseconds()

Availability

Flash Player 5.

Usage

my_date.setUTCMilliseconds(*millisecond:Number*) *: Number*

Parameters

millisecond An integer from 0 to 999.

Returns

An integer.

Description

Method; sets the milliseconds for the specified Date object in universal time and returns the new time in milliseconds.

Example

The following example initially creates a new Date object, setting the date to 8:30 a.m. on May 15, 2004 with the milliseconds value set to 250, and uses `Date.setUTCMilliseconds()` to change the milliseconds value to 575:

```
var my_date:Date = new Date(2004,4,15,8,30,0,250);
trace(my_date.getUTCMilliseconds()); // output: 250
my_date.setUTCMilliseconds(575);
trace(my_date.getUTCMilliseconds()); // output: 575
```

Date.setUTCMinutes()

Availability

Flash Player 5.

Usage

```
my_date.setUTCMinutes(minute:Number [, second:Number [, millisecond:Number]])
  : Number
```

Parameters

`minute` An integer from 0 to 59.

`second` An integer from 0 to 59. This parameter is optional.

`millisecond` An integer from 0 to 999. This parameter is optional.

Returns

An integer.

Description

Method; sets the minute for the specified Date object in universal time and returns the new time in milliseconds.

Example

The following example initially creates a new Date object, setting the time and date to 8:00 a.m. on May 15, 2004, and uses `Date.setUTCMinutes()` to change the time to 8:30 a.m.:

```
var my_date:Date = new Date(2004,4,15,8,0);
trace(my_date.getUTCMinutes()); // output: 0
my_date.setUTCMinutes(30);
trace(my_date.getUTCMinutes()); // output: 30
```

Date.setUTCMonth()

Availability

Flash Player 5.

Usage

```
my_date.setUTCMonth(month:Number [, date:Number]) : Number
```

Parameters

`month` An integer from 0 (January) to 11 (December).

`date` An integer from 1 to 31. This parameter is optional.

Returns

An integer.

Description

Method; sets the month, and optionally the day, for the specified Date object in universal time and returns the new time in milliseconds. Calling this method does not modify the other fields of the specified Date object, but Date.getUTCDay() and Date.getDay() might report a new value if the day of the week changes as a result of specifying a value for the `date` parameter.

Example

The following example initially creates a new Date object, setting the date to May 15, 2004, and uses `Date.setMonth()` to change the date to June 15, 2004:

```
var today_date:Date = new Date(2004,4,15);
trace(today_date.getUTCMonth()); // output: 4
today_date.setUTCMonth(5);
trace(today_date.getUTCMonth()); // output: 5
```

Date.setUTCSeconds()

Availability

Flash Player 5.

Usage

```
my_date.setUTCSeconds(second:Number [, millisecond:Number]) : Number
```

Parameters

second An integer from 0 to 59.

millisecond An integer from 0 to 999. This parameter is optional.

Returns

An integer.

Description

Method; sets the seconds for the specified Date object in universal time and returns the new time in milliseconds.

Example

The following example initially creates a new Date object, setting the time and date to 8:00:00 a.m. on May 15, 2004, and uses `Date.setSeconds()` to change the time to 8:30:45 a.m.:

```
var my_date:Date = new Date(2004,4,15,8,0,0);
trace(my_date.getUTCSeconds()); // output: 0
my_date.setUTCSeconds(45);
trace(my_date.getUTCSeconds()); // output: 45
```

Date.setYear()

Availability

Flash Player 5.

Usage

my_date.setYear(*year:Number*) : *Number*

Parameters

year A number that represents the year. If *year* is an integer between 0–99, setYear sets the year at 1900 + *year*; otherwise, the year is the value of the year parameter.

Returns

An integer.

Description

Method; sets the year for the specified Date object in local time and returns the new time in milliseconds. Local time is determined by the operating system on which Flash Player is running.

Example

The following example creates a new Date object with the date set to May 25, 2004, uses setYear() to change the year to 1999, and changes the year to 2003:

```
var my_date:Date = new Date(2004,4,25);
trace(my_date.getYear()); // output: 104
trace(my_date.getFullYear()); // output: 2004
my_date.setYear(99);
trace(my_date.getYear()); // output: 99
trace(my_date.getFullYear()); // output: 1999
my_date.setYear(2003);
trace(my_date.getYear()); // output: 103
trace(my_date.getFullYear()); // output: 2003
```

Date.toString()

Availability

Flash Player 5.

Usage

```
my_date.toString() : String
```

Parameters

None.

Returns

A string.

Description

Method; returns a string value for the specified date object in a readable format.

Example

The following example returns the information in the `dateOfBirth_date` Date object as a string:

```
var dateOfBirth_date:Date = new Date(74, 7, 12, 18, 15);
trace (dateOfBirth_date);
trace (dateOfBirth_date.toString());
/*
The output from the trace statements will be in local time and will vary 
accordingly. For Pacific Daylight Time the output will be seven hours 
earlier than universal time:
Mon Aug 12 18:15:00 GMT-0700 1974
Mon Aug 12 18:15:00 GMT-0700 1974
*/
```

Date.UTC()

Availability

Flash Player 5.

Usage

```
Date.UTC(year:Number, month:Number [, date:Number [, hour:Number [,
  minute:Number [, second:Number [, millisecond:Number ]]]]]) : Number
```

Parameters

year A four-digit integer that represents the year (for example, 2000).

month An integer from 0 (January) to 11 (December).

date An integer from 1 to 31. This parameter is optional.

hour An integer from 0 (midnight) to 23 (11 p.m.).

minute An integer from 0 to 59. This parameter is optional.

second An integer from 0 to 59. This parameter is optional.

millisecond An integer from 0 to 999. This parameter is optional.

Returns

An integer.

Description

Method; returns the number of milliseconds between midnight on January 1, 1970, universal time, and the time specified in the parameters. This is a static method that is invoked through the Date object constructor, not through a specific Date object. This method lets you create a Date object that assumes universal time, whereas the Date constructor assumes local time.

Example

The following example creates a new `maryBirthday_date` Date object defined in universal time. This is the universal time variation of the example used for the `new Date` constructor method.

```
var maryBirthday_date:Date = new Date(Date.UTC(1974, 7, 12));
trace(maryBirthday_date);
// output will be in local time and will vary accordingly
// for Pacific Daylight Time the output will be seven hours earlier than UTC:
// Sun Aug 11 17:00:00 GMT-0700 1974
```

default

Availability

Flash Player 6.

Usage

`default: statements`

Parameters

statements Any statements.

Returns

Nothing.

Description

Statement; defines the default case for a `switch` statement. The statements execute if the *expression* parameter of the `switch` statement doesn't equal (using the strict equality [===] operation) any of the *expression* parameters that follow the `case` keywords for a given `switch` statement.

A `switch` is not required to have a `default` case statement. A `default` case statement does not have to be last in the list. If you use a `default` statement outside a `switch` statement, it produces an error and the script doesn't compile.

Example

In the following example, the expression A does not equal the expressions B or D, so the statement following the `default` keyword is run and the `trace()` statement is sent to the Output panel.

```
var dayOfWeek:Number = new Date().getDay();
switch (dayOfWeek) {
case 1 :
  trace("Monday");
  break;
case 2 :
  trace("Tuesday");
  break;
case 3 :
  trace("Wednesday");
  break;
case 4 :
  trace("Thursday");
  break;
case 5 :
  trace("Friday");
  break;
default :
  trace("Weekend");
}
```

See also

`switch`, `case`, `break`

delete

Availability

Flash Player 5.

Usage

```
delete reference
```

Parameters

reference The name of the variable or object to eliminate.

Returns

A Boolean value.

Description

Operator; destroys the object reference specified by the *reference* parameter, and returns `true` if the reference is successfully deleted; `false` otherwise. This operator is useful for freeing memory used by scripts. You can use the `delete` operator to remove references to objects. After all references to an object are removed, Flash Player takes care of removing the object and freeing the memory used by that object.

Although `delete` is an operator, it is typically used as a statement, as shown in the following example:

```
delete x;
```

The `delete` operator can fail and return `false` if the *reference* parameter does not exist or cannot be deleted. Predefined objects and properties, and variables declared with `var`, cannot be deleted. You cannot use the `delete` operator to remove movie clips.

Example

Usage 1: The following example creates an object, uses it, and deletes it after it is no longer needed:

```
var account:Object = new Object();
account.name = "Jon";
account.balance = 10000;
trace(account.name);
delete account;
trace(account.name);  //output: Jon undefined
```

Usage 2: The following example deletes a property of an object:

```
// create the new object "account"
var account:Object = new Object();
// assign property name to the account
account.name = "Jon";
// delete the property
delete account.name;
```

Usage 3: The following example deletes an object property:

```
var my_array:Array = new Array();
my_array[0] = "abc"; // my_array.length == 1
my_array[1] = "def"; // my_array.length == 2
my_array[2] = "ghi"; // my_array.length == 3
// my_array[2] is deleted, but Array.length is not changed
delete my_array[2];
trace(my_array.length); // output: 3
trace(my_array); // output: abc,def,undefined
```

Usage 4: The following example shows the behavior of `delete` on object references:

```
var ref1:Object = new Object();
ref1.name = "Jody";
// copy the reference variable into a new variable
// and delete ref1
ref2 = ref1;
delete ref1;
trace("ref1.name "+ref1.name); //output: undefined
trace("ref2.name "+ref2.name); //output: Jody
```

If `ref1` had not been copied into `ref2`, the object would have been deleted when `ref1` was deleted because there would be no references to it. If you delete `ref2`, there are no references to the object; it will be destroyed, and the memory it used becomes available.

See also

`var`

do while

Availability

Flash Player 4.

Usage

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

Parameters

`condition` The condition to evaluate.

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

Returns

Nothing.

Description

Statement; similar to a `while` loop, except that the statements are executed once before the initial evaluation of the condition. Subsequently, the statements are executed only if the condition evaluates to `true`.

A `do..while` loop ensures that the code inside the loop executes at least once. Although this can also be done with a `while` loop by placing a copy of the statements to be executed before the `while` loop begins, many programmers believe that `do..while` loops are easier to read.

If the condition always evaluates to `true`, the `do..while` loop is infinite. If you enter an infinite loop, you encounter problems with Flash Player and eventually get a warning message or crash the player. Whenever possible, you should use a `for` loop if you know the number of times you want to loop. Although `for` loops are easy to read and debug, they cannot replace `do..while` loops in all circumstances.

Example

The following example uses a `do..while` loop to evaluate whether a condition is `true`, and traces `myVar` until `myVar` is greater than 5. When `myVar` is greater than 5, the loop ends.

```
var myVar:Number = 0;
do {
  trace(myVar);
  myVar++;
} while (myVar<5);
/* output:
  0
  1
  2
  3
  4
*/
```

See also

`break`, `continue`, `while`

duplicateMovieClip()

Availability

Flash Player 4.

Usage

`duplicateMovieClip(`*`target:String`*`, `*`newname:String`*`, `*`depth:Number`*`) : `*`Void`*

Parameters

`target` The target path of the movie clip to duplicate.

`newname` A unique identifier for the duplicated movie clip.

`depth` A unique depth level for the duplicated movie clip. The depth level is a stacking order for duplicated movie clips. This stacking order is similar to the stacking order of layers in the Timeline; movie clips with a lower depth level are hidden under clips with a higher stacking order. You must assign each duplicated movie clip a unique depth level to prevent it from replacing SWF files on occupied depths.

Returns

Nothing

Description

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

Example

In the following example, a new movie clip instance is created called `img_mc`. An image is loaded into the movie clip, and then the `img_mc` clip is duplicated. The duplicated clip is called `newImg_mc`, and this new clip is moved on the Stage so it does not overlap the original clip, and the same image is loaded into the second clip.

```
this.createEmptyMovieClip("img_mc", this.getNextHighestDepth());
img_mc.loadMovie("http://www.macromedia.com/images/shared/product_boxes/
  112x112/box_studio_112x112.jpg");
duplicateMovieClip(img_mc, "newImg_mc", this.getNextHighestDepth());
newImg_mc._x = 200;
newImg_mc.loadMovie("http://www.macromedia.com/images/shared/product_boxes/
  112x112/box_studio_112x112.jpg");
```

To remove the duplicate movie clip, you could add this code for a button called `myButton_btn`.

```
this.myButton_btn.onRelease = function(){
  removeMovieClip(newImg_mc);
};
```

See also

`MovieClip.duplicateMovieClip()`, `removeMovieClip()`, `MovieClip.removeMovieClip()`

dynamic

Availability

Flash Player 6.

Usage

```
dynamic class className  [ extends superClass ]
                [ implements interfaceName [, interfaceName... ] ]
{
  // class definition here
}
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Description

Keyword; specifies that objects based on the specified class can add and access dynamic properties at runtime.

Type checking on dynamic classes is less strict than type checking on nondynamic classes, because members accessed inside the class definition and on class instances are not compared with those defined in the class scope. Class member functions, however, can still be type checked for return type and parameter types. This behavior is especially useful when you work with MovieClip objects, because there are many different ways of adding properties and objects to a movie clip dynamically, such as `MovieClip.createEmptyMovieClip()` and `MovieClip.createTextField()`.

Subclasses of dynamic classes are also dynamic.

For more information, see "Creating dynamic classes" in *Using ActionScript in Flash*.

Example

In the following example, class `Person2` has not yet been marked as dynamic, so calling an undeclared function on it generates an error at compile time:

```
class Person2 {
  var name:String;
  var age:Number;
  function Person2(param_name:String, param_age:Number) {
    trace ("anything");
    this.name = param_name;
    this.age = param_age;
  }
}
```

In a FLA or AS file that's in the same directory, add the following ActionScript to Frame 1 on the Timeline:

```
// Before dynamic is added
var craig:Person2 = new Person2("Craiggers", 32);
for (i in craig) {
  trace("craig."+i +" = "+ craig[i]);
}
```

```
/* output:
  craig.age = 32
  craig.name = Craiggers
*/
```

If you add an undeclared function, `dance`, an error is generated, as shown in the following example:

```
trace("");
craig.dance = true;
for (i in craig) {
  trace("craig."+i +" = "+ craig[i]);
}
/* output:
  **Error** Scene=Scene 1, layer=Layer 1, frame=1:Line 14: There is no
  property with the name 'dance'.
     craig.dance = true;

  Total ActionScript Errors: 1  Reported Errors: 1
*/
```

Add the `dynamic` keyword to the Person2 class, so that the first line appears as follows:

```
dynamic class Person2 {
```

Test the code again, and you see the following output:

```
craig.dance = true
craig.age = 32
craig.name = Craiggers
```

See also

`class`, `extends`

else

Availability

Flash Player 4.

Usage

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

Parameters

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

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

Returns

Nothing.

Description

Statement; specifies the statements to run if the condition in the `if` statement returns `false`. The curly braces (`{}`) used to enclose the block of statements to be executed by the `else` statement are not necessary if only one statement will execute.

Example

In the following example, the `else` condition is used to check whether the `age_txt` variable is greater than or less than 18:

```
if (age_txt.text>=18) {
  trace("welcome, user");
} else {
  trace("sorry, junior");
  userObject.minor = true;
  userObject.accessAllowed = false;
}
```

In the following example, curly braces (`{}`) are not necessary because only one statement follows the `else` statement:

```
if (age_txt.text>18) {
  trace("welcome, user");
}
else trace("sorry, junior");
```

See also

`if`, `else if`, `switch`

else if

Availability

Flash Player 4.

Usage

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

Parameters

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

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

Returns

Nothing.

Description

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

Use the `else if` statement to create branching logic in your scripts. If there are multiple branches, you should consider using a `switch` statement.

Example

The following example uses `else if` statements to compare `score_txt` to a specified value:

```
if (score_txt.text>90) {
   trace("A");
} else if (score_txt.text>75) {
   trace("B");
} else if (score_txt.text>60) {
   trace("C");
} else {
   trace("F");
}
```

See also

`if`, `else`, `switch`

#endinitclip

Availability

Flash Player 6.

Usage

```
#endinitclip
```

Parameters

None.

Returns

Nothing.

Description

Compiler directive; indicates the end of a block of initialization actions.

Example

```
#initclip
...initialization actions go here...
#endinitclip
```

See also

`#initclip`

Error class

Availability

Flash Player 7.

Description

Contains information about an error that occurred in a script. You create an Error object using the `Error` constructor function. Typically, you *throw* a new Error object from within a `try` code block that is then *caught* by a `catch` or `finally` code block.

You can also create a subclass of the Error class and throw instances of that subclass.

Method summary for the Error class

Method	Description
`Error.toString()`	Returns the string representation of an Error object.

Property summary for the Error class

Property	Description
`Error.message`	A string that contains an error message associated with an error.
`Error.name`	A string that contains the name of the Error object.

Constructor for the Error class

Availability

Flash Player 7.

Usage

```
new Error([message:String]) : Error
```

Parameters

`message` A string associated with the Error object; this parameter is optional.

Returns

A reference to an Error object.

Description

Constructor; creates a new Error object. If *`message`* is specified, its value is assigned to the object's `Error.message` property.

Example

In the following example, a function throws an error (with a specified message) if the two strings that are passed to it are not identical:

```
function compareStrings(str1_str:String, str2_str:String):Void {
  if (str1_str != str2_str) {
        throw new Error("Strings to not match.");
    }
}
try {
    compareStrings("Dog", "dog");
    // output: Strings to not match.
} catch (e_err:Error) {
    trace(e_err.toString());
}
```

See also

`throw`, `try..catch..finally`

Error.message

Availability

Flash Player 7.

Usage

```
my_err.message:String
```

Description

Property; contains the message associated with the Error object. By default, the value of this property is "`Error`". You can specify a `message` property when you create an Error object by passing the error string to the `Error` constructor function.

Example

In the following example, a function throws a specified message depending on the parameters entered into `theNum`. If two numbers can be divided, `SUCCESS` and the number are shown. Specific errors are shown if you try to divide by 0 or enter only 1 parameter:

```
function divideNum(num1:Number, num2:Number):Number {
  if (isNaN(num1) || isNaN(num2)) {
    throw new Error("divideNum function requires two numeric parameters.");
  } else if (num2 == 0) {
    throw new Error("cannot divide by zero.");
  }
  return num1/num2;
}
try {
  var theNum:Number = divideNum(1, 0);
  trace("SUCCESS! "+theNum);
} catch (e_err:Error) {
  trace("ERROR! "+e_err.message);
  trace("\t"+e_err.name);
}
```

If you test this ActionScript without any modifications to the numbers you divide, you see an error displayed in the Output panel because you are trying to divide by 0.

See also

`throw`, `try..catch..finally`

Error.name

Availability

Flash Player 7.

Usage

```
myError.name:String
```

Description

Property; contains the name of the Error object. By default, the value of this property is `"Error"`.

Example

In the following example, a function throws a specified error depending on the two numbers that you try to divide. Add the following ActionScript to Frame 1 of the Timeline:

```
function divideNumber(numerator:Number, denominator:Number):Number {
  if (isNaN(numerator) || isNaN(denominator)) {
    throw new Error("divideNum function requires two numeric parameters.");
  } else if (denominator == 0) {
    throw new DivideByZeroError();
  }
  return numerator/denominator;
}
try {
  var theNum:Number = divideNumber(1, 0);
  trace("SUCCESS! "+theNum);
  // output: DivideByZeroError -> Unable to divide by zero.
} catch (e_err:DivideByZeroError) {
  // divide by zero error occurred
  trace(e_err.name+" -> "+e_err.toString());
} catch (e_err:Error) {
  // generic error occurred
  trace(e_err.name+" -> "+e_err.toString());
}
```

To add a custom error, add the following code to a .AS file called DivideByZeroError.as and save the class file in the same directory as your FLA document.

```
class DivideByZeroError extends Error {
  var name:String = "DivideByZeroError";
  var message:String = "Unable to divide by zero.";
}
```

See also

`throw`, `try..catch..finally`

Error.toString()

Availability

Flash Player 7.

Usage

```
my_err.toString() : String
```

Returns

A string.

Description

Method; returns the string `"Error"` by default or the value contained in `Error.message`, if defined.

Example

In the following example, a function throws an error (with a specified message) if the two strings that are passed to it are not identical:

```
function compareStrings(str1_str:String, str2_str:String):Void {
  if (str1_str != str2_str) {
    throw new Error("Strings to not match.");
  }
}
try {
  compareStrings("Dog", "dog");
  // output: Strings to not match.
} catch (e_err:Error) {
  trace(e_err.toString());
}
```

See also

Error.message, `throw`, `try..catch..finally`

escape()

Availability

Flash Player 5.

Usage

```
escape(expression:String) : String
```

Parameters

`expression` The expression to convert into a string and encode in a URL-encoded format.

Returns

URL-encoded string.

Description

Function; converts the parameter to a string and encodes it in a URL-encoded format, where all nonalphanumeric characters are replaced with % hexadecimal sequences. When used in a URL-encoded string, the percentage symbol (%) is used to introduce escape characters, and is not equivalent to the modulo operator (%).

Example

The following code produces the result `someuser%40somedomain%2Ecom`:

```
var email:String = "someuser@somedomain.com";
trace(escape(email));
```

In this example, the at symbol (@) was replaced with `%40` and the dot symbol (`.`) was replaced with `%2E`. This is useful if you're trying to pass information to a remote server and the data has special characters in it (for example, `&` or `?`), as shown in the following code:

```
var redirectUrl = "http://www.somedomain.com?loggedin=true&username=Gus";
getURL("http://www.myothersite.com?returnurl="+ escape(redirectUrl));
```

See also

`unescape()`

eval()

Availability

Flash Player 5 or later for full functionality. You can use the `eval()` function when exporting to Flash Player 4, but you must use slash notation and can access only variables, not properties or objects.

Usage

`eval(`*`expression:String`*`) : `*`Object`*

Parameters

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

Returns

A value, reference to an object or movie clip, or `undefined`.

Description

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

In Flash 4, `eval()` was used to simulate arrays; in Flash 5 or later, you should use the Array class to simulate arrays.

In Flash 4, you can also use `eval()` to dynamically set and retrieve the value of a variable or instance name. However, you can also do this with the array access operator (`[]`).

In Flash 5 or later, you cannot use `eval()` to dynamically set and retrieve the value of a variable or instance name, because you cannot use `eval()` on the left side of an equation. For example, replace the code

```
eval ("var" + i) = "first";
```

with this:

```
this["var"+i] = "first"
```

or this:

```
set ("var" + i, "first");
```

Example

The following example uses `eval()` to set properties for dynamically named movie clips. This ActionScript sets the `_rotation` property for three movie clips, called `square1_mc`, `square2_mc`, and `square3_mc`.

```
for (var i = 1; i<=3; i++) {
  setProperty(eval("square"+i+"_mc"), _rotation, 5);
}
```

You can also use the following ActionScript:

```
for (var i = 1; i<=3; i++) {
  this["square"+i+"_mc"]._rotation = -5;
}
```

See also

Array class, set variable

extends

Availability

Flash Player 6.

Usage

```
class className extends otherClassName {}
```

```
interface interfaceName extends otherInterfaceName {}
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Parameters

`className` The name of the class you are defining.

`otherClassName` The name of the class on which `className` is based.

`interfaceName` The name of the interface you are defining.

`otherInterfaceName` The name of the interface on which `interfaceName` is based.

Description

Keyword; defines a class that is a subclass of another class; the latter is the superclass. The subclass inherits all the methods, properties, functions, and so on that are defined in the superclass.

For more information, see "Creating subclasses" in *Using ActionScript in Flash*.Interfaces can also be extended using the `extends` keyword. An interface that extends another interface includes all the original interface's method declarations.

Example

In the following example, the Car class extends the Vehicle class so that all its methods, properties, and functions are inherited. If your script instantiates a Car object, methods from both the Car class and the Vehicle class can be used.

The following example shows the contents of a file called Vehicle.as, which defines the Vehicle class:

```
class Vehicle {
  var numDoors:Number;
  var color:String;
  function Vehicle(param_numDoors:Number, param_color:String) {
    this.numDoors = param_numDoors;
    this.color = param_color;
  }
  function start():Void {
    trace("[Vehicle] start");
  }
  function stop():Void {
    trace("[Vehicle] stop");
  }
}
```

The following example shows a second AS file, called Car.as, in the same directory. This class extends the Vehicle class, modifying it in three ways. First, the Car class adds a variable `fullSizeSpare` to track whether the car object has a full-size spare tire. Second, it adds a new method specific to cars, `activateCarAlarm()`, that activates the car's anti-theft alarm. Third, it overrides the `stop()` function to add the fact that the Car class uses an anti-lock braking system to stop.

```
class Car extends Vehicle {
  var fullSizeSpare:Boolean;
  function Car(param_numDoors:Number, param_color:String,
  param_fullSizeSpare:Boolean) {
    this.numDoors = param_numDoors;
    this.color = param_color;
    this.fullSizeSpare = param_fullSizeSpare;
  }
  function activateCarAlarm():Void {
    trace("[Car] activateCarAlarm");
  }
  function stop():Void {
    trace("[Car] stop with anti-lock brakes");
  }
}
```

The following example instantiates a Car object, calls a method defined in the Vehicle class (`start()`), then calls the method overridden by the Car class (`stop()`), and finally calls a method from the Car class (`activateCarAlarm()`):

```
var myNewCar:Car = new Car(2, "Red", true);
myNewCar.start(); // output: [Vehicle] start
myNewCar.stop(); // output: [Car] stop with anti-lock brakes
myNewCar.activateCarAlarm(); // output: [Car] activateCarAlarm
```

See also

`class`, `implements`, `interface`

false

Availability

Flash Player 5.

Usage

```
false
```

Description

Constant; a unique Boolean value that represents the opposite of `true`. When automatic data typing converts `false` to a number, it becomes 0; when it converts `false` to a string, it becomes `"false"`. For more information, see "Automatic data typing" in *Using ActionScript in Flash*.

Example

This example shows how automatic data typing converts `false` to a number and to a string:

```
var bool1:Boolean = Boolean(false);
// converts it to the number 0
trace(1 + bool1); // outputs 1
// converts it to a string
trace("String: " + bool1); // outputs String: false
```

See also

`true`

_focusrect

Availability

Flash Player 4.

Usage

```
_focusrect = Boolean;
```

Description

Property (global); specifies whether a yellow rectangle appears around the button or movie clip that has keyboard focus. If `_focusrect` is set to its default value of `true`, then a yellow rectangle appears around the currently focused button or movie clip as the user presses the Tab key to navigate through objects in a SWF file. Specify `false` if you do not want to show the yellow rectangle. This is a global property that can be overridden for specific instances.

Note: If you use a component, then FocusManager overrides Flash Player's focus handling, including use of this global property.

Example

The following example demonstrates how to hide the yellow rectangle around any instances in a SWF file when they have focus in a browser window. Create some buttons or movie clips and add the following ActionScript in frame 1 of the Timeline:

```
_focusrect = false;
```

Change the publish settings to Flash Player 6, and test the SWF file in a browser window by selecting File > Publish Preview > HTML. Give the SWF focus by clicking it in the browser window, and use the Tab key to focus each instance. Pressing Enter or the Space key when `_focusrect` is disabled does not invoke the `onRelease` event handler as it does when `_focusrect` is enabled or `true`.

See also

`Button._focusrect`, `MovieClip._focusrect`

for

Availability

Flash Player 5.

Usage

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

Parameters

`init` An expression to evaluate before beginning the looping sequence; usually an assignment expression. A `var` statement is also permitted for this parameter.

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

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

`statement(s)` An instruction or instructions to execute within the body of the loop.

Description

Statement; evaluates the *`init`* (initialize) expression once and then starts a looping sequence. The looping sequence begins by evaluating the *`condition`* expression. If the *`condition`* expression evaluates to true, *`statement`* is executed and the *`next`* expression is evaluated. The looping sequence then begins again with the evaluation of the *`condition`* expression.

The curly braces (`{}`) used to enclose the block of statements to be executed by the `for` statement are not necessary if only one statement will execute.

Example

The following example uses `for` to add the elements in an array:

```
var my_array:Array = new Array();
for (var i:Number = 0; i<10; i++) {
  my_array[i] = (i+5)*10;
  //trace(my_array[i]);
}
trace(my_array);  // output: 50,60,70,80,90,100,110,120,130,140
```

The following example uses `for` to perform the same action repeatedly. In the code, the `for` loop adds the numbers from 1 to 100.

```
var sum:Number = 0;
for (var i:Number = 1; i<=100; i++) {
  sum += i;
}
trace(sum);  // output: 5050
```

The following example shows that curly braces ({}) are not necessary if only one statement will execute:

```
var sum:Number = 0;
for (var i:Number = 1; i<=100; i++)
  sum += i;
trace(sum);  // output: 5050
```

See also

`++ (increment)`, `-- (decrement)`, `for..in`, `var`, `while`, `do while`

for..in

Availability

Flash Player 5.

Usage

```
for(variableIterant in object){
  statement(s);
}
```

Parameters

variableIterant The name of a variable to act as the iterant, referencing each property of an object or element in an array.

object The name of an object to be iterated.

statement(s) An instruction to execute for each iteration.

Returns

Nothing.

Description

Statement; iterates over the properties of an object or elements in an array and executes the *statement* for each property or element. Methods of an object are not enumerated by the `for..in` action.

Some properties cannot be enumerated by the `for..in` action. For example, movie clip properties, such as `_x` and `_y`, are not enumerated. In external class files, static members are not enumerable, unlike instance members.

The `for..in` statement iterates over properties of objects in the iterated object's prototype chain. Properties of the object are enumerated first, then properties of its immediate prototype, then properties of the prototype's prototype, and so on. The `for..in` statement does not enumerate the same property name twice. If the object `child` has prototype `parent` and both contain the property `prop`, the `for..in` statement called on `child` enumerates `prop` from `child` but ignores the one in `parent`.

The curly braces (`{}`) used to enclose the block of statements to be executed by the `for..in` statement are not necessary if only one statement will execute.

If you write a `for..in` loop in a class file (an external AS file), then instance members are not available for the loop, but static members are. However, if you write a `for..in` loop in a FLA file for an instance of the class, then instance members are available but static ones are not.

Examples

The following example shows using `for..in` to iterate over the properties of an object:

```
var myObject:Object = {name:"Tara", age:27, city:"San Francisco"};
for (var name in myObject) {
  trace("myObject."+name+" = "+myObject[name]);
}
```

```
//output
myObject.name = Tara
myObject.age = 27
myObject.city = San Francisco
```

The following example shows using `for..in` to iterate over the elements of an array:

```
var myArray:Array = new Array("one", "two", "three");
for (var index in myArray)
  trace("myArray["+index+"] = " + myArray[index]);

// output:
myArray[2] = three
myArray[1] = two
myArray[0] = one
```

The following example uses the `typeof` operator with `for..in` to iterate over a particular type of child:

```
for (var name in this) {
  if (typeof (this[name]) == "movieclip") {
    trace("I have a movie clip child named "+name);
  }
}
```

Note: If you have several movie clips, the output consists of the instance names of those clips.

The following example enumerates the children of a movie clip and sends each to Frame 2 in their respective Timelines. The `RadioButtonGroup` movie clip is a parent with several children, `_RedRadioButton_`, `_GreenRadioButton_`, and `_BlueRadioButton`.

```
for (var name in RadioButtonGroup) {
  RadioButtonGroup[name].gotoAndStop(2);
}
```

fscommand()

Availability

Flash Player 3.

Usage

```
fscommand("command", "parameters")
```

Parameters

`command` A string passed to the host application for any use, or a command passed to Flash Player.

`parameters` A string passed to the host application for any use, or a value passed to Flash Player.

Returns

Nothing.

Description

Function; lets the SWF file communicate with either Flash Player or the program hosting Flash Player, such as a web browser. You can also use the `fscommand()` function to pass messages to Macromedia Director, or to Visual Basic, Visual C++, and other programs that can host ActiveX controls.

Usage 1: To send a message to Flash Player, you must use predefined commands and parameters. The following table shows the values you can specify for the *`command`* and *`parameters`* parameters of the `fscommand()` function to control a SWF file playing in Flash Player, including projectors.

Command	Parameters	Purpose
`quit`	`None`	Closes the projector.
`fullscreen`	`true` or `false`	Specifying `true` sets Flash Player to full-screen mode. Specifying `false` returns the player to normal menu view.
`allowscale`	`true` or `false`	Specifying `false` sets the player so that the SWF file is always drawn at its original size and never scaled. Specifying `true` forces the SWF file to scale to 100% of the player.
`showmenu`	`true` or `false`	Specifying `true` enables the full set of context menu items. Specifying `false` dims all the context menu items except About Flash Player.
`exec`	`Path to application`	Executes an application from within the projector.
`trapallkeys`	`true` or `false`	Specifying `true` sends all key events, including accelerator keys, to the `onClipEvent(keyDown/keyUp)` handler in Flash Player.

The `exec` command can contain only the characters A–Z, a–z, 0–9, period (.), and underscore (_). The `exec` command runs in the subdirectory fscommand only. In other words, if you use the `fscommand exec` command to call an application, the application must reside in a subdirectory named fscommand.

Usage 2: To use the `fscommand()` function to send a message to a scripting language such as JavaScript in a web browser, you can pass any two parameters in the *`command`* and *`parameters`* parameters. These parameters can be strings or expressions and are used in a JavaScript function that handles, or *catches*, the `fscommand()` function.

In a web browser, the `fscommand()` function calls the JavaScript function `moviename_DoFSCommand` in the HTML page containing the SWF file. The `moviename` is the name of the Flash Player as assigned by the `NAME` attribute of the `EMBED` tag or the ID property of the `OBJECT` tag. If you assign the Flash Player the name `myDocument`, the JavaScript function called is `myDocument_DoFSCommand`.

Usage 3: The `fscommand()` function can send messages to Macromedia Director that are interpreted by Lingo (Director's scripting language) as strings, events, or executable Lingo code. If the message is a string or an event, you must write the Lingo code to receive the message from the `fscommand()` function and carry out an action in Director. For more information, see the Director Support Center at www.macromedia.com/support/director.

Usage 4: In Visual Basic, Visual C++, and other programs that can host ActiveX controls, the `fscommand()` function sends a VB event with two strings that can be handled in the environment's programming language. For more information, use the keywords *Flash method* to search the Flash Support Center at www.macromedia.com/support/flash.

Example

In the following example, the `fscommand()` function sets Flash Player to scale the SWF file to the full monitor screen size when the `fullscreen_btn` button or `unfullscreen_btn` is released:

```
this.fullscreen_btn.onRelease = function() {
  fscommand("fullscreen", true);
};
this.unfullscreen_btn.onRelease = function() {
  fscommand("fullscreen", false);
};
```

The following example uses the `fscommand()` function applied to a button in Flash to open a JavaScript message box in an HTML page. The message itself is sent to JavaScript as the `fscommand` parameter.

You must add a function to the HTML page that contains the SWF file. This function, *`myDocument`*`_DoFSCommand`, sits in the HTML page and waits for an `fscommand()` function in Flash. When an `fscommand` is triggered in Flash (for example, when a user presses the button), the `command` and `parameter` strings are passed to the *`myDocument`*`_DoFSCommand` function. You can use the passed strings in your JavaScript or VBScript code in any way you like. In this example, the function contains a conditional `if` statement that checks to see if the command string is `"messagebox"`. If it is, a JavaScript alert box (or "message box") opens and displays the contents of the `parameters` string.

```
function myDocument_DoFSCommand(command, args) {
  if (command == "messagebox") {
    alert(args);
  }
}
```

In the Flash document, add the `fscommand()` function to a button:

```
fscommand("messagebox", "This is a message box called from within Flash.")
```

You can also use expressions for the `fscommand()` function and parameters, as in the following example:

```
fscommand("messagebox", "Hello, " + name + ", welcome to our website!")
```

To test the SWF file, select File > Publish Preview > HTML.

Note: If you publish your SWF file using the Flash with FSCommand template in the HTML tab of the Publish Settings dialog box, the `myDocument_DoFSCommand` function is inserted automatically. The SWF file's `NAME` and `ID` attributes will be the filename. For example, for the file myDocument.fla, the attributes would be set to `myDocument`.

function

Availability

Flash Player 5.

Usage

```
function functionname ([parameter0, parameter1,...parameterN]){
  statement(s)
}
function ([parameter0, parameter1,...parameterN]){
  statement(s)
}
```

Parameters

`functionname` The name of the new function. This parameter is optional.

`parameter` An identifier that represents a parameter to pass to the function. This parameter is optional.

`statement(s)` Any ActionScript instruction you have defined for the body of the `function`.

Returns

Usage 1: Nothing.

Usage 2: A reference to the anonymous function.

Description

Statement; comprises a set of statements that you define to perform a certain task. You can define a function in one location and invoke, or *call*, it from different scripts in a SWF file. When you define a function, you can also specify parameters for the function. Parameters are placeholders for values on which the function operates. You can pass different parameters to a function each time you call it so you can reuse a function in different situations.

Use the `return` statement in a function's `statement(s)` to cause a function to generate, or *return*, a value.

You can use this statement to define a `function` with the specified `functionname`, `parameters`, and `statement(s)`. When a script calls a function, the statements in the function's definition are executed. Forward referencing is permitted; within the same script, a function may be declared after it is called. A function definition replaces any prior definition of the same function. You can use this syntax wherever a statement is permitted.

You can also use this statement to create an anonymous function and return a reference to it. This syntax is used in expressions and is particularly useful for installing methods in objects.

For additional functionality, you can use the `arguments object` in your function definition. Some common uses of the `arguments object` are creating a function that accepts a variable number of parameters and creating a recursive anonymous function.

Example

The following example defines the function `sqr`, which accepts one parameter and returns the `Math.pow(x, 2)` of the parameter:

```
function sqr(x:Number) {
  return Math.pow(x, 2);
}
var y:Number = sqr(3);
trace(y);  // output: 9
```

If the function is defined and used in the same script, the function definition may appear after using the function:

```
var y:Number = sqr(3);
trace(y);  // output: 9
function sqr(x:Number) {
  return Math.pow(x, 2);
}
```

The following function creates a LoadVars object and loads params.txt into the SWF file. When the file successfully loads, `variables loaded` traces:

```
var myLV:LoadVars = new LoadVars();
myLV.load("params.txt");
myLV.onLoad = function(success:Boolean) {
  trace("variables loaded");
}
```

Function class

Availability

Flash Player 6.

Description

Both user-defined and built-in functions in ActionScript are represented by Function objects, which are instances of the Function class.

Method summary for the Function class

Method	Description
`Function.apply()`	Invokes the function represented by a Function object, with parameters passed in through an array.
`Function.call()`	Invokes the function represented by a Function object.

Function.apply()

Availability

Flash Player 6.

Usage

```
myFunction.apply(thisObject:Object, argumentsArray:Array)
```

Parameters

`thisObject` The object to which `myFunction` is applied.

`argumentsArray` An array whose elements are passed to `myFunction` as parameters.

Returns

Any value that the called function specifies.

Description

Method; specifies the value of `this` to be used within any function that ActionScript calls. This method also specifies the parameters to be passed to any called function. Because `apply()` is a method of the Function class, it is also a method of every Function object in ActionScript.

The parameters are specified as an Array object, unlike Function.call(), which specifies parameters as a comma-delimited list. This is often useful when the number of parameters to be passed is not known until the script actually executes.

Example

The following function invocations are equivalent:

```
Math.atan2(1, 0)
Math.atan2.apply(null, [1, 0])
```

The following simple example shows how `apply()` passes an array of parameters:

```
function theFunction() {
  trace(arguments);
}

// create a new array to pass as a parameter to apply()
var firstArray:Array = new Array(1,2,3);
theFunction.apply(null,firstArray);
// outputs: 1,2,3

// create a second array to pass as a parameter to apply()
var secondArray:Array = new Array("a", "b", "c");
theFunction.apply(null,secondArray);
// outputs a,b,c
```

The following example shows how `apply()` passes an array of parameters and specifies the value of `this`:

```
// define a function
function theFunction() {
  trace("this == myObj? " + (this == myObj));
  trace("arguments: " + arguments);
}

// instantiate an object
var myObj:Object = new Object();

// create arrays to pass as a parameter to apply()
var firstArray:Array = new Array(1,2,3);
var secondArray:Array = new Array("a", "b", "c");

// use apply() to set the value of this to be myObj and send firstArray
theFunction.apply(myObj,firstArray);
// output:
// this == myObj? true
// arguments: 1,2,3

// use apply() to set the value of this to be myObj and send secondArray
theFunction.apply(myObj,secondArray);
// output:
// this == myObj? true
// arguments: a,b,c
```

See also

Function.call()

Function.call()

Availability

Flash Player 6.

Usage

`myFunction.call(thisObject:Object, parameter1, ..., parameterN)`

Parameters

`thisObject` An object that specifies the value of `this` within the function body.

`parameter1` A parameter to be passed to the `myFunction`. You can specify zero or more parameters.

`parameterN`

Returns

Nothing.

Description

Method; invokes the function represented by a Function object. Every function in ActionScript is represented by a Function object, so all functions support this method.

In almost all cases, the function call (`()`) operator can be used instead of this method. The function call operator produces code that is concise and readable. This method is primarily useful when the `this` parameter of the function invocation needs to be explicitly controlled. Normally, if a function is invoked as a method of an object, within the body of the function, `this` is set to `myObject`, as shown in the following example:

```
myObject.myMethod(1, 2, 3);
```

In some situations, you might want `this` to point somewhere else; for example, if a function must be invoked as a method of an object, but is not actually stored as a method of that object:

```
myObject.myMethod.call(myOtherObject, 1, 2, 3);
```

You can pass the value `null` for the `thisObject` parameter to invoke a function as a regular function and not as a method of an object. For example, the following function invocations are equivalent:

```
Math.sin(Math.PI / 4)
Math.sin.call(null, Math.PI / 4)
```

Example

The following example uses `Function.call()` to make a function behave as a method of another object, without storing the function in the object:

```
function myObject() {
}
function myMethod(obj) {
  trace("this == obj? " + (this == obj));
}
```

```
var obj:Object = new myObject();
myMethod.call(obj, obj);
```

The `trace()` statement displays:

```
this == obj? true
```

See also

`Function.apply()`

get

Availability

Flash Player 6.

Usage

```
function get property() {
  // your statements here
}
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Parameters

`property` The word you use to refer to the property that `get` accesses; this value must be the same as the value used in the corresponding `set` command.

Returns

Nothing.

Description

Keyword; permits implicit *getting* of properties associated with objects based on classes you have defined in external class files. Using implicit get methods lets you access properties of objects without accessing the property directly. Implicit get/set methods are syntactic shorthand for the `Object.addProperty()` method in ActionScript 1.

For more information, see "Implicit getter/setter methods" in *Using ActionScript in Flash.*

Example

In the following example, you define a Team class. The Team class includes get/set methods that let you retrieve and set properties within the class:

```
class Team {
  var teamName:String;
  var teamCode:String;
  var teamPlayers:Array = new Array();
  function Team(param_name:String, param_code:String) {
    this.teamName = param_name;
    this.teamCode = param_code;
  }
  function get name():String {
    return this.teamName;
  }
  function set name(param_name:String):Void {
    this.teamName = param_name;
  }
}
```

Enter the following ActionScript in a frame on the Timeline:

```
var giants:Team = new Team("San Fran", "SFO");
trace(giants.name);
giants.name = "San Francisco";
trace(giants.name);
/* output:
   San Fran
   San Francisco
*/
```

When you trace giants.name, you use the get method to return the value of the property.

See also

`Object.addProperty()`, `set`

getProperty()

Availability

Flash Player 4.

Usage

`getProperty(`*`my_mc:Object`*`, `*`property:Object`*`) : `*`Object`*

Parameters

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

`property` A property of a movie clip.

Returns

The value of the specified property.

Description

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

Example

The following example creates a new movie clip `someClip_mc` and shows the alpha value (`_alpha`) for the movie clip `someClip_mc` in the Output panel:

```
this.createEmptyMovieClip("someClip_mc", 999);
trace("The alpha of "+getProperty(someClip_mc, _name)+" is:
  "+getProperty(someClip_mc, _alpha));
```

getTimer()

Availability

Flash Player 4.

Usage

```
getTimer() : Number
```

Parameters

None.

Returns

The number of milliseconds that have elapsed since the SWF file started playing.

Description

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

Example

In the following example, the `getTimer()` and `setInterval()` functions are used to create a simple timer:

```
this.createTextField("timer_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
function updateTimer():Void {
  timer_txt.text = getTimer();
}
var intervalID:Number = setInterval(updateTimer, 100);
```

getURL()

Availability

Flash 2. The `GET` and `POST` options are available only in Flash Player 4 and later versions.

Usage

`getURL(url:String [, window:String [, "variables":String]]) : Void`

Parameters

`url` The URL from which to obtain the document.

`window` An optional parameter specifying the window or HTML frame into which the document should load. You can enter the name of a specific window or select from the following reserved target names:

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

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

Returns

Nothing.

Description

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

Example

This example loads an image into a movie clip. When the image is clicked, a new URL is loaded in a new browser window.

```
var listenerObject:Object = new Object();
listenerObject.onLoadInit = function(target_mc:MovieClip) {
  target_mc.onRelease = function() {
    getURL("http://www.macromedia.com/software/flash/flashpro/", "_blank");
  };
};
var logo:MovieClipLoader = new MovieClipLoader();
logo.addListener(listenerObject);
logo.loadClip("http://www.macromedia.com/images/shared/product_boxes/159x120/
  159x120_box_flashpro.jpg", this.createEmptyMovieClip("macromedia_mc",
  this.getNextHighestDepth()));
```

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

```
myBtn_btn.onRelease = function(){
  getURL("mailto:you@somedomain.com");
};
```

In the following ActionScript, JavaScript is used to open an alert window when the SWF file is embedded in a browser window:

```
myBtn_btn.onRelease = function(){
  getURL("javascript:alert('you clicked me')");
};
```

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

```
var firstName:String = "Gus";
var lastName:String = "Richardson";
var age:Number = 92;
myBtn_btn.onRelease = function() {
  getURL("http://www.macromedia.com", "_blank", "GET");
};
```

The following ActionScript uses `POST` to send variables in the HTTP header. Make sure you test your documents in a browser window, because otherwise your variables are sent using `GET`:

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

See also

`loadVariables()`, `XML.send()`, `XML.sendAndLoad()`, `XMLSocket.send()`

getVersion()

Availability

Flash Player 5.

Usage

```
getVersion() : String
```

Parameters

None.

Returns

A string containing Flash Player version and platform information.

Description

Function; returns a string containing Flash Player version and platform information.

The `getVersion` function returns information only for Flash Player 5 or later versions of Flash Player.

Example

The following examples trace the version number of the Flash Player playing the SWF file:

```
var flashVersion:String = getVersion();
trace(flashVersion);  // output: WIN 7,0,19,0
trace($version);  // output: WIN 7,0,19,0
trace(System.capabilities.version);  // output: WIN 7,0,19,0
```

The following string is returned by the `getVersion` function:

```
WIN 7,0,19,0
```

This returned string indicates that the platform is Microsoft Windows, and the version number of Flash Player is major version 7, minor version 19 (7.19).

See also

`System.capabilities.os`, `System.capabilities.version`

_global object

Availability

Flash Player 6.

Usage

_global.identifier

Parameters

None.

Returns

A reference to the global object that holds the core ActionScript classes, such as String, Object, Math, and Array.

Description

Identifier; creates global variables, objects, or classes. For example, you could create a library that is exposed as a global ActionScript object, similar to the Math or Date object. Unlike Timeline-declared or locally declared variables and functions, global variables and functions are visible to every Timeline and scope in the SWF file, provided they are not obscured by identifiers with the same names in inner scopes.

Example

The following example creates a top-level function, `factorial()`, that is available to every Timeline and scope in a SWF file:

```
_global.factorial = function(n:Number) {
  if (n<=1) {
    return 1;
  } else {
    return n*factorial(n-1);
  }
};
// Note: factorial 4 == 4*3*2*1 == 24
trace(factorial(4)); // output: 24
```

See also

`var`, `set variable`

gotoAndPlay()

Availability

Flash 2.

Usage

`gotoAndPlay([scene:String,] frame:Object) : Void`

Parameters

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

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

Returns

Nothing.

Description

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

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

Example

In the following example, a document has two scenes: `sceneOne` and `sceneTwo`. Scene one contains a frame label on Frame 10 called `newFrame` and two buttons, `myBtn_btn` and `myOtherBtn_btn`. This ActionScript is placed on Frame 1, Scene 1 of the main Timeline.

```
stop();
myBtn_btn.onRelease = function(){
  gotoAndPlay("newFrame");
};
myOtherBtn_btn.onRelease = function(){
  gotoAndPlay("sceneTwo", 1);
};
```

When the user clicks the buttons, the playhead moves to the specified location and continues playing.

See also

`MovieClip.gotoAndPlay()`, `play()`, `nextFrame()`, `prevFrame()`

gotoAndStop()

Availability

Flash 2.

Usage

`gotoAndStop([`*`scene:String,`*`]`*`frame:Object`*`) : `*`Void`*

Parameters

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

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

Returns

Nothing.

Description

Function; sends the playhead to the specified frame in a scene and stops it. If no scene is specified, the playhead is sent to the frame in the current scene.

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

Example

In the following example, a document has two scenes: `sceneOne` and `sceneTwo`. Scene one contains a frame label on Frame 10 called `newFrame`, and two buttons, `myBtn_btn` and `myOtherBtn_btn`. This ActionScript is placed on Frame 1, Scene 1 of the main Timeline:

```
stop();
myBtn_btn.onRelease = function(){
  gotoAndStop("newFrame");
};
myOtherBtn_btn.onRelease = function(){
  gotoAndStop("sceneTwo", 1);
};
```

When the user clicks the buttons, the playhead moves to the specified location and stops.

See also

`MovieClip.gotoAndStop()`, `stop()`, `play()`, `gotoAndPlay()`

if

Availability

Flash Player 4.

Usage

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

Parameters

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

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

Returns

Nothing.

Description

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

The curly braces (`{}`) used to enclose the block of statements to be executed by the `if` statement are not necessary if only one statement will execute.

Example

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

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

The following example uses an `if` statement to evaluate how long it takes a user to click the `submit_btn` instance in a SWF file. If a user clicks the button more than 10 seconds after the SWF file plays, the condition evaluates to `true` and the message inside the curly braces (`{}`) appears in a text field that's created at runtime (using `createTextField()`). If the user clicks the button less than 10 seconds after the SWF file plays, the condition evaluates to `false` and a different message appears.

```
this.createTextField("message_txt", this.getNextHighestDepth, 0, 0, 100, 22);
message_txt.autoSize = true;
var startTime:Number = getTimer();
this.submit_btn.onRelease = function() {
  var difference:Number = (getTimer()-startTime)/1000;
  if (difference>10) {
    this._parent.message_txt.text = "Not very speedy, you took "+difference+"
  seconds.";
```

```
    } else {
      this._parent.message_txt.text = "Very good, you hit the button in
    "+difference+" seconds.";
    }
  };
```

See also

else

implements

Availability

Flash Player 6.

Usage

```
myClass implements interface01 [, interface02, ...]
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Description

Keyword; specifies that a class must define all the methods declared in the interface (or interfaces) being implemented. For more information, see "Interfaces as data types" in *Using ActionScript in Flash.*

Example

See `interface`.

See also

`class`, `extends`, `interface`

import

Availability

Flash Player 6.

Usage

```
import className
```

```
import packageName.*
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This statement is supported in the Actions panel as well as in external class files.

Parameters

className The fully qualified name of a class you have defined in an external class file.

packageName A directory in which you have stored related class files.

Description

Keyword; lets you access classes without specifying their fully qualified names. For example, if you want to use a custom class macr.util.users.UserClass in a script, you must refer to it by its fully qualified name or import it; if you import it, you can refer to it by the class name:

```
// before importing
var myUser:macr.util.users.UserClass = new macr.util.users.UserClass();
// after importing
import macr.util.users.UserClass;
var myUser:UserClass = new UserClass();
```

If there are several class files in the package (`working_directory/macr/utils/users`) that you want to access, you can import them all in a single statement, as shown in the following example:

```
import macr.util.users.*;
```

You must issue the `import` statement before you try to access the imported class without fully specifying its name.

If you import a class but don't use it in your script, the class isn't exported as part of the SWF file. This means you can import large packages without being concerned about the size of the SWF file; the bytecode associated with a class is included in a SWF file only if that class is actually used.

The `import` statement applies only to the current script (frame or object) in which it's called. For example, suppose on Frame 1 of a Flash document you import all the classes in the macr.util package. On that frame, you can reference classes in that package by their simple names:

```
// On Frame 1 of a FLA:
import macr.util.*;

var myFoo:foo = new foo();
```

On another frame script, however, you would need to reference classes in that package by their fully qualified names (`var myFoo:foo = new macr.util.foo();`) or add an `import` statement to the other frame that imports the classes in that package.

For more information on importing, see “Importing classes” and “Using packages” in *Using ActionScript in Flash*.

#include

Availability

Flash Player 4.

Usage

```
#include "[path] filename.as"
```

Note: Do not place a semicolon (;) at the end of the line that contains the `#include` statement.

Parameters

`[path] filename.as` The filename and optional path for the script to add to the Actions panel or to the current script; *.as* is the recommended filename extension.

Returns

Nothing.

Description

Compiler directive: includes the contents of the specified file, as if the commands in the file are part of the calling script. The `#include` directive is invoked at compile time. Therefore, if you make any changes to an external file, you must save the file and recompile any FLA files that use it.

If you use the Check Syntax button for a script that contains `#include` statements, the syntax of the included files is also checked.

You can use `#include` in FLA files and in external script files, but not in ActionScript 2.0 class files.

You can specify no path, a relative path, or an absolute path for the file to be included. If you don't specify a path, the AS file must be in one of the following locations:

- The same directory as the FLA file
- The same directory as the script containing the `#include` statement
- The global Include directory, which is one of the following:
  - Windows 2000 or Windows XP: C:\Documents and Settings\user\Local Settings\Application Data\Macromedia\Flash MX 2004\language\Configuration\Include
  - Windows 98: C:\Windows\Application Data\Macromedia\Flash MX 2004\language\Configuration\Include
  - Macintosh OS X: Hard Drive/Users/Library/Application Support/Macromedia/Flash MX 2004/*language*/Configuration/Include
- The *Flash MX 2004 program\language*\First Run\Include directory; if you save a file here, it is copied to the global Include directory the next time you start Flash.

To specify a relative path for the AS file, use a single dot (.) to indicate the current directory, two dots (`..`) to indicate a parent directory, and forward slashes (`/`) to indicate subdirectories. See the following example section.

To specify an absolute path for the AS file, use the format supported by your platform (Macintosh or Windows). See the following example section. (This usage is not recommended because it requires the directory structure to be the same on any computer that you use to compile the script.)

Note: If you place files in the First Run/Include directory or in the global Include directory, back up these files. If you ever need to uninstall and reinstall Flash, these directories might be deleted and overwritten.

Example

The following examples show various ways of specifying a path for a file to be included in your script:

```
// Note that #include statements do not end with a semicolon (;)
// AS file is in same directory as FLA file or script
// or is in the global Include directory or the First Run/Include directory
#include "init_script.as"

// AS file is in a subdirectory of one of the above directories
// The subdirectory is named "FLA_includes"
#include "FLA_includes/init_script.as"

// AS file is in a directory at the same level as one of the above directories
// The directory is named "ALL_includes"
#include "../ALL_includes/init_script.as"

// AS file is specified by an absolute path in Windows
// Note use of forward slashes, not backslashes
#include "C:/Flash_scripts/init_script.as"

// AS file is specified by an absolute path on Macintosh
#include "Mac HD:Flash_scripts:init_script.as"
```

See also

`import`

Infinity

Availability

Flash Player 5.

Usage

```
Infinity
```

Description

Constant; specifies the IEEE-754 value representing positive infinity. The value of this constant is the same as `Number.POSITIVE_INFINITY`.

#initclip

Availability

Flash Player 6.

Usage

```
#initclip order
```

Parameters

`order` An integer that specifies the execution order of blocks of `#initclip` code. This is an optional parameter.

Description

Compiler directive; indicates the beginning of a block of initialization actions. When multiple clips are initialized at the same time, you can use the `order` parameter to specify which initialization occurs first. Initialization actions execute when a movie clip symbol is defined. If the movie clip is an exported symbol, the initialization actions execute before the actions on Frame 1 of the SWF file. Otherwise, they execute immediately before the frame actions of the frame that contains the first instance of the associated movie clip symbol.

Initialization actions execute only once when a SWF file plays; use them for one-time initializations, such as class definition and registration.

Example

In the following example, ActionScript is placed on Frame 1 inside a movie clip instance. A variables.txt text file is placed in the same directory.

```
#initclip
trace("initializing app");
var variables:LoadVars = new LoadVars();
variables.load("variables.txt");
variables.onLoad = function(success:Boolean) {
  trace("variables loaded:"+success);
  if (success) {
    for (i in variables) {
      trace("variables."+i+" = "+variables[i]);
    }
  }
};
#endinitclip
```

See also

`#endinitclip`

instanceof

Availability

Flash Player 6.

Usage

`object` `instanceof` *`class`*

Parameters

`object` An ActionScript object.

`class` A reference to an ActionScript constructor function, such as String or Date.

Returns

If *`object`* is an instance of *`class`*, `instanceof` returns `true`; `false` otherwise. Also, `_global instanceof Object` returns `false`.

Description

Operator; determines whether an object belongs to a specified class. Tests whether *`object`* is an instance of *`class`*.

The `instanceof` operator does not convert primitive types to wrapper objects. For example, the following code returns `true`:

```
new String("Hello") instanceof String;
```

The following code returns `false`:

```
"Hello" instanceof String;
```

Example

In the following example, a `for` loop is used to loop through the SWF file and trace only TextInput component instances. The instance names appear in the output.

```
for (var i in this) {
  if (this[i] instanceof mx.controls.TextInput) {
    trace("Instance \""+i+"\" is a TextInput component instance.");
  }
}
```

See also

`typeof`

interface

Availability

Flash Player 6.

Usage

```
interface InterfaceName [extends InterfaceName ] {}
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Description

Keyword; defines an interface. An interface is similar to a class, with the following important differences:

- Interfaces contain only declarations of methods, not their implementation. That is, every class that implements an interface must provide an implementation for each method declared in the interface.
- Only public members are allowed in an interface definition; instance and class members are not permitted.
- The `get` and `set` statements are not allowed in interface definitions.

For more information, see "Creating and using interfaces" in *Using ActionScript in Flash*.

Example

The following example shows several ways to define and implement interfaces:

```
(in top-level package .as files Ia, B, C, Ib, D, Ic, E)

// filename Ia.as
interface Ia
{
  function k():Number;         // method declaration only
  function n(x:Number):Number; // without implementation
}
// filename B.as
class B implements Ia
{
  function k():Number {return 25;}
  function n(x:Number):Number {return x+5;}
}
// external script or Actions panel
var mvar:B = new B();
trace(mvar.k());   // 25
trace(mvar.n(7)); // 12

// filename c.as
class C implements Ia
{
  function k():Number {return 25;}
} // error: class must implement all interface methods
```

```
// filename Ib.as
interface Ib
{
  function o():Void;
}
class D implements Ia, Ib
{
  function k():Number {return 15;}
  function n(x:Number):Number {return x*x;}
  function o():Void {trace("o");}
}

// external script or Actions panel
mvar = new D();
trace(mvar.k());   // 15
trace(mvar.n(7));  // 49
trace(mvar.o());   // "o"

interface Ic extends Ia
{
  function p():Void;
}
class E implements Ib, Ic
{
  function k():Number {return 25;}
  function n(x:Number):Number {return x+5;}
  function o():Void {trace("o");}
  function p():Void {trace("p");}
}
```

See also

`class`, `extends`, `implements`

intrinsic

Availability

Flash Player 6.

Usage

```
intrinsic class className  [ extends superClass ]
               [ implements interfaceName [, interfaceName... ] ]
{
  // class definition here
}
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Description

Keyword; allows compile-time type checking of previously defined classes. Flash uses intrinsic class declarations to enable compile-time type checking of built-in classes such as Array, Object, and String. This keyword indicates to the compiler that no function implementation is required, and that no bytecode should be generated for it.

The intrinsic keyword can also be used with variable and function declarations. Flash uses this keyword in the topLevel.as file to enable compile-time type checking for global functions and properties.

The `intrinsic` keyword was created specifically to enable compile-time type checking for built-in classes and objects, and global variables and functions. This keyword was not meant for general purpose use, but may be of some value to developers seeking to enable compile-time type checking with previously defined classes, especially if the classes are defined using ActionScript 1.0.

Example

The following example shows how to enable compile-time file checking for a previously defined ActionScript 1.0 class. The code will generate a compile-time error because the call `myCircle.setRadius()` sends a String value as a parameter instead of a Number value. You can avoid the error by changing the parameter to a Number value (for example, by changing `"10"` to `10`).

```
// The following code must be placed in a file named Circle.as
// that resides within your classpath:
intrinsic class Circle {
  var radius:Number;
  function Circle(radius:Number);
  function getArea():Number;
  function getDiameter():Number;
  function setRadius(param_radius:Number):Number;
}

// This ActionScript 1.0 class definition may be placed in your FLA file.
// Circle class is defined using ActionScript 1.0
function Circle(radius) {
  this.radius = radius;
  this.getArea = function(){
    return Math.PI*this.radius*this.radius;
  };
  this.getDiameter = function() {
    return 2*this.radius;
  };
  this.setRadius = function(param_radius) {
    this.radius = param_radius;
  }
}

// ActionScript 2.0 code that uses the Circle class
var myCircle:Circle = new Circle(5);
trace(myCircle.getArea());
trace(myCircle.getDiameter());
myCircle.setRadius("10");
trace(myCircle.radius);
trace(myCircle.getArea());
trace(myCircle.getDiameter());
```

See also

`class`, `implements`

isFinite()

Availability

Flash Player 5.

Usage

```
isFinite(expression:Object) : Boolean
```

Parameters

expression A Boolean value, variable, or other expression to be evaluated.

Returns

A Boolean value.

Description

Function; evaluates *expression* and returns `true` if it is a finite number or `false` if it is infinity or negative infinity. The presence of infinity or negative infinity indicates a mathematical error condition such as division by 0.

Example

The following example shows return values for `isFinite`:

```
isFinite(56)
// returns true

isFinite(Number.POSITIVE_INFINITY)
// returns false
```

isNaN()

Availability

Flash Player 5.

Usage

`isNaN(`*`expression:Object`*`) : `*`Boolean`*

Parameters

`expression` A Boolean, variable, or other expression to be evaluated.

Returns

A Boolean value.

Description

Function; evaluates the parameter and returns `true` if the value is `NaN` (not a number). This function is useful for checking whether a mathematical expression evaluates successfully to a number.

Example

The following code illustrates return values for the `isNaN()` function:

```
trace( isNaN("Tree") );
// returns true

trace( isNaN(56) );
// returns false

trace( isNaN(Number.POSITIVE_INFINITY) )
// returns false
```

The following example shows how you can use `isNAN()` to check whether a mathematical expression contains an error:

```
var dividend:Number;
var divisor:Number;
divisor = 1;
trace( isNaN(dividend/divisor) );
// output: true
// The output is true because the variable dividend is undefined.
// Do not use isNAN() to check for division by 0 because it will return false.
// A positive number divided by 0 equals Infinity (Number.POSITIVE_INFINITY).
// A negative number divided by 0 equals -Infinity (Number.NEGATIVE_INFINITY).
```

See also

`NaN`, `Number.NaN`

Key class

Availability

Flash Player 6.

Description

The Key class is a top-level class whose methods and properties you can use without using a constructor. Use the methods of the Key class to build an interface that can be controlled by a user with a standard keyboard. The properties of the Key class are constants representing the keys most commonly used to control games. For a complete list of key code values, see Appendix C, "Keyboard Keys and Key Code Values" in *Using ActionScript in Flash*.

Method summary for the Key class

Method	Description
`Key.addListener()`	Registers an object to receive notification when the `onKeyDown` and `onKeyUp` methods are invoked.
`Key.getAscii()`	Returns the ASCII value of the last key pressed.
`Key.getCode()`	Returns the virtual key code of the last key pressed.
`Key.isDown()`	Returns `true` if the key specified in the parameter is pressed.
`Key.isToggled()`	Returns `true` if the Num Lock or Caps Lock key is activated.
`Key.removeListener()`	Removes an object that was previously registered with `Key.addListener()`.

Property summary for the Key class

All the properties for the Key class are constants.

Property	Description	Key code
`Key.BACKSPACE`	The key code value for the Backspace key.	8
`Key.CAPSLOCK`	The key code value for the Caps Lock key.	20
`Key.CONTROL`	The key code value for the Control key.	17
`Key.DELETEKEY`	The key code value for the Delete key.	46
`Key.DOWN`	The key code value for the Down Arrow key.	40
`Key.END`	The key code value for the End key.	35
`Key.ENTER`	The key code value for the Enter key.	13
`Key.ESCAPE`	The key code value for the Escape key.	27
`Key.HOME`	The key code value for the Home key.	36
`Key.INSERT`	The key code value for the Insert key.	45
`Key.LEFT`	The key code value for the Left Arrow key.	37
`Key.PGDN`	The key code value for the Page Down key.	34

Property	Description	Key code
`Key.PGUP`	The key code value for the Page Up key.	33
`Key.RIGHT`	The key code value for the Right Arrow key.	39
`Key.SHIFT`	The key code value for the Shift key.	16
`Key.SPACE`	The key code value for the Spacebar.	32
`Key.TAB`	The key code value for the Tab key.	9
`Key.UP`	The key code value for the Up Arrow key.	38

Listener summary for the Key class

Method	Description
`Key.onKeyDown`	Notified when a key is pressed.
`Key.onKeyUp`	Notified when a key is released.

Key.addListener()

Availability

Flash Player 6.

Usage

`Key.addListener (`*`newListener:Object`*`) : `*`Void`*

Parameters

`newListener`	An object with methods `onKeyDown` and `onKeyUp`.

Returns

Nothing.

Description

Method; registers an object to receive `onKeyDown` and `onKeyUp` notification. When a key is pressed or released, regardless of the input focus, all listening objects registered with `addListener()` have either their `onKeyDown` method or `onKeyUp` method invoked. Multiple objects can listen for keyboard notifications. If the listener *`newListener`* is already registered, no change occurs.

Example

The following example creates a new listener object and defines a function for `onKeyDown` and `onKeyUp`. The last line uses `addListener()` to register the listener with the Key object so that it can receive notification from the key down and key up events.

```
var myListener:Object = new Object();
myListener.onKeyDown = function () {
  trace ("You pressed a key.");
}
myListener.onKeyUp = function () {
  trace ("You released a key.");
}
Key.addListener(myListener);
```

The following example assigns the keyboard shortcut Control+7 to a button with an instance name of `my_btn` and makes information about the shortcut available to screen readers (see _accProps). In this example, when you press Control+7 the `myOnPress` function displays the text `hello` in the Output panel.

```
function myOnPress() {
  trace("hello");
}
function myOnKeyDown() {
  // 55 is key code for 7
  if (Key.isDown(Key.CONTROL) && Key.getCode() == 55) {
    Selection.setFocus(my_btn);
    my_btn.onPress();
  }
}
var myListener:Object = new Object();
```

```
myListener.onKeyDown = myOnKeyDown;
Key.addListener(myListener);
my_btn.onPress = myOnPress;
my_btn._accProps.shortcut = "Ctrl+7";
Accessibility.updateProperties();
```

See also

`Key.getCode()`, `Key.isDown()`, `Key.onKeyDown`, `Key.onKeyUp`, `Key.removeListener()`

Key.BACKSPACE

Availability

Flash Player 5.

Usage

`Key.BACKSPACE:Number`

Description

Property; constant associated with the key code value for the Backspace key (8).

Example

The following example creates a new listener object and defines a function for `onKeyDown`. The last line uses `addListener()` to register the listener with the Key object so that it can receive notification from the key down event.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  if (Key.isDown(Key.BACKSPACE)) {
    trace("you pressed the Backspace key.");
  } else {
    trace("you DIDN'T press the Backspace key.");
  }
};
Key.addListener(keyListener);
```

When using this example, make sure that you select Control > Disable Keyboard Shortcuts in the test environment.

Key.CAPSLOCK

Availability

Flash Player 5.

Usage

`Key.CAPSLOCK:`*`Number`*

Description

Property; constant associated with the key code value for the Caps Lock key (20).

Example

The following example creates a new listener object and defines a function for `onKeyDown`. The last line uses `addListener()` to register the listener with the Key object so that it can receive notification from the key down event.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  if (Key.isDown(Key.CAPSLOCK)) {
    trace("you pressed the Caps Lock key.");
    trace("\tCaps Lock == "+Key.isToggled(Key.CAPSLOCK));
  }
};
Key.addListener(keyListener);
```

Information displays in the Output panel when you press the Caps Lock key. The Output panel displays either `true` or `false`, depending on whether the Caps Lock is activated using the isToggled method.

Key.CONTROL

Availability

Flash Player 5.

Usage

```
Key.CONTROL:Number
```

Description

Property; constant associated with the key code value for the Control key (17).

Example

The following example assigns the keyboard shortcut Control+7 to a button with an instance name of `my_btn` and makes information about the shortcut available to screen readers (see _accProps). In this example, when you press Control+7 the `myOnPress` function displays the text `hello` in the Output panel.

```
function myOnPress() {
  trace("hello");
}
function myOnKeyDown() {
  // 55 is key code for 7
  if (Key.isDown(Key.CONTROL) && Key.getCode() == 55) {
    Selection.setFocus(my_btn);
    my_btn.onPress();
  }
}
var myListener:Object = new Object();
myListener.onKeyDown = myOnKeyDown;
Key.addListener(myListener);
my_btn.onPress = myOnPress;
my_btn._accProps.shortcut = "Ctrl+7";
Accessibility.updateProperties();
```

Key.DELETEKEY

Availability

Flash Player 5.

Usage

`Key.DELETEKEY:`*`Number`*

Description

Property; constant associated with the key code value for the Delete key (46).

Example

The following example lets you draw lines with the mouse pointer using the Drawing API and listener objects. Press the Backspace or Delete key to remove the lines that you draw.

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
  this.drawing = true;
  canvas_mc.moveTo(_xmouse, _ymouse);
  canvas_mc.lineStyle(3, 0x99CC00, 100);
};
mouseListener.onMouseUp = function() {
  this.drawing = false;
};
mouseListener.onMouseMove = function() {
  if (this.drawing) {
    canvas_mc.lineTo(_xmouse, _ymouse);
  }
  updateAfterEvent();
};
Mouse.addListener(mouseListener);
//
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  if (Key.isDown(Key.DELETEKEY) || Key.isDown(Key.BACKSPACE)) {
    canvas_mc.clear();
  }
};
Key.addListener(keyListener);
```

When using this example, make sure that you select Control > Disable Keyboard Shortcuts in the test environment.

Key.DOWN

Availability

Flash Player 5.

Usage

`Key.DOWN:`*`Number`*

Description

Property; constant associated with the key code value for the Down Arrow key (40).

Example

The following example moves a movie clip called `car_mc` a constant distance (10) when you press the arrow keys. A sound plays when you press the Spacebar. Give a sound in the library a linkage identifier of `horn_id` for this example.

```
var DISTANCE:Number = 10;
var horn_sound:Sound = new Sound();
horn_sound.attachSound("horn_id");
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
  switch (Key.getCode()) {
  case Key.SPACE :
    horn_sound.start();
    break;
  case Key.LEFT :
    car_mc._x -= DISTANCE;
    break;
  case Key.UP :
    car_mc._y -= DISTANCE;
    break;
  case Key.RIGHT :
    car_mc._x += DISTANCE;
    break;
  case Key.DOWN :
    car_mc._y += DISTANCE;
    break;
  }
};
Key.addListener(keyListener_obj);
```

Key.END

Availability

Flash Player 5.

Usage

```
Key.END:Number
```

Description

Property; constant associated with the key code value for the End key (35).

Key.ENTER

Availability

Flash Player 5.

Usage

`Key.ENTER:Number`

Description

Property; constant associated with the key code value for the Enter key (13).

Example

The following example moves a movie clip called `car_mc` a constant distance (10) when you press the arrow keys. The `car_mc` instance stops when you press Enter and delete the `onEnterFrame` event.

```
var DISTANCE:Number = 5;
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  switch (Key.getCode()) {
  case Key.LEFT :
    car_mc.onEnterFrame = function() {
      this._x -= DISTANCE;
    };
    break;
  case Key.UP :
    car_mc.onEnterFrame = function() {
      this._y -= DISTANCE;
    };
    break;
  case Key.RIGHT :
    car_mc.onEnterFrame = function() {
      this._x += DISTANCE;
    };
    break;
  case Key.DOWN :
    car_mc.onEnterFrame = function() {
      this._y += DISTANCE;
    };
    break;
  case Key.ENTER :
    delete car_mc.onEnterFrame;
    break;
  }
};
Key.addListener(keyListener);
```

When using this example, make sure that you select Control > Disable Keyboard Shortcuts in the test environment.

Key.ESCAPE

Availability

Flash Player 5.

Usage

`Key.ESCAPE:Number`

Description

Property; constant associated with the key code value for the Escape key (27).

Example

The following example sets a timer. When you press Escape, the Output panel displays information that includes how long it took you to press the key.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  if (Key.isDown(Key.ESCAPE)) {
    // get the current timer, convert the value to seconds and round it to two
  decimal places.
    var timer:Number = Math.round(getTimer()/10)/100;
    trace("you pressed the Esc key: "+getTimer()+" ms ("+timer+" s)");
  }
};
Key.addListener(keyListener);
```

When using this example, make sure that you select Control > Disable Keyboard Shortcuts in the test environment.

Key.getAscii()

Availability

Flash Player 5.

Usage

```
Key.getAscii() : Number
```

Parameters

None.

Returns

A number; an integer that represents the ASCII value of the last key pressed.

Description

Method; returns the ASCII code of the last key pressed or released. The ASCII values returned are English keyboard values. For example, if you press Shift+2, `Key.getAscii()` returns `@` on a Japanese keyboard, which is the same as it does on an English keyboard.

Example

The following example calls the `getAscii()` method any time a key is pressed. The example creates a listener object named `keyListener` and defines a function that responds to the `onKeyDown` event by calling `Key.getAscii()`. For more information, see "Using event listeners" in *Using ActionScript in Flash*. The `keyListener` object is then registered to the `Key` object, which broadcasts the `onKeyDown` message whenever a key is pressed while the SWF file plays.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  trace("The ASCII code for the last key typed is: "+Key.getAscii());
};
Key.addListener(keyListener);
```

When using this example, make sure that you select Control > Disable Keyboard Shortcuts in the test environment.

The following example adds a call to `Key.getAscii()` to show how the two methods differ. The main difference is that `Key.getAscii()` differentiates between uppercase and lowercase letters, and `Key.getCode()` does not.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  trace("For the last key typed:");
  trace("\tThe Key code is: "+Key.getCode());
  trace("\tThe ASCII value is: "+Key.getAscii());
  trace("");
};
Key.addListener(keyListener);
```

When using this example, make sure that you select Control > Disable Keyboard Shortcuts in the test environment.

Key.getCode()

Availability

Flash Player 5.

Usage

```
Key.getCode() : Number
```

Parameters

None.

Returns

A number; an integer that represents the key code of the last key pressed.

Description

Method; returns the key code value of the last key pressed. To match the returned key code value with the key on a standard keyboard, see Appendix C, "Keyboard Keys and Key Code Values". in *Using ActionScript in Flash.*

Example

The following example calls the `getCode()` method any time a key is pressed. The example creates a listener object named `keyListener` and defines a function that responds to the `onKeyDown` event by calling `Key.getCode()`. For more information, see "Using event listeners" in *Using ActionScript in Flash*. The `keyListener` object is then registered to the `Key` object, which broadcasts the `onKeyDown` message whenever a key is pressed while the SWF file plays.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  trace("The ASCII code for the last key typed is: "+Key.getAscii());
};
Key.addListener(keyListener);
```

When using this example, make sure that you select Control > Disable Keyboard Shortcuts in the test environment.

The following example adds a call to `Key.getAscii()` to show how the two methods differ. The main difference is that `Key.getAscii()` differentiates between uppercase and lowercase letters, and `Key.getCode()` does not.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  trace("For the last key typed:");
  trace("\tThe Key code is: "+Key.getCode());
  trace("\tThe ASCII value is: "+Key.getAscii());
  trace("");
};
Key.addListener(keyListener);
```

When using this example, make sure that you select Control > Disable Keyboard Shortcuts in the test environment.

Key.HOME

Availability

Flash Player 5.

Usage

`Key.HOME:Number`

Description

Property; constant associated with the key code value for the Home key (36).

Example

The following example attaches a draggable movie clip called `car_mc` at the *x* and *y* coordinates of 0,0. When you press the Home key, car_mc returns to 0,0. Create a movie clip that has a linkage ID `car_id`, and add the following ActionScript to Frame 1 of the Timeline:

```
this.attachMovie("car_id", "car_mc", this.getNextHighestDepth(), {_x:0,
  _y:0});
car_mc.onPress = function() {
  this.startDrag();
};
car_mc.onRelease = function() {
  this.stopDrag();
};
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  if (Key.isDown(Key.HOME)) {
    car_mc._x = 0;
    car_mc._y = 0;
  }
};
Key.addListener(keyListener);
```

Key.INSERT

Availability

Flash Player 5.

Usage

`Key.INSERT:Number`

Description

Property; constant associated with the key code value for the Insert key (45).

Example

The following example creates a new listener object and defines a function for `onKeyDown`. The last line uses `addListener()` to register the listener with the Key object so that it can receive notification from the key down event and display information in the Output panel.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  if (Key.isDown(Key.INSERT)) {
    trace("You pressed the Insert key.");
  }
};
Key.addListener(keyListener);
```

Key.isDown()

Availability

Flash Player 5.

Usage

`Key.isDown(`*`keycode:Number`*`) :` *`Boolean`*

Parameters

`keycode` The key code value assigned to a specific key or a Key class property associated with a specific key. To match the returned key code value with the key on a standard keyboard, see Appendix C, "Keyboard Keys and Key Code Values" in *Using ActionScript in Flash.*

Returns

A Boolean value.

Description

Method: returns `true` if the key specified in *`keycode`* is pressed; `false` otherwise. On the Macintosh, the key code values for the Caps Lock and Num Lock keys are identical.

Example

The following script lets the user control a movie clip's (`car_mc`) location:

```
car_mc.onEnterFrame = function() {
  if (Key.isDown(Key.RIGHT)) {
    this._x += 10;
  } else if (Key.isDown(Key.LEFT)) {
    this._x -= 10;
  }
};
```

Key.isToggled()

Availability

Flash Player 5.

Usage

```
Key.isToggled(keycode:Number) : Boolean
```

Parameters

`keycode` The key code for the Caps Lock key (20) or the Num Lock key (144).

Returns

A Boolean value.

Description

Method: returns `true` if the Caps Lock or Num Lock key is activated (toggled to an active state); `false` otherwise. Although the term *toggled* usually means that something is switched between two options, the method Key.isToggled() will only return `true` if the key is toggled to an active state. On the Macintosh, the key code values for the Caps Lock and Num Lock keys are identical.

Example

The following example calls the `isToggled()` method any time a key is pressed and executes a trace statement any time the Caps Lock key is toggled to an active state. The example creates a listener object named `keyListener` and defines a function that responds to the `onKeyDown` event by calling `Key.isToggled()`. For more information, see "Using event listeners" in *Using ActionScript in Flash*. The `keyListener` object is then registered to the `Key` object, which broadcasts the `onKeyDown` message whenever a key is pressed while the SWF file plays.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  if (Key.isDown(Key.CAPSLOCK)) {
    trace("you pressed the Caps Lock key.");
    trace("\tCaps Lock == "+Key.isToggled(Key.CAPSLOCK));
  }
};
Key.addListener(keyListener);
```

Information displays in the Output panel when you press the Caps Lock key. The Output panel displays either `true` or `false`, depending on whether the Caps Lock is activated using the isToggled method.

The following example creates two text fields that update when the Caps Lock and Num Lock keys are toggled. Each text field displays true when the key is activated, and false when the key is deactivated.

```
this.createTextField("capsLock_txt", this.getNextHighestDepth(), 0, 0, 100,
  22);
capsLock_txt.autoSize = true;
capsLock_txt.html = true;
this.createTextField("numLock_txt", this.getNextHighestDepth(), 0, 22, 100,
  22);
```

```
numLock_txt.autoSize = true;
numLock_txt.html = true;
//
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  capsLock_txt.htmlText = "<b>Caps Lock:</b> "+Key.isToggled(Key.CAPSLOCK);
  numLock_txt.htmlText = "<b>Num Lock:</b> "+Key.isToggled(144);
};
Key.addListener(keyListener);
```

Key.LEFT

Availability

Flash Player 5.

Usage

`Key.LEFT:Number`

Description

Property; constant associated with the key code value for the Left Arrow key (37).

Example

The following example moves a movie clip called `car_mc` a constant distance (10) when you press the arrow keys. A sound plays when you press the Spacebar. Give a sound in the library a linkage identifier of `horn_id` for this example.

```
var DISTANCE:Number = 10;
var horn_sound:Sound = new Sound();
horn_sound.attachSound("horn_id");
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
  switch (Key.getCode()) {
  case Key.SPACE :
    horn_sound.start();
    break;
  case Key.LEFT :
    car_mc._x -= DISTANCE;
    break;
  case Key.UP :
    car_mc._y -= DISTANCE;
    break;
  case Key.RIGHT :
    car_mc._x += DISTANCE;
    break;
  case Key.DOWN :
    car_mc._y += DISTANCE;
    break;
  }
};
Key.addListener(keyListener_obj);
```

Key.onKeyDown

Availability

Flash Player 6.

Usage

`*keyListener*.onKeyDown`

Description

Listener; notified when a key is pressed. To use `onKeyDown`, you must create a listener object. You can then define a function for `onKeyDown` and use `addListener()` to register the listener with the Key object, as shown in the following example:

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  trace("DOWN -> Code: "+Key.getCode()+"\tACSII: "+Key.getAscii()+"\tKey:
  "+chr(Key.getAscii()));
};
keyListener.onKeyUp = function() {
  trace("UP -> Code: "+Key.getCode()+"\tACSII: "+Key.getAscii()+"\tKey:
  "+chr(Key.getAscii()));
};
Key.addListener(keyListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

See also

`Key.addListener()`

Key.onKeyUp

Availability

Flash Player 6.

Usage

keyListener.onKeyUp

Description

Listener; notified when a key is released. To use onKeyUp, you must create a listener object. You can then define a function for onKeyUp and use addListener() to register the listener with the Key object, as shown in the following example:

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  trace("DOWN -> Code: "+Key.getCode()+"\tACSII: "+Key.getAscii()+"\tKey:
  "+chr(Key.getAscii()));
};
keyListener.onKeyUp = function() {
  trace("UP -> Code: "+Key.getCode()+"\tACSII: "+Key.getAscii()+"\tKey:
  "+chr(Key.getAscii()));
};
Key.addListener(keyListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

See also

Key.addListener()

Key.PGDN

Availability

Flash Player 5.

Usage

`Key.PGDN:Number`

Description

Property; constant associated with the key code value for the Page Down key (34).

Example

The following example rotates a movie clip called `car_mc` when you press the Page Down and Page Up keys.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  if (Key.isDown(Key.PGDN)) {
    car_mc._rotation += 5;
  } else if (Key.isDown(Key.PGUP)) {
    car_mc._rotation -= 5;
  }
};
Key.addListener(keyListener);
```

Key.PGUP

Availability

Flash Player 5.

Usage

`Key.PGUP:Number`

Description

Property; constant associated with the key code value for the Page Up key (33).

Example

The following example rotates a movie clip called `car_mc` when you press the Page Down and Page Up keys.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  if (Key.isDown(Key.PGDN)) {
    car_mc._rotation += 5;
  } else if (Key.isDown(Key.PGUP)) {
    car_mc._rotation -= 5;
  }
};
Key.addListener(keyListener);
```

Key.removeListener()

Availability

Flash Player 6.

Usage

`Key.removeListener (listener:Object) : Boolean`

Parameters

listener An object.

Returns

If the *listener* was successfully removed, the method returns `true`. If the *listener* was not successfully removed (for example, because the *listener* was not on the Key object's listener list), the method returns `false`.

Description

Method; removes an object previously registered with Key.addListener().

Example

The following example moves a movie clip called `car_mc` using the Left and Right arrow keys. The listener is removed when you press Escape, and `car_mc` no longer moves.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  switch (Key.getCode()) {
  case Key.LEFT :
    car_mc._x -= 10;
    break;
  case Key.RIGHT :
    car_mc._x += 10;
    break;
  case Key.ESCAPE :
    Key.removeListener(keyListener);
  }
};
Key.addListener(keyListener);
```

Key.RIGHT

Availability

Flash Player 5.

Usage

```
Key.RIGHT:Number
```

Description

Property; constant associated with the key code value for the Right Arrow key (39).

Example

The following example moves a movie clip called `car_mc` a constant distance (10) when you press the arrow keys. A sound plays when you press the Spacebar. Give a sound in the library a linkage identifier of `horn_id` for this example.

```
var DISTANCE:Number = 10;
var horn_sound:Sound = new Sound();
horn_sound.attachSound("horn_id");
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
  switch (Key.getCode()) {
  case Key.SPACE :
    horn_sound.start();
    break;
  case Key.LEFT :
    car_mc._x -= DISTANCE;
    break;
  case Key.UP :
    car_mc._y -= DISTANCE;
    break;
  case Key.RIGHT :
    car_mc._x += DISTANCE;
    break;
  case Key.DOWN :
    car_mc._y += DISTANCE;
    break;
  }
};
Key.addListener(keyListener_obj);
```

Key.SHIFT

Availability

Flash Player 5.

Usage

`Key.SHIFT:Number`

Description

Property; constant associated with the key code value for the Shift key (16).

Example

The following example scales `car_mc` when you press Shift.

```
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  if (Key.isDown(Key.SHIFT)) {
    car_mc._xscale *= 2;
    car_mc._yscale *= 2;
  } else if (Key.isDown(Key.CONTROL)) {
    car_mc._xscale /= 2;
    car_mc._yscale /= 2;
  }
};
Key.addListener(keyListener);
```

Key.SPACE

Availability

Flash Player 5.

Usage

`Key.SPACE:Number`

Description

Property; constant associated with the key code value for the Spacebar (32).

Example

The following example moves a movie clip called `car_mc` a constant distance (10) when you press the arrow keys. A sound plays when you press the Spacebar. Give a sound in the library a linkage identifier of `horn_id` for this example.

```
var DISTANCE:Number = 10;
var horn_sound:Sound = new Sound();
horn_sound.attachSound("horn_id");
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
  switch (Key.getCode()) {
  case Key.SPACE :
    horn_sound.start();
    break;
  case Key.LEFT :
    car_mc._x -= DISTANCE;
    break;
  case Key.UP :
    car_mc._y -= DISTANCE;
    break;
  case Key.RIGHT :
    car_mc._x += DISTANCE;
    break;
  case Key.DOWN :
    car_mc._y += DISTANCE;
    break;
  }
};
Key.addListener(keyListener_obj);
```

Key.TAB

Availability

Flash Player 5.

Usage

```
Key.TAB:Number
```

Description

Property; constant associated with the key code value for the Tab key (9).

Example

The following example creates a text field, and displays the date in the text field when you press Tab.

```
this.createTextField("date_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
date_txt.autoSize = true;
var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  if (Key.isDown(Key.TAB)) {
    var today_date:Date = new Date();
    date_txt.text = today_date.toString();
  }
};
Key.addListener(keyListener);
```

When using this example, make sure that you select Control > Disable Keyboard Shortcuts in the test environment.

Key.UP

Availability

Flash Player 5.

Usage

`Key.UP:`*`Number`*

Description

Property; constant associated with the key code value for the Up Arrow key (38).

Example

The following example moves a movie clip called `car_mc` a constant distance (10) when you press the arrow keys. A sound plays when you press the Spacebar. Give a sound in the library a linkage identifier of `horn_id` for this example.

```
var DISTANCE:Number = 10;
var horn_sound:Sound = new Sound();
horn_sound.attachSound("horn_id");
var keyListener_obj:Object = new Object();
keyListener_obj.onKeyDown = function() {
  switch (Key.getCode()) {
  case Key.SPACE :
    horn_sound.start();
    break;
  case Key.LEFT :
    car_mc._x -= DISTANCE;
    break;
  case Key.UP :
    car_mc._y -= DISTANCE;
    break;
  case Key.RIGHT :
    car_mc._x += DISTANCE;
    break;
  case Key.DOWN :
    car_mc._y += DISTANCE;
    break;
  }
};
Key.addListener(keyListener_obj);
```

_level

Availability

Flash Player 4.

Usage

`_levelN`

Description

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

The initial SWF file loaded into an instance of the Flash Player is automatically loaded into `_level0`. The SWF file in `_level0` sets the frame rate, background color, and frame size for all subsequently loaded SWF files. SWF files are then stacked in higher-numbered levels above the SWF file in `_level0`.

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

Example

The following example stops the playhead in the main Timeline of the SWF file sub.swf that is loaded into `_level9`. The sub.swf file contains animation and is in the same directory as the document that contains the following ActionScript:

```
loadMovieNum("sub.swf", 9);
myBtn_btn.onRelease = function() {
  _level9.stop();
};
```

You could replace `_level9.stop()` in the previous example with the following code:

```
_level9.gotoAndStop(5);
```

This action sends the playhead in the main Timeline of the SWF file in `_level9` to Frame 5 instead of stopping the playhead.

See also

`loadMovie()`, `MovieClip.swapDepths()`

loadMovie()

Availability

Flash Player 3.

Usage

```
loadMovie(url:String,target:Object [, method:String]) : Void
loadMovie(url:String,target:String [, method:String]) : Void
```

Parameters

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

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

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

Returns

Nothing.

Description

Function; loads a SWF or JPEG file into Flash Player while the original SWF file plays.

Tip: If you want to monitor the progress of the download, use MovieClipLoader.loadClip() instead of this function.

The `loadMovie()` function lets you display several SWF files at once and switch among SWF files without loading another HTML document. Without the `loadMovie()` function, Flash Player displays a single SWF file.

If you want to load a SWF or JPEG file into a specific level, use `loadMovieNum()` instead of `loadMovie()`.

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

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

Example

Usage 1: The following example loads the SWF file circle.swf from the same directory and replaces a movie clip called `mySquare` that already exists on the Stage:

```
loadMovie("circle.swf", mySquare);
// equivalent statement (Usage 1): loadMovie("circle.swf", _level0.mySquare);
// equivalent statement (Usage 2): loadMovie("circle.swf", “mySquare”);
```

The following example loads the SWF file circle.swf from the same directory, but replaces the main movie clip instead of the `mySquare` movie clip:

```
loadMovie("circle.swf", this);
// Note that using "this" as a string for the target parameter will not work
// equivalent statement (Usage 2): loadMovie("circle.swf", “_level0”);
```

The following `loadMovie()` statement loads the SWF file sub.swf from the same directory into a new movie clip called `logo_mc` that's created using `createEmptyMovieClip()`:

```
this.createEmptyMovieClip("logo_mc", 999);
loadMovie("sub.swf", logo_mc);
```

You could add the following code to load a JPEG image called image1.jpg from the same directory as the SWF file loading sub.swf. The JPEG is loaded when you click a button called `myBtn_btn`. This code loads the JPEG into `logo_mc`. Therefore, it will replace sub.swf with the JPEG image.

```
myBtn_btn.onRelease = function(){
  loadMovie("image1.jpg", logo_mc);
};
```

Usage 2: The following example loads the SWF file circle.swf from the same directory and replaces a movie clip called `mySquare` that already exists on the Stage:

```
loadMovie("circle.swf", “mySquare”);
```

See also

`_level`, `loadMovieNum()`, MovieClip.loadMovie(), MovieClipLoader.loadClip(), `unloadMovie()`

loadMovieNum()

Availability

Flash Player 4. Flash 4 files opened in Flash 5 or later are converted to use the correct syntax.

Usage

```
loadMovieNum(url:String,level:Number [, variables:String]) : Void
```

Parameters

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

`level` An integer specifying the level in Flash Player into which the SWF file will load.

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

Returns

Nothing.

Description

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

Tip: If you want to monitor the progress of the download, use MovieClipLoader.loadClip() instead of this function.

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

If you want to specify a target instead of a level, use loadMovie() instead of `loadMovieNum()`.

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

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

The `loadMovieNum()` action also lets you load JPEG files into a SWF file while it plays. For images and SWF files, the upper left corner of the image aligns with the upper left corner of the Stage when the file loads. Also in both cases, the loaded file inherits rotation and scaling, and the original content is overwritten in the specified level.

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

Example

The following example loads the JPEG image tim.jpg into level 2 of Flash Player:

```
loadMovieNum("http://www.macromedia.com/devnet/mx/coldfusion/articles/
  basic_chart/tim.jpg", 2);
```

See also

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

loadVariables()

Availability

Flash Player 4; behavior changed in Flash Player 7.

Usage

```
loadVariables (url:String , target:Object [, variables:String]) : Void
```

Parameters

`url` An absolute or relative URL where the variables are located. If the SWF file issuing this call is running in a web browser, `url` must be in the same domain as the SWF file; for details, see the Description section.

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

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

Returns

Nothing.

Description

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

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

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

In SWF files running in a version earlier than Flash Player 7, `url` must be in the same superdomain as the SWF file that is issuing this call. A superdomain is derived by removing the leftmost component of a file's URL. For example, a SWF file at www.someDomain.com can load data from a source at store.someDomain.com because both files are in the same superdomain of someDomain.com.

In SWF files of any version running in Flash Player 7 or later, `url` must be in exactly the same domain as the SWF file that is issuing this call (see "Flash Player security features" in *Using ActionScript in Flash*). For example, a SWF file at www.someDomain.com can load data only from sources that are also at www.someDomain.com. If you want to load data from a different domain, you can place a *cross-domain policy file* on the server hosting the SWF file that is being accessed. For more information, see "About allowing cross-domain data loading" in *Using ActionScript in Flash*.

If you want to load variables into a specific level, use loadVariablesNum() instead of loadVariables().

Example

The following example loads information from a text file called params.txt into the target_mc movie clip that is created using createEmptyMovieClip(). The setInterval() function is used to check the loading progress. The script checks for a variable in the params.txt file named done.

```
this.createEmptyMovieClip("target_mc", this.getNextHighestDepth());
loadVariables("params.txt", target_mc);
function checkParamsLoaded() {
  if (target_mc.done == undefined) {
    trace("not yet.");
  } else {
    trace("finished loading. killing interval.");
    trace("-------------");
    for (i in target_mc) {
      trace(i+": "+target_mc[i]);
    }
    trace("-------------");
    clearInterval(param_interval);
  }
}
var param_interval = setInterval(checkParamsLoaded, 100);
// params.txt includes the following text
var1="hello"&var2="goodbye"&done="done"
```

See also

loadVariablesNum(), loadMovie(), loadMovieNum(), getURL(), MovieClip.loadMovie(), MovieClip.loadVariables(), LoadVars.load()

loadVariablesNum()

Availability

Flash Player 4; behavior changed in Flash Player 7. Flash 4 files opened in Flash 5 or later are converted to use the correct syntax.

Usage

```
loadVariablesNum (url:String ,level:Number [, variables:String]) : Void
```

Parameters

`url` An absolute or relative URL where the variables are located. If the SWF file issuing this call is running in a web browser, `url` must be in the same domain as the SWF file; for details, see the Description section.

`level` An integer specifying the level in Flash Player to receive the variables.

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

Returns

Nothing.

Description

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

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

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

In SWF files running in a version of the player earlier than Flash Player 7, `url` must be in the same superdomain as the SWF file that is issuing this call. A superdomain is derived by removing the leftmost component of a file's URL. For example, a SWF file at www.someDomain.com can load data from a source at store.someDomain.com, because both files are in the same superdomain of someDomain.com.

In SWF files of any version running in Flash Player 7 or later, `url` must be in exactly the same domain as the SWF file that is issuing this call (see "Flash Player security features" in *Using ActionScript in Flash*). For example, a SWF file at www.someDomain.com can load data only from sources that are also at www.someDomain.com. If you want to load data from a different domain, you can place a *cross-domain policy file* on the server hosting the SWF file. For more information, see "About allowing cross-domain data loading" in *Using ActionScript in Flash*.

If you want to load variables into a target MovieClip, use `loadVariables()` instead of `loadVariablesNum()`.

Example

The following example loads information from a text file called params.txt into the main Timeline of the SWF at level 2 in Flash Player. The variable names of the text fields must match the variable names in the params.txt file. The `setInterval()` function is used to check the progress of the data being loaded into the SWF. The script checks for a variable in the params.txt file named `done`.

```
loadVariablesNum("params.txt", 2);
function checkParamsLoaded() {
  if (_level2.done == undefined) {
    trace("not yet.");
  } else {
    trace("finished loading. killing interval.");
    trace("-------------");
    for (i in _level2) {
      trace(i+": "+_level2[i]);
    }
    trace("-------------");
    clearInterval(param_interval);
  }
}
var param_interval = setInterval(checkParamsLoaded, 100);

// Params.txt includes the following text
var1="hello"&var2="goodbye"&done="done"
```

See also

`getURL()`, `loadMovie()`, `loadMovieNum()`, `loadVariables()`, `MovieClip.loadMovie()`, `MovieClip.loadVariables()`, `LoadVars.load()`

LoadVars class

Availability

Flash Player 6.

Description

The LoadVars class is an alternative to the `loadVariables()` function for transferring variables between a Flash application and a server.

You can use the LoadVars class to obtain verification of successful data loading and to monitor download progress. The LoadVars class lets you send all the variables in an object to a specified URL and load all the variables at a specified URL into an object. It also lets you send specific variables, rather than all the variables, which can make your application more efficient. You can use the `LoadVars.onLoad` handler to ensure that your application runs when data is loaded, and not before.

The LoadVars class works much like the XML class; it uses the methods `load()`, `send()`, and `sendAndLoad()` to communicate with a server. The main difference between the LoadVars class and the XML class is that LoadVars transfers ActionScript name and value pairs, rather than an XML DOM tree stored in the XML object. The LoadVars class follows the same security restrictions as the XML class.

For information about using the LoadVars class and example code, see "Using the LoadVars class" in *Using ActionScript in Flash*.

Method summary for the LoadVars class

Method	Description
`LoadVars.addRequestHeader()`	Adds or changes HTTP headers for `POST` operations.
`LoadVars.decode()`	Converts a variable string to properties of the specified LoadVars object.
`LoadVars.getBytesLoaded()`	Returns the number of bytes downloaded by `LoadVars.load()` or `LoadVars.sendAndLoad()`.
`LoadVars.getBytesTotal()`	Returns the total number of bytes that will be downloaded by a `load` or `sendAndLoad` method.
`LoadVars.load()`	Downloads variables from a specified URL.
`LoadVars.send()`	Posts variables from a LoadVars object to a URL.
`LoadVars.sendAndLoad()`	Posts variables from a LoadVars object to a URL and downloads the server's response to a target object.
`LoadVars.toString()`	Returns a URL-encoded string that contains all the enumerable variables in the LoadVars object.

Property summary for the LoadVars class

Property	Description
`LoadVars.contentType`	A string that indicates the MIME type of the data.
`LoadVars.loaded`	A Boolean value that indicates whether a `load` or `sendAndLoad` operation has completed.

Event handler summary for the LoadVars class

Event handler	Description
`LoadVars.onData`	Invoked when data has been completely downloaded from the server, or when an error occurs while data is downloading from a server.
`LoadVars.onLoad`	Invoked when a `load` or `sendAndLoad` operation has completed.

Constructor for the LoadVars class

Availability

Flash Player 6.

Usage

```
new LoadVars() : LoadVars
```

Parameters

None.

Returns

A reference to a LoadVars object.

Description

Constructor; creates a LoadVars object. You can then use the methods of that LoadVars object to send and load data.

Example

The following example creates a LoadVars object called `my_lv`:

```
var my_lv:LoadVars = new LoadVars();
```

LoadVars.addRequestHeader()

Availability

Flash Player 6.

Usage

```
my_lv.addRequestHeader(headerName:String, headerValue:String) : Void
my_lv.addRequestHeader(["headerName_1":String, "headerValue_1" ...
  "headerName_n", "headerValue_n":String]) : Void
```

Parameters

headerName A string that represents an HTTP request header name.

headerValue A string that represents the value associated with *headerName*.

Returns

Nothing.

Description

Method; adds or changes HTTP request headers (such as `Content-Type` or `SOAPAction`) sent with `POST` actions. In the first usage, you pass two strings to the method: *headerName* and *headerValue*. In the second usage, you pass an array of strings, alternating header names and header values.

If multiple calls are made to set the same header name, each successive value will replace the value set in the previous call.

The following standard HTTP headers *cannot* be added or changed with this method: `Accept-Ranges`, `Age`, `Allow`, `Allowed`, `Connection`, `Content-Length`, `Content-Location`, `Content-Range`, `ETag`, `Host`, `Last-Modified`, `Locations`, `Max-Forwards`, `Proxy-Authenticate`, `Proxy-Authorization`, `Public`, `Range`, `Retry-After`, `Server`, `TE`, `Trailer`, `Transfer-Encoding`, `Upgrade`, `URI`, `Vary`, `Via`, `Warning`, and `WWW-Authenticate`.

Example

The following example adds a custom HTTP header named `SOAPAction` with a value of `Foo` to the my_lv object:

```
my_lv.addRequestHeader("SOAPAction", "'Foo'");
```

The following example creates an array named `headers` that contains two alternating HTTP headers and their associated values. The array is passed as an argument to `addRequestHeader()`.

```
var headers = ["Content-Type", "text/plain", "X-ClientAppVersion", "2.0"];
my_lv.addRequestHeader(headers);
```

The following example creates a new LoadVars object that adds a request header called `FLASH-UUID`. The header contains a variable that can be checked by the server.

```
var my_lv:LoadVars = new LoadVars();
my_lv.addRequestHeader("FLASH-UUID", "41472");
my_lv.name = "Mort";
my_lv.age = 26;
my_lv.send("http://flash-mx.com/mm/cgivars.cfm", "_blank", "POST");
```

See also

`XML.addRequestHeader()`

LoadVars.contentType

Availability

Flash Player 6.

Usage

my_lv.contentType:*String*

Description

Property; the MIME type that is sent to the server when you call `LoadVars.send()` or `LoadVars.sendAndLoad()`. The default is *application/x-www-form-urlencoded*.

Example

The following example creates a LoadVars object and displays the default content type of the data that is sent to the server.

```
var my_lv:LoadVars = new LoadVars();
trace(my_lv.contentType);  // output: application/x-www-form-urlencoded
```

See also

`LoadVars.send()`, `LoadVars.sendAndLoad()`

LoadVars.decode()

Availability

Flash Player 7.

Usage

my_lv.decode(*variables*:*String*) : *Void*

Parameters

variables A URL-encoded query string containing name/value pairs.

Returns

Nothing.

Description

Method; converts the variable string to properties of the specified LoadVars object.

This method is used internally by the `LoadVars.onData` event handler. Most users do not need to call this method directly. If you override the `LoadVars.onData` event handler, you can explicitly call `LoadVars.decode()` to parse a string of variables.

Example

The following example traces the three variables:

```
// Create a new LoadVars object
var my_lv:LoadVars = new LoadVars();
//Convert the variable string to properties
my_lv.decode("name=Mort&score=250000");
trace(my_lv.toString());
// Iterate over properties in my_lv
for (var prop in my_lv) {
  trace(prop+" -> "+my_lv[prop]);
}
```

See also

`LoadVars.onData`, `XML.parseXML()`

LoadVars.getBytesLoaded()

Availability

Flash Player 6.

Usage

my_lv.getBytesLoaded() *: Number*

Parameters

None.

Returns

An integer.

Description

Method; returns the number of bytes downloaded by `LoadVars.load()` or `LoadVars.sendAndLoad()`. This method returns `undefined` if no load operation is in progress or if a load operation has not yet begun.

Example

The following example uses a ProgressBar instance and a LoadVars object to download a text file. When you test the file, two things are displayed in the Output panel: whether the file loads successfully and how much data loads into the SWF file. You must replace the URL parameter of the `LoadVars.load()` command so that the parameter refers to a valid text file using HTTP. If you attempt to use this example to load a local file that resides on your hard disk, this example will not work properly because in test movie mode Flash Player loads local files in their entirety. To see this code work, add a ProgressBar instance called `loadvars_pb` to the Stage. Then add the following ActionScript to Frame 1 of the Timeline:

```
var loadvars_pb:mx.controls.ProgressBar;
var my_lv:LoadVars = new LoadVars();
loadvars_pb.mode = "manual";
this.createEmptyMovieClip("timer_mc", 999);
timer_mc.onEnterFrame = function() {
  var lvBytesLoaded:Number = my_lv.getBytesLoaded();
  var lvBytesTotal:Number = my_lv.getBytesTotal();
  if (lvBytesTotal != undefined) {
    trace("Loaded "+lvBytesLoaded+" of "+lvBytesTotal+" bytes.");
    loadvars_pb.setProgress(lvBytesLoaded, lvBytesTotal);
  }
};
my_lv.onLoad = function(success:Boolean) {
  loadvars_pb.setProgress(my_lv.getBytesLoaded(), my_lv.getBytesTotal());
  delete timer_mc.onEnterFrame;
  if (success) {
    trace("LoadVars loaded successfully.");
  } else {
    trace("An error occurred while loading variables.");
  }
};
my_lv.load("[place a valid URL pointing to a text file here]");
```

LoadVars.getBytesTotal()

Availability

Flash Player 6.

Usage

`my_lv.getBytesTotal() : Number`

Parameters

None.

Returns

An integer.

Description

Method; returns the total number of bytes downloaded by `LoadVars.load()` or `LoadVars.sendAndLoad()`. This method returns `undefined` if no load operation is in progress or if a load operation has not started. This method also returns `undefined` if the number of total bytes can't be determined (for example, if the download was initiated but the server did not transmit an HTTP content-length).

Example

The following example uses a ProgressBar instance and a LoadVars object to download a text file. When you test the file, two things are displayed in the Output panel: whether the file loads successfully and how much data loads into the SWF file. You must replace the URL parameter of the `LoadVars.load()` command so that the parameter refers to a valid text file using HTTP. If you attempt to use this example to load a local file that resides on your hard disk, this example will not work properly because in test movie mode Flash Player loads local files in their entirety. To see this code work, add a ProgressBar instance called `loadvars_pb` to the Stage. Then add the following ActionScript to Frame 1 of the Timeline:

```
var loadvars_pb:mx.controls.ProgressBar;
var my_lv:LoadVars = new LoadVars();
loadvars_pb.mode = "manual";
this.createEmptyMovieClip("timer_mc", 999);
timer_mc.onEnterFrame = function() {
  var lvBytesLoaded:Number = my_lv.getBytesLoaded();
  var lvBytesTotal:Number = my_lv.getBytesTotal();
  if (lvBytesTotal != undefined) {
    trace("Loaded "+lvBytesLoaded+" of "+lvBytesTotal+" bytes.");
    loadvars_pb.setProgress(lvBytesLoaded, lvBytesTotal);
  }
};
my_lv.onLoad = function(success:Boolean) {
  loadvars_pb.setProgress(my_lv.getBytesLoaded(), my_lv.getBytesTotal());
  delete timer_mc.onEnterFrame;
  if (success) {
    trace("LoadVars loaded successfully.");
  } else {
    trace("An error occurred while loading variables.");
    }
};
my_lv.load("[place a valid URL pointing to a text file here]");
```

LoadVars.load()

Availability

Flash Player 6; behavior changed in Flash Player 7.

Usage

`my_lv.load(url:String) : Boolean`

Parameters

`url` A string; the URL from which to download the variables. If the SWF file issuing this call is running in a web browser, `url` must be in the same domain as the SWF file; for details, see the Description section.

Returns

A Boolean value; `false` if no parameter is specified, `true` otherwise.

Description

Method; downloads variables from the specified URL, parses the variable data, and places the resulting variables into `my_lv`. Any properties in `my_lv` with the same names as downloaded variables are overwritten. Any properties in `my_lv` with different names than downloaded variables are not deleted. This is an asynchronous action.

The downloaded data must be in the MIME content type *application/x-www-form-urlencoded*. This is the same format used by loadVariables().

In SWF files running in a version of the player earlier than Flash Player 7, `url` must be in the same superdomain as the SWF file that is issuing this call. A superdomain is derived by removing the left-most component of a file's URL. For example, a SWF file at www.someDomain.com can load data from sources at store.someDomain.com because both files are in the same superdomain named someDomain.com.

In SWF files of any version running in Flash Player 7 or later, `url` must be in exactly the same domain (see "Flash Player security features" in *Using ActionScript in Flash*). For example, a SWF file at www.someDomain.com can load data only from sources that are also at www.someDomain.com. If you want to load data from a different domain, you can place a *cross-domain policy file* on the server hosting the SWF file. For more information, see "About allowing cross-domain data loading" in *Using ActionScript in Flash*.

Also, in files published for Flash Player 7, case-sensitivity (see "Case sensitivity" in *Using ActionScript in Flash*) is supported for external variables loaded with `LoadVars.load()`.

This method is similar to `XML.load()`.

Example

The following code defines an `onLoad` handler function that signals when data is returned to the Flash application from a server-side PHP script, and then loads the data in passvars.php.

```
var my_lv:LoadVars = new LoadVars();
my_lv.onLoad = function(success:Boolean) {
  if (success) {
    trace(this.toString());
  } else {
    trace("Error loading/parsing LoadVars.");
  }
};
my_lv.load("http://www.flash-mx.com/mm/params.txt");
```

For an in-depth example, see "Using the LoadVars class" in *Using ActionScript in Flash* and the Macromedia DevNet article "Macromedia Flash MX and PHP" at www.macromedia.com/devnet/mx/flash/articles/flashmx_php.html.

An example is also in the guestbook.fla file in the HelpExamples folder. The following list gives typical paths to this folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/
-

`LoadVars.loaded`

LoadVars.loaded

Availability

Flash Player 6.

Usage

my_lv.loaded:*Boolean*

Description

Property; a Boolean value that indicates whether a load or sendAndLoad operation has completed, undefined by default. When a `LoadVars.load()` or `LoadVars.sendAndLoad()` operation is started, the `loaded` property is set to `false`; when the operation completes, the `loaded` property is set to `true`. If the operation has not completed or has failed with an error, the `loaded` property remains set to `false`.

This property is similar to the `XML.loaded` property.

Example

The following example loads a text file and displays information in the Output panel when the operation completes.

```
var my_lv:LoadVars = new LoadVars();
my_lv.onLoad = function(success:Boolean) {
  trace("LoadVars loaded successfully: "+this.loaded);
};
my_lv.load("http://flash-mx.com/mm/params.txt");
```

LoadVars.onData

Availability

Flash Player 6.

Usage

```
my_lv.onData = function(src:String) {
  // your statements here
}
```

Parameters

src A string or `undefined`; the raw (unparsed) data from a `LoadVars.load()` or `LoadVars.sendAndLoad()` method call.

Returns

Nothing.

Description

Event handler; invoked when data has completely downloaded from the server or when an error occurs while data is downloading from a server. This handler is invoked before the data is parsed and can be used to call a custom parsing routine instead of the one built in to Flash Player. The value of the *src* parameter passed to the function assigned to `LoadVars.onData` can be either `undefined` or a string that contains the URL-encoded name-value pairs downloaded from the server. If the *src* parameter is `undefined`, an error occurred while downloading the data from the server.

The default implementation of `LoadVars.onData` invokes `LoadVars.onLoad`. You can override this default implementation by assigning a custom function to `LoadVars.onData`, but `LoadVars.onLoad` is not called unless you call it in your implementation of `LoadVars.onData`.

Example

The following example loads a text file and displays content in a TextArea instance called `content_ta` when the operation completes. If an error occurs, then information displays in the Output panel.

```
var my_lv:LoadVars = new LoadVars();
my_lv.onData = function(src:String) {
  if (src == undefined) {
    trace("Error loading content.");
    return;
  }
  content_ta.text = src;
};
my_lv.load("content.txt", my_lv, "GET");
```

LoadVars.onLoad

Availability

Flash Player 6.

Usage

```
my_lv.onLoad = function(success:Boolean) {
  // your statements here
}
```

Parameters

success A Boolean value that indicates whether the load operation ended in success (`true`) or failure (`false`).

Returns

A Boolean value.

Description

Event handler; invoked when a `LoadVars.load()` or `LoadVars.sendAndLoad()` operation has ended. If the operation was successful, *my_lv* is populated with variables downloaded by the operation, and these variables are available when this handler is invoked.

This handler is undefined by default.

This event handler is similar to `XML.onLoad`.

Example

For the following example, add a TextInput instance called `name_ti`, a TextArea instance called `result_ta`, and a Button instance called `submit_button` to the Stage. When the user clicks the Login button instance in the following example, two LoadVars objects are created: `send_lv` and `result_lv`. The `send_lv` object copies the name from the `name_ti` instance and sends the data to greeting.cfm. The result from this script loads into the `result_lv` object, and the server response displays in the TextArea instance (`result_ta`). Add the following ActionScript on Frame 1 of the Timeline:

```
var submitListener:Object = new Object();
submitListener.click = function(evt:Object) {
  var result_lv:LoadVars = new LoadVars();
  result_lv.onLoad = function(success:Boolean) {
    if (success) {
      result_ta.text = result_lv.welcomeMessage;
    } else {
      result_ta.text = "Error connecting to server.";
    }
  };
  var send_lv:LoadVars = new LoadVars();
  send_lv.name = name_ti.text;
  send_lv.sendAndLoad("http://www.flash-mx.com/mm/greeting.cfm", result_lv,
  "POST");
};
submit_button.addEventListener("click", submitListener);
```

To view a more robust example, see the login.fla file in the HelpExamples folder. Typical paths to the HelpExamples folder are:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/
-

`LoadVars.loaded`, `LoadVars.load()`, `LoadVars.sendAndLoad()`

LoadVars.send()

Availability

Flash Player 6.

Usage

`my_lv.send(url:String,target:String [, method:String]) : Boolean`

Parameters

`url` A string; the URL to which to upload variables.

`target` A string; the browser window or frame in which any response will appear. You can enter the name of a specific window or select from the following reserved target names:

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

`method` A string; the `GET` or `POST` method of the HTTP protocol.

Returns

A Boolean value; `false` if no parameters are specified, `true` otherwise.

Description

Method; sends the variables in the *`my_lv`* object to the specified URL. All enumerable variables in *`my_lv`* are concatenated into a string in the *`application/x-www-form-urlencoded`* format by default, and the string is posted to the URL using the HTTP `POST` method. This is the same format used by `loadVariables()`. The MIME content type sent in the HTTP request headers is the value of `my_lv.contentType` or the default *`application/x-www-form-urlencoded`*. The `POST` method is used unless `GET` is specified.

You must specify the *`target`* parameter to ensure that the script or application at the specified URL will be executed. If you omit the *`target`* parameter, the function will return `true`, but the script or application will not be executed.

The `send()` method is useful if you want the server response to:

- replace the SWF content (use `"_self"` as the *`target`* parameter);
- appear in a new window (use `"_blank"` as the *`target`* parameter);
- appear in the parent or top-level frame (use `"_parent"` or `"_top"` as the *`target`* parameter);
- appear in a named frame (use the frame's name as a string for the *`target`* parameter).

A successful `send()` method call will always open a new browser window or replace content in an existing window or frame. If you would rather send information to a server and continue playing your SWF file without opening a new window or replacing content in a window or frame, then you should use `LoadVars.sendAndLoad()`.

This method is similar to `XML.send()`.

Example

The following example copies two values from text fields and sends the data to a CFM script, which is used to handle the information. For example, the script might check if the user got a high score and then insert that data into a database table.

```
var my_lv:LoadVars = new LoadVars();
my_lv.playerName = playerName_txt.text;
my_lv.playerScore = playerScore_txt.text;
my_lv.send("setscore.cfm", "_blank", "POST");
```

See also

`LoadVars.sendAndLoad()`, `XML.send()`

LoadVars.sendAndLoad()

Availability

Flash Player 6; behavior changed in Flash Player 7.

Usage

`my_lv.sendAndLoad(url:String, targetObject[, method:String]) : Boolean`

Parameters

`url` A string; the URL to which to upload variables. If the SWF file issuing this call is running in a web browser, `url` must be in the same domain as the SWF file; for details, see "Description".

`targetObject` LoadVars; the LoadVars object that receives the downloaded variables.

`method` A string; the `GET` or `POST` method of the HTTP protocol.

Returns

A Boolean value.

Description

Method; posts variables in the `my_lv` object to the specified URL. The server response is downloaded, parsed as variable data, and the resulting variables are placed in the `targetObject` object.

Variables are posted in the same manner as `LoadVars.send()`. Variables are downloaded into `targetObject` in the same manner as `LoadVars.load()`.

In SWF files running in a version of the player earlier than Flash Player 7, `url` must be in the same superdomain as the SWF file that is issuing this call. A superdomain is derived by removing the left-most component of a file's URL. For example, a SWF file at www.someDomain.com can load data from sources at store.someDomain.com, because both files are in the same superdomain of someDomain.com.

In SWF files of any version running in Flash Player 7 or later, `url` must be in exactly the same domain (see "Flash Player security features" in *Using ActionScript in Flash*). For example, a SWF file at www.someDomain.com can load data only from sources that are also at www.someDomain.com. If you want to load data from a different domain, you can place a *cross-domain policy file* on the server hosting the SWF file. For more information, see "About allowing cross-domain data loading" in *Using ActionScript in Flash*.

This method is similar to `XML.sendAndLoad()`.

Example

For the following example, add a TextInput instance called `name_ti`, a TextArea instance called `result_ta`, and a Button instance called `submit_button` to the Stage. When the user clicks the Login button instance in the following example, two LoadVars objects are created: `send_lv` and `result_lv`. The `send_lv` object copies the name from the `name_ti` instance and sends the data to greeting.cfm. The result from this script loads into the `result_lv` object, and the server response displays in the TextArea instance (`result_ta`). Add the following ActionScript to Frame 1 of the Timeline:

```
var submitListener:Object = new Object();
submitListener.click = function(evt:Object) {
  var result_lv:LoadVars = new LoadVars();
  result_lv.onLoad = function(success:Boolean) {
    if (success) {
      result_ta.text = result_lv.welcomeMessage;
    } else {
      result_ta.text = "Error connecting to server.";
    }
  };
  var send_lv:LoadVars = new LoadVars();
  send_lv.name = name_ti.text;
  send_lv.sendAndLoad("http://www.flash-mx.com/mm/greeting.cfm", result_lv,
  "POST");
};
submit_button.addEventListener("click", submitListener);
```

To view a more robust example, see the login.fla file in the HelpExamples folder. Typical paths to the HelpExamples folder are:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/
-

`LoadVars.send()`, `XML.sendAndLoad()`

LoadVars.toString()

Availability

Flash Player 6.

Usage

`my_lv``.toString()` *`: String`*

Parameters

None.

Returns

A string.

Description

Method; returns a string containing all enumerable variables in *`my_lv`*, in the MIME content encoding *application/x-www-form-urlencoded*.

Example

The following example instantiates a new `LoadVars()` object, creates two properties, and uses `toString()` to return a string containing both properties in URL encoded format:

```
var my_lv:LoadVars = new LoadVars();
my_lv.name = "Gary";
my_lv.age = 26;
trace (my_lv.toString()); //output: age=26&name=Gary
```

LocalConnection class

Availability

Flash Player 6.

Description

The LocalConnection class lets you develop SWF files that can send instructions to each other without the use of fscommand() or JavaScript. LocalConnection objects can communicate only among SWF files that are running on the same client computer, but they can be running in different applications—for example, a SWF file running in a browser and a SWF file running in a projector. You can use LocalConnection objects to send and receive data within a single SWF file, but this is not a standard implementation; all the examples in this section illustrate communication between different SWF files.

The primary methods used to send and receive data are `LocalConnection.send()` and `LocalConnection.connect()`. At its most basic, your code will implement the following commands; notice that both the `LocalConnection.send()` and `LocalConnection.connect()` commands specify the same connection name, `lc_name`:

```
// Code in the receiving SWF file
this.createTextField("result_txt", 1, 10, 10, 100, 22);
result_txt.border = true;
var receiving_lc:LocalConnection = new LocalConnection();
receiving_lc.methodToExecute = function(param1:Number, param2:Number) {
  result_txt.text = param1+param2;
};
receiving_lc.connect("lc_name");

// Code in the sending SWF file
var sending_lc:LocalConnection = new LocalConnection();
sending_lc.send("lc_name", "methodToExecute", 5, 7);
```

The simplest way to use a LocalConnection object is to allow communication only between LocalConnection objects located in the same domain because you won't have security issues. However, if you need to allow communication between domains, you have several ways to implement security measures. For more information, see the discussion of the *`connectionName`* parameter in `LocalConnection.send()` and the `LocalConnection.allowDomain` and `LocalConnection.domain()` entries.

Method summary for the LocalConnection class

Method	Description
`LocalConnection.close()`	Closes (disconnects) the LocalConnection object.
`LocalConnection.connect()`	Prepares the LocalConnection object to receive commands from a `LocalConnection.send()` command.
`LocalConnection.domain()`	Returns a string representing the superdomain of the location of the current SWF file.
`LocalConnection.send()`	Invokes a method on a specified LocalConnection object.

Event handler summary for the LocalConnection class

Event handler	Description
`LocalConnection.allowDomain`	Invoked whenever the current (receiving) LocalConnection object receives a request to invoke a method from a sending LocalConnection object.
`LocalConnection.allowInsecureDomain`	Invoked whenever the current (receiving) LocalConnection object, which is in a SWF file hosted at a domain using a secure protocol (HTTPS), receives a request to invoke a method from a sending LocalConnection object that is in a SWF file hosted at a non-secure protocol.
`LocalConnection.onStatus`	Invoked after a sending LocalConnection object tries to send a command to a receiving LocalConnection object.

Constructor for the LocalConnection class

Availability

Flash Player 6.

Usage

```
new LocalConnection() : LocalConnection
```

Parameters

None.

Returns

A reference to a LocalConnection object.

Description

Constructor; creates a LocalConnection object.

Example

The following example shows how receiving and sending SWF files create LocalConnnection objects. The two SWF files can use the same name or different names for their respective LocalConnection objects. In this example they use different names.

```
// Code in the receiving SWF file
this.createTextField("result_txt", 1, 10, 10, 100, 22);
result_txt.border = true;
var receiving_lc:LocalConnection = new LocalConnection();
receiving_lc.methodToExecute = function(param1:Number, param2:Number) {
  result_txt.text = param1+param2;
};
receiving_lc.connect("lc_name");
```

The following SWF file sends the request to the first SWF file.

```
// Code in the sending SWF file
var sending_lc:LocalConnection = new LocalConnection();
sending_lc.send("lc_name", "methodToExecute", 5, 7);
```

See also

`LocalConnection.connect()`, `LocalConnection.send()`

LocalConnection.allowDomain

Availability

Flash Player 6; behavior changed in Flash Player 7.

Usage

```
receiving_lc.allowDomain = function([sendingDomain:String]) : Boolean {
  // Your statements here return true or false
}
```

Parameters

sendingDomain A string that represents an optional parameter specifying the domain of the SWF file that contains the sending LocalConnection object.

Returns

Nothing.

Description

Event handler; invoked whenever *receiving_lc* receives a request to invoke a method from a sending LocalConnection object. Flash expects the code you implement in this handler to return a Boolean value of `true` or `false`. If the handler doesn't return `true`, the request from the sending object is ignored, and the method is not invoked.

When this event handler is absent, Flash Player applies a default security policy, which is equivalent to the following code:

```
my_lc.allowDomain = function (sendingDomain)
{
  return (sendingDomain == this.domain());
}
```

Use `LocalConnection.allowDomain` to explicitly permit LocalConnection objects from specified domains, or from any domain, to execute methods of the receiving LocalConnection object. If you don't declare the *sendingDomain* parameter, you probably want to accept commands from any domain, and the code in your handler would be simply `return true`. If you do declare *sendingDomain*, you probably want to compare the value of *sendingDomain* with domains from which you want to accept commands. The following examples show both implementations.

In files authored for Flash Player 6, the *sendingDomain* parameter contains the superdomain of the caller. In files authored for Flash Player 7 or later, the *sendingDomain* parameter contains the exact domain of the caller. In the latter case, to allow access by SWF files hosted at either www.domain.com or store.domain.com, you must explicitly allow access from both domains.

```
// For Flash Player 6
receiving_lc.allowDomain = function(sendingDomain) {
  return(sendingDomain=="domain.com");
}
// For Flash Player 7 or later
receiving_lc.allowDomain = function(sendingDomain) {
  return(sendingDomain=="www.domain.com" ||
    sendingDomain=="store.domain.com");
}
```

Also, for files authored for Flash Player 7 or later, you can't use this method to let SWF files hosted using a secure protocol (HTTPS) allow access from SWF files hosted in nonsecure protocols; you must use the LocalConnection.allowInsecureDomain event handler instead.

Occasionally, you might encounter the following situation. Suppose you load a child SWF file from a different domain. You want to implement this method so that the child SWF file can make LocalConnection calls to the parent SWF file, but you don't know the final domain from which the child SWF file will come. This can happen, for example, when you use load-balancing redirects or third-party servers.

In this situation, you can use the MovieClip._url property in your implementation of this method. For example, if you load a SWF file into `my_mc`, you can then implement this method by checking whether the domain argument matches the domain of `my_mc._url`. (You must parse the domain out of the full URL contained in `my_mc._url`.)

If you do this, make sure that you wait until the SWF file in `my_mc` is loaded, because the `_url` property will not have its final, correct value until the file is completely loaded. The best way to determine when a child SWF file finishes loading is to use MovieClipLoader.onLoadComplete.

The opposite situation can also occur: You might create a child SWF file that wants to accept LocalConnection calls from its parent but doesn't know the domain of its parent. In this situation, implement this method by checking whether the domain argument matches the domain of `_parent._url`. Again, you must parse the domain out of the full URL from `_parent._url`. In this situation, you don't have to wait for the parent SWF file to load; the parent will already be loaded by the time the child loads.

Example

The following example shows how a LocalConnection object in a receiving SWF file can permit SWF files from any domain to invoke its methods. Compare this to the example in `LocalConnection.connect()`, in which only SWF files from the same domain can invoke the `trace()` method in the receiving SWF file. For a discussion of the use of the underscore (_) in the connection name, see `LocalConnection.send()`.

```
this.createTextField("welcome_txt", this.getNextHighestDepth(), 10, 10, 100,
  20);
var my_lc:LocalConnection = new LocalConnection();
my_lc.allowDomain = function(sendingDomain:String) {
  domain_txt.text = sendingDomain;
  return true;
};
my_lc.allowInsecureDomain = function(sendingDomain:String) {
  return (sendingDomain == this.domain());
}
my_lc.sayHello = function(name:String) {
  welcome_txt.text = "Hello, "+name;
};
my_lc.connect("_mylc");
```

The following example sends a string to the previous SWF file and displays a status message about whether the local connection was able to connect to the file. A TextInput component called `name_ti`, a TextArea instance called `status_ta` and a Button instance called `send_button` are used to display content.

```
var sending_lc:LocalConnection;
var sendListener:Object = new Object();
sendListener.click = function(evt:Object) {
  sending_lc = new LocalConnection();
  sending_lc.onStatus = function(infoObject:Object) {
    switch (infoObject.level) {
    case 'status' :
      status_ta.text = "LocalConnection connected successfully.";
      break;
    case 'error' :
      status_ta.text = "LocalConnection encountered an error.";
      break;
    }
  };
  sending_lc.send("_mylc", "sayHello", name_ti.text);
};
send_button.addEventListener("click", sendListener);
```

In the following example, the receiving SWF file, which resides in thisDomain.com, accepts commands only from SWF files located in thisDomain.com or thatDomain.com:

```
var aLocalConn:LocalConnection = new LocalConnection();
aLocalConn.Trace = function(aString) {
  aTextField += aString+newline;
};
aLocalConn.allowDomain = function(sendingDomain) {
  return (sendingDomain == this.domain() || sendingDomain ==
  "www.macromedia.com");
};
aLocalConn.connect("_mylc");
```

When published for Flash Player 7 or later, exact domain matching is used. This means that the example will fail if the SWF files are located at www.thatDomain.com but will work if the files are located at thatDomain.com.

See also

`LocalConnection.connect()`, `LocalConnection.domain()`, `LocalConnection.send()`, `MovieClip._url`, `MovieClipLoader.onLoadComplete`, `_parent`

LocalConnection.allowInsecureDomain

Availability

Flash Player 7.

Usage

```
receiving_lc.allowInsecureDomain = function([sendingDomain:String]) : Boolean
  {
  // Your statements here return true or false
}
```

Parameters

`sendingDomain` A string that represents an optional parameter specifying the domain of the SWF file that contains the sending LocalConnection object.

Returns

A Boolean value.

Description

Event handler; invoked whenever *`receiving_lc`*, which is in a SWF file hosted at a domain using a secure protocol (HTTPS), receives a request to invoke a method from a sending LocalConnection object that is in a SWF file hosted at a nonsecure protocol. Flash expects the code you implement in this handler to return a Boolean value of `true` or `false`. If the handler doesn't return `true`, the request from the sending object is ignored, and the method is not invoked.

By default, SWF files hosted using the HTTPS protocol can be accessed only by other SWF files hosted using the HTTPS protocol. This implementation maintains the integrity provided by the HTTPS protocol.

Using this method to override the default behavior is not recommended, as it compromises HTTPS security. However, you might need to do so, for example, if you need to permit access to HTTPS files published for Flash Player 7 or later from HTTP files published for Flash Player 6.

A SWF file published for Flash Player 6 can use the LocalConnection.allowDomain event handler to permit HTTP to HTTPS access. However, because security is implemented differently in Flash Player 7, you must use the `LocalConnection.allowInsecureDomain()` method to permit such access in SWF files published for Flash Player 7 or later.

Example

The following example allows connections from the current domain or from www.macromedia.com, or allows insecure connections only from the current domain.

```
this.createTextField("welcome_txt", this.getNextHighestDepth(), 10, 10, 100,
  20);
var my_lc:LocalConnection = new LocalConnection();
my_lc.allowDomain = function(sendingDomain:String) {
  domain_txt.text = sendingDomain;
  return (sendingDomain == this.domain() || sendingDomain ==
  "www.macromedia.com");
};
my_lc.allowInsecureDomain = function(sendingDomain:String) {
  return (sendingDomain == this.domain());
}
my_lc.sayHello = function(name:String) {
  welcome_txt.text = "Hello, "+name;
};
my_lc.connect("lc_name");
```

See also

`LocalConnection.allowDomain`, `LocalConnection.connect()`

LocalConnection.close()

Availability

Flash Player 6.

Usage

```
receiving_lc.close() : Void
```

Parameters

None.

Returns

Nothing.

Description

Method; closes (disconnects) a LocalConnection object. Issue this command when you no longer want the object to accept commands—for example, when you want to issue a `LocalConnection.connect()` command using the same *`connectionName`* parameter in another SWF file.

Example

The following example closes a connection called `receiving_lc` when you click a Button component instance called `close_button`:

```
this.createTextField("welcome_txt", this.getNextHighestDepth(), 10, 10, 100,
  22);
this.createTextField("status_txt", this.getNextHighestDepth(), 10, 42,
  100,44);

var receiving_lc:LocalConnection = new LocalConnection();
receiving_lc.sayHello = function(name:String) {
  welcome_txt.text = "Hello, "+name;
};
receiving_lc.connect("lc_name");
var closeListener:Object = new Object();
closeListener.click = function(evt:Object) {
  receiving_lc.close();
  status_txt.text = "connection closed";
};
close_button.addEventListener("click", closeListener);
```

See also

`LocalConnection.connect()`

LocalConnection.connect()

Availability

Flash Player 6.

Usage

`receiving_lc.connect(connectionName:String) : Boolean`

Parameters

`connectionName` A string that corresponds to the connection name specified in the `LocalConnection.send()` command that wants to communicate with `receiving_lc`.

Returns

A Boolean value: `true` if no other process running on the same client computer has already issued this command using the same value for the `connectionName` parameter; `false` otherwise.

Description

Method; prepares a LocalConnection object to receive commands from a `LocalConnection.send()` command (called the *sending LocalConnection object*). The object used with this command is called the *receiving LocalConnection object*. The receiving and sending objects must be running on the same client computer.

Make sure you define the methods attached to `receiving_lc` before calling this method, as shown in all the examples in this section.

By default, Flash Player resolves `connectionName` into a value of `"superdomain:connectionName"`, where `superdomain` is the superdomain of the SWF file containing the `LocalConnection.connect()` command. For example, if the SWF file containing the receiving LocalConnection object is located at www.someDomain.com, `connectionName` resolves to `"someDomain.com:connectionName"`. (If a SWF file is located on the client computer, the value assigned to `superdomain` is `"localhost"`.)

Also by default, Flash Player lets the receiving LocalConnection object accept commands only from sending LocalConnection objects whose connection name also resolves into a value of `"superdomain:connectionName"`. In this way, Flash makes it simple for SWF files located in the same domain to communicate with each other.

If you are implementing communication only between SWF files in the same domain, specify a string for `connectionName` that does not begin with an underscore (_) and that does not specify a domain name (for example, `"myDomain:connectionName"`). Use the same string in the `LocalConnection.connect(connectionName)` command.

If you are implementing communication between SWF files in different domains, specifying a string for `connectionName` that begins with an underscore (_) will make the SWF with the receiving LocalConnection object more portable between domains. Here are the two possible cases:

- If the string for `connectionName` does not begin with an underscore (_), Flash Player adds a prefix with the superdomain and a colon (for example, `"myDomain:connectionName"`). Although this ensures that your connection does not conflict with connections of the same name from other domains, any sending LocalConnection objects must specify this superdomain (for example, `"myDomain:connectionName"`). If the SWF with the receiving LocalConnection object is moved to another domain, the player changes the prefix to reflect the new superdomain (for example, `"anotherDomain:connectionName"`). All sending LocalConnection objects would have to be manually edited to point to the new superdomain.
- If the string for `connectionName` begins with an underscore (for example, `"_connectionName"`), Flash Player does not add a prefix to the string. This means that the receiving and sending LocalConnection objects will use identical strings for `connectionName`. If the receiving object uses LocalConnection.allowDomain to specify that connections from any domain will be accepted, the SWF with the receiving LocalConnection object can be moved to another domain without altering any sending LocalConnection objects.

For more information, see the discussion of `connectionName` in `LocalConnection.send()` and also the `LocalConnection.allowDomain` and `LocalConnection.domain()` entries.

Note: Colons are used as special characters to separate the superdomain from the `connectionName` string. A string for `connectionName` that contains a colon is not valid.

Example

The following example shows how a SWF file in a particular domain can invoke a method named `Trace` in a receiving SWF file in the same domain. The receiving SWF file functions as a trace window for the sending SWF file; it contains two methods that other SWF files can call—`Trace` and `Clear`. Buttons pressed in the sending SWF files call these methods with specified parameters.

```
// Receiving SWF
var aLocalConnection:LocalConnection = new LocalConnection();
aLocalConnection.Trace = function(aString):String{
  aTextField += aString + newline;
}
aLocalConnection.Clear = function(){
  aTextField = "";
}
aLocalConnection.connect("trace");
stop();
```

SWF 1 contains the following code, which creates a new Sound object that plays back an MP3 file at runtime. A ProgressBar called `playback_pb` displays the playback progress of the MP3 file. A Label component instance called `song_lbl` displays the name of the MP3 file. Buttons in different SWF files will be used to control the playback using a LocalConnection object.

```
var playback_pb:mx.controls.ProgressBar;
var my_sound:Sound;
```

```
playback_pb.setStyle("themeColor", "haloBlue");
this.createEmptyMovieClip("timer_mc", this.getNextHighestDepth());
var receiving_lc:LocalConnection = new LocalConnection();
receiving_lc.playMP3 = function(mp3Path:String, mp3Name:String) {
  song_lbl.text = mp3Name;
  playback_pb.indeterminate = true;
  my_sound = new Sound();
  my_sound.onLoad = function(success:Boolean) {
    playback_pb.indeterminate = false;
  };
  my_sound.onSoundComplete = function() {
    delete timer_mc.onEnterFrame;
  };
  timer_mc.onEnterFrame = function() {
    playback_pb.setProgress(my_sound.position, my_sound.duration);
  };
  my_sound.loadSound(mp3Path, true);
};
receiving_lc.connect("lc_name");
```

SWF 2 contains a button called `play_btn`. When you click the button, it connects to SWF 1 and passes two variables. The first variable contains the MP3 file to stream, and the second variable is the filename that you display in the Label component instance in SWF 1.

```
play_btn.onRelease = function() {
  var sending_lc:LocalConnection = new LocalConnection();
  sending_lc.send("lc_name", "playMP3", "song1.mp3", "Album - 01 - Song");
};
```

SWF 3 contains a button called `play_btn`. When you click the button, it connects to SWF 1 and passes two variables. The first variable contains the MP3 file to stream, and the second variable is the filename that you display in the Label component instance in SWF 1.

```
play_btn.onRelease = function() {
  var sending_lc:LocalConnection = new LocalConnection();
  sending_lc.send("lc_name", "playMP3", "song2.mp3", "Album - 02 - Another
  Song");
};
```

See also

`LocalConnection.send()`

LocalConnection.domain()

Availability

Flash Player 6; behavior changed in Flash Player 7.

Usage

my_lc.domain() : *String*

Parameters

None.

Returns

A string representing the domain of the location of the current SWF file; for more information, see the Description section.

Description

Method; returns a string representing the domain of the location of the current SWF file.

In SWF files published for Flash Player 6, the returned string is the superdomain of the current SWF file. For example, if the SWF file is located at www.macromedia.com, this command returns `"macromedia.com"`.

In SWF files published for Flash Player 7 or later, the returned string is the exact domain of the current SWF file. For example, if the SWF file is located at www.macromedia.com, this command returns `"www.macromedia.com"`.

If the current SWF file is a local file residing on the client computer, this command returns `"localhost"`.

The most common way to use this command is to include the domain name of the sending LocalConnection object as a parameter to the method you plan to invoke in the receiving LocalConnection object or with LocalConnection.allowDomain to accept commands from a specified domain. If you are enabling communication only between LocalConnection objects that are located in the same domain, you probably don't need to use this command.

Example

In the following example, a receiving SWF file accepts commands only from SWF files located in the same domain or at macromedia.com:

```
// If both the sending and receiving SWF files are Flash Player 6,
// then use the superdomain
var my_lc:LocalConnection = new LocalConnection();
my_lc.allowDomain = function(sendingDomain):String{
  return (sendingDomain==this.domain() || sendingDomain=="macromedia.com");
}

// If either the sending or receiving SWF file is Flash Player 7 or later,
// then use the exact domain. In this case, commands from a SWF file posted
// at www.macromedia.com will be accepted, but those from one posted at
// a different subdomain, e.g. livedocs.macromedia.com, will not.
var my_lc:LocalConnection = new LocalConnection();
```

```
my_lc.allowDomain = function(sendingDomain):String{
  return (sendingDomain==this.domain() ||
  sendingDomain=="www.macromedia.com");
}
```

In the following example, a sending SWF file located at www.yourdomain.com invokes a method in a receiving SWF file located at www.mydomain.com. The sending SWF file includes its domain name as a parameter to the method it invokes, so the receiving SWF file can return a reply value to a LocalConnection object in the correct domain. The sending SWF file also specifies that it will accept commands only from SWF files at mydomain.com.

Line numbers are included for reference purposes. The sequence of events is described in the following list:

- The receiving SWF file prepares to receive commands on a connection named `"sum"` (line 11). The Flash Player resolves the name of this connection to `"mydomain.com:sum"` (see LocalConnection.connect()).
- The sending SWF file prepares to receive a reply on the LocalConnection object named `"result"` (line 67). It also specifies that it will accept commands only from SWF files at mydomain.com (lines 51 to 53).
- The sending SWF file invokes the `aSum` method of a connection named `"mydomain.com:sum"` (line 68) and passes the following parameters: its superdomain, the name of the connection to receive the reply (`"result"`), and the values to be used by `aSum` (123 and 456).
- The `aSum` method (line 6) is invoked with the following values:
  sender = `"mydomain.com:result"`, `replyMethod` = `"aResult"`, `n1` = 123, and `n2` = 456. It then executes the following line of code:

  ```
  this.send("mydomain.com:result", "aResult", (123 + 456));
  ```
- The `aResult` method (line 54) shows the value returned by `aSum` (579).

  ```
  // The receiving SWF at http://www.mydomain.com/folder/movie.swf
  // contains the following code

  1    var aLocalConnection:LocalConnection = new LocalConnection();
  2    aLocalConnection.allowDomain = function()
  3    {
       // Allow connections from any domain
  4      return true;
  5    }
  6    aLocalConnection.aSum = function(sender, replyMethod, n1, n2)
  7    {
  8      this.send(sender, replyMethod, (n1 + n2));
  9    }
  10
  11   aLocalConnection.connect("sum");

  // The sending SWF at http://www.yourdomain.com/folder/movie.swf
  // contains the following code

  50   var lc:LocalConnection = new LocalConnection();
  51   lc.allowDomain = function(aDomain) {
         // Allow connections only from mydomain.com
  ```

```
  return (aDomain == "mydomain.com");
}
lc.aResult = function(aParam) {
  trace("The sum is " + aParam);
}

// determine our domain and see if we need to truncate it
var channelDomain:String = lc.domain();
if (getVersion() >= 7 && this.getSWFVersion() >= 7)
{
  // split domain name into elements
  var domainArray:Array = channelDomain.split(".");

  // if more than two elements are found,
  // chop off first element to create superdomain
  if (domainArray.length > 2)
  {
    domainArray.shift();
    channelDomain = domainArray.join(".");
  }
}

lc.connect("result");
lc.send("mydomain.com:sum", "aSum", channelDomain + ':' + "result",
  "aResult", 123, 456);
```

See also

`LocalConnection.allowDomain`

LocalConnection.onStatus

Availability

Flash Player 6.

Usage

```
sending_lc.onStatus = function(infoObject:Object) : Void{
  // your statements here
}
```

Parameters

infoObject A parameter defined according to the status message. For details about this parameter, see the Description section.

Returns

Nothing.

Description

Event handler; invoked after a sending LocalConnection object tries to send a command to a receiving LocalConnection object. If you want to respond to this event handler, you must create a function to process the information object sent by the LocalConnection object.

If the information object returned by this event handler contains a `level` value of `status`, Flash successfully sent the command to a receiving LocalConnection object. This does not mean that Flash successfully invoked the specified method of the receiving LocalConnection object; it means only that Flash could send the command. For example, the method is not invoked if the receiving LocalConnection object doesn't allow connections from the sending domain or if the method does not exist. The only way to know for sure if the method was invoked is to have the receiving object send a reply to the sending object.

If the information object returned by this event handler contains a `level` value of `error`, Flash cannot send the command to a receiving LocalConnection object, most likely because there is no receiving LocalConnection object connected whose name corresponds to the name specified in the *`sending_lc`*`.send()` command that invoked this handler.

In addition to this `onStatus` handler, Flash also provides a "super" function called System.onStatus. If `onStatus` is invoked for a particular object and there is no function assigned to respond to it, Flash processes a function assigned to `System.onStatus` if it exists.

In most cases, you implement this handler only to respond to error conditions, as shown in the following example.

Example

The following example displays a status message about whether the SWF file connects to another local connection object called `lc_name`. A TextInput component called `name_ti`, a TextArea instance called `status_ta` and a Button instance called `send_button` are used to display content.

```
var sending_lc:LocalConnection;
var sendListener:Object = new Object();
sendListener.click = function(evt:Object) {
  sending_lc = new LocalConnection();
  sending_lc.onStatus = function(infoObject:Object) {
    switch (infoObject.level) {
    case 'status' :
      status_ta.text = "LocalConnection connected successfully.";
      break;
    case 'error' :
      status_ta.text = "LocalConnection encountered an error.";
      break;
    }
  };
  sending_lc.send("lc_name", "sayHello", name_ti.text);
};
send_button.addEventListener("click", sendListener);
```

See also

`LocalConnection.send()`, `System.onStatus`

LocalConnection.send()

Availability

Flash Player 6.

Usage

```
sending_lc.send (connectionName:String, method:String [, p1,...,pN]) : Boolean
```

Parameters

`connectionName` A string that corresponds to the connection name specified in the `LocalConnection.connect()` command that wants to communicate with `sending_lc`.

`method` A string specifying the name of the method to be invoked in the receiving LocalConnection object. The following method names cause the command to fail: `send`, `connect`, `close`, `domain`, `onStatus`, and `allowDomain`.

`p1,...pN` Optional parameters to be passed to the specified method.

Returns

A Boolean value: `true` if Flash can carry out the request; `false` otherwise.

Note: A return value of `true` does not necessarily mean that Flash successfully connected to a receiving LocalConnection object; it means only that the command is syntactically correct. To determine whether the connection succeeded, see `LocalConnection.onStatus`.

Description

Method; invokes the method named `method` on a connection opened with the `LocalConnection.connect(connectionName)` command (the receiving LocalConnection object). The object used with this command is called the sending LocalConnection object. The SWF files that contain the sending and receiving objects must be running on the same client computer.

There is a limit to the amount of data you can pass as parameters to this command. If the command returns `false` but your syntax is correct, try dividing the LocalConnection.send() requests into multiple commands.

As discussed in the entry LocalConnection.connect(), Flash adds the current superdomain to `connectionName` by default. If you are implementing communication between different domains, you need to define `connectionName` in both the sending and receiving LocalConnection objects in such a way that Flash does not add the current superdomain to `connectionName`. You can do this in one of the following two ways:

- Use an underscore (_) at the beginning of `connectionName` in both the sending and receiving LocalConnection objects. In the SWF file containing the receiving object, use LocalConnection.allowDomain to specify that connections from any domain will be accepted. This implementation lets you store your sending and receiving SWF files in any domain.

- Include the superdomain in *`connectionName`* in the sending LocalConnection object—for example, `myDomain.com:myConnectionName`. In the receiving object, use LocalConnection.allowDomain to specify that connections from the specified superdomain will be accepted (in this case, myDomain.com) or that connections from any domain will be accepted.

Note: You cannot specify a superdomain in *`connectionName`* in the receiving LocalConnection object–you can only do this in the sending LocalConnection object.

Example

For an example of communicating between LocalConnection objects located in the same domain, see LocalConnection.connect(). For an example of communicating between LocalConnection objects located in any domain, see LocalConnection.allowDomain. For an example of communicating between LocalConnection objects located in specified domains, see LocalConnection.allowDomain and LocalConnection.domain().

See also

`LocalConnection.allowDomain`, `LocalConnection.connect()`, `LocalConnection.domain()`, `LocalConnection.onStatus`

Math class

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Description

The Math class is a top-level class whose methods and properties you can use without using a constructor.

Use the methods and properties of this class to access and manipulate mathematical constants and functions. All the properties and methods of the Math class are static and must be called using the syntax `Math.method(parameter)` or `Math.constant`. In ActionScript, constants are defined with the maximum precision of double-precision IEEE-754 floating-point numbers.

Several Math class methods use the measure of an angle in radians as a parameter.You can use the following equation to calculate radian values before calling the method and then provide the calculated value as the parameter, or you can provide the entire right side of the equation (with the angle's measure in degrees in place of `degrees`) as the radian parameter.

To calculate a radian value, use the following formula:

```
radians = degrees * Math.PI/180
```

The following is an example of passing the equation as a parameter to calculate the sine of a 45° angle:

`Math.sin(45 * Math.PI/180)` is the same as `Math.sin(.7854)`

Method summary for the Math class

Method	Description
`Math.abs()`	Computes an absolute value.
`Math.acos()`	Computes an arc cosine.
`Math.asin()`	Computes an arc sine.
`Math.atan()`	Computes an arc tangent.
`Math.atan2()`	Computes an angle from the *x*-axis to the point.
`Math.ceil()`	Rounds a number up to the nearest integer.
`Math.cos()`	Computes a cosine.
`Math.exp()`	Computes an exponential value.
`Math.floor()`	Rounds a number down to the nearest integer.
`Math.log()`	Computes a natural logarithm.
`Math.max()`	Returns the larger of the two integers.
`Math.min()`	Returns the smaller of the two integers.

Method	Description
`Math.pow()`	Computes *x* raised to the power of the *y*.
`Math.random()`	Returns a pseudo-random number between 0.0 and 1.0.
`Math.round()`	Rounds to the nearest integer.
`Math.sin()`	Computes a sine.
`Math.sqrt()`	Computes a square root.
`Math.tan()`	Computes a tangent.

Property summary for the Math class

All the following properties for the Math class are constants:

Property	Description
`Math.E`	Euler's constant and the base of natural logarithms (approximately 2.718).
`Math.LN2`	The natural logarithm of 2 (approximately 0.693).
`Math.LOG2E`	The base 2 logarithm of *e* (approximately 1.442).
`Math.LN2`	The natural logarithm of 10 (approximately 2.302).
`Math.LOG10E`	The base 10 logarithm of *e* (approximately 0.434).
`Math.PI`	The ratio of the circumference of a circle to its diameter (approximately 3.14159).
`Math.SQRT1_2`	The reciprocal of the square root of 1/2 (approximately 0.707).
`Math.SQRT2`	The square root of 2 (approximately 1.414).

Math.abs()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

```
Math.abs(x:Number) : Number
```

Parameters

x A number.

Returns

A number.

Description

Method; computes and returns an absolute value for the number specified by the parameter *x*.

Example

The following example shows how `Math.abs()` returns the absolute value of a number and does not affect the value of the *x* parameter (called `num` in this example):

```
var num:Number = -12;
var numAbsolute:Number = Math.abs(num);
trace(num); // output: -12
trace(numAbsolute); // output: 12
```

Math.acos()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

`Math.acos(x:Number) : Number`

Parameters

x	A number from -1.0 to 1.0.

Returns

A number; the arc cosine of the parameter *x*.

Description

Method; computes and returns the arc cosine of the number specified in the parameter *x*, in radians.

Example

The following example displays the arc cosine for several values.

```
trace(Math.acos(-1)); // output: 3.14159265358979
trace(Math.acos(0));  // output: 1.5707963267949
trace(Math.acos(1));  // output: 0
```

See also

`Math.asin()`, `Math.atan()`, `Math.atan2()`, `Math.cos()`, `Math.sin()`, `Math.tan()`

Math.asin()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

`Math.asin(`*`x:Number`*`) : `*`Number`*`;`

Parameters

`x` A number from -1.0 to 1.0.

Returns

A number between negative pi divided by 2 and positive pi divided by 2.

Description

Method; computes and returns the arc sine for the number specified in the parameter *`x`*, in radians.

Example

The following example displays the arc sine for several values.

```
trace(Math.asin(-1)); // output: -1.5707963267949
trace(Math.asin(0));  // output: 0
trace(Math.asin(1));  // output: 1.5707963267949
```

See also

`Math.acos()`, `Math.atan()`, `Math.atan2()`, `Math.cos()`, `Math.sin()`, `Math.tan()`

Math.atan()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

`Math.atan(tangent:Number) : Number`

Parameters

tangent A number that represents the tangent of an angle.

Returns

A number between negative pi divided by 2 and positive pi divided by 2.

Description

Method; computes and returns the value, in radians, of the angle whose tangent is specified in the parameter *tangent*. The return value is between negative pi divided by 2 and positive pi divided by 2.

Example

The following example displays the angle value for several tangents.

```
trace(Math.atan(-1)); // output: -0.785398163397448
trace(Math.atan(0));  // output: 0
trace(Math.atan(1));  // output: 0.785398163397448
```

See also

`Math.acos()`, `Math.asin()`, `Math.atan2()`, `Math.cos()`, `Math.sin()`, `Math.tan()`

Math.atan2()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

```
Math.atan2(y:Number, x:Number) : Number
```

Parameters

y	A number specifying the *y* coordinate of the point.
x	A number specifying the *x* coordinate of the point.

Returns

A number.

Description

Method; computes and returns the angle of the point *y*/*x* in radians, when measured counterclockwise from a circle's *x* axis (where 0,0 represents the center of the circle). The return value is between positive pi and negative pi.

Example

The following example displays the following radians from the specified coordinates 10, 0.

```
trace(Math.atan2(10, 0)); // output: 1.5707963267949
```

See also

`Math.acos()`, `Math.asin()`, `Math.atan()`, `Math.cos()`, `Math.sin()`, `Math.tan()`

Math.ceil()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

```
Math.ceil(x:Number) : Number
```

Parameters

x	A number or expression.

Returns

An integer that is both closest to, and greater than or equal to, parameter *x*.

Description

Method; returns the ceiling of the specified number or expression. The ceiling of a number is the closest integer that is greater than or equal to the number.

Example

The following code returns a value of 13:

```
Math.ceil(12.5);
```

See also

`Math.floor()`, `Math.round()`

Math.cos()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

`Math.cos(x:Number) : Number`

Parameters

x A number that represents an angle measured in radians.

Returns

A number from -1.0 to 1.0.

Description

Method; computes and returns the cosine of the specified angle in radians. To calculate a radian, see "Description" on page 419 of the Math class entry.

Example

The following example displays the cosine for several different angles.

```
trace(Math.cos(0)); // output: 1
trace(Math.cos(90)); // output: -0.44807361612917
trace(Math.cos(180)); // output: -0.598460069057858
trace(Math.cos(360)); // output: -0.283691091486527
```

See also

`Math.acos()`, `Math.asin()`, `Math.atan()`, `Math.atan2()`, `Math.sin()`, `Math.tan()`

Math.E

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

`Math.E:`*`Number`*

Description

Constant; a mathematical constant for the base of natural logarithms, expressed as *e*. The approximate value of *e* is 2.71828182845905.

Example

This example shows how `Math.E` is used to compute continuously compounded interest for a simple case of 100 percent interest over a one-year period.

```
var principal:Number = 100;
var simpleInterest:Number = 100;
var continuouslyCompoundedInterest:Number = (100 * Math.E) - principal;

trace ("Beginning principal: $" + principal);
trace ("Simple interest after one year: $" + simpleInterest);
trace ("Continuously compounded interest after one year: $" +
  continuouslyCompoundedInterest);

/*
Output:
Beginning principal: $100
Simple interest after one year: $100
Continuously compounded interest after one year: $171.828182845905
*/
```

Math.exp()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

```
Math.exp(x:Number) : Number
```

Parameters

`x` The exponent; a number or expression.

Returns

A number.

Description

Method; returns the value of the base of the natural logarithm (*e*), to the power of the exponent specified in the parameter `x`. The constant `Math.E` can provide the value of *e*.

Example

The following example displays the logarithm for two number values.

```
trace(Math.exp(1)); // output: 2.71828182845905
trace(Math.exp(2)); // output: 7.38905609893065
```

Math.floor()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

```
Math.floor(x:Number) : Number
```

Parameters

x A number or expression.

Returns

The integer that is both closest to, and less than or equal to, parameter *x*.

Description

Method; returns the floor of the number or expression specified in the parameter *x*. The floor is the closest integer that is less than or equal to the specified number or expression.

Example

The following code returns a value of 12:

```
Math.floor(12.5);
```

The following code returns a value of -7:

```
Math.floor(-6.5);
```

Math.log()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

```
Math.log(x:Number) : Number
```

Parameters

x A number or expression with a value greater than 0.

Returns

The logarithm of parameter *x*.

Description

Method; returns the logarithm of parameter *x*.

Example

The following example displays the logarithm for three numerical values.

```
trace(Math.log(0)); // output: -Infinity
trace(Math.log(1)); // output: 0
trace(Math.log(2)); // output: 0.693147180559945
```

Math.LN2

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

```
Math.LN2:Number
```

Description

Constant; a mathematical constant for the natural logarithm of 2, expressed as $\log_e 2$, with an approximate value of 0.6931471805599453.

Math.LN10

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

`Math.LN10:`*`Number`*

Description

Constant; a mathematical constant for the natural logarithm of 10, expressed as $\log_e 10$, with an approximate value of 2.302585092994046.

Example

This example traces the value of `Math.LN10`.

```
trace(Math.LN10);
// output: 2.30258509299405
```

Math.LOG2E

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

```
Math.LOG2E:Number
```

Parameters

None.

Returns

Nothing.

Description

Constant; a mathematical constant for the base-2 logarithm of the constant *e* (`Math.E`), expressed as $\log2_e$, with an approximate value of 1.442695040888963387.

Example

This example traces the value of `Math.LOG2E`.

```
trace(Math.LOG2E);
// Output: 1.44269504088896
```

Math.LOG10E

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

`Math.LOG10E:Number`

Description

Constant; a mathematical constant for the base-10 logarithm of the constant *e* (`Math.E`), expressed as $\log_{10}e$, with an approximate value of 0.4342944819032518.

Example

This example traces the value of `Math.LOG10E`.

```
trace(Math.LOG10E);
// Output: 0.434294481903252
```

Math.max()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

```
Math.max(x:Number , y:Number) : Number
```

Parameters

x A number or expression.

y A number or expression.

Returns

A number.

Description

Method; evaluates *x* and *y* and returns the larger value.

Example

The following example displays `Thu Dec 30 00:00:00 GMT-0700 2004`, which is the larger of the evaluated expressions.

```
var date1:Date = new Date(2004, 11, 25);
var date2:Date = new Date(2004, 11, 30);
var maxDate:Number = Math.max(date1.getTime(), date2.getTime());
trace(new Date(maxDate).toString());
```

See Also

`Math.min()`

Math.min()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

`Math.min(`*`x:Number , y:Number`*`) :`*` Number`*

Parameters

x A number or expression.

y A number or expression.

Returns

A number.

Description

Method; evaluates *x* and *y* and returns the smaller value.

Example

The following example displays `Sat Dec 25 00:00:00 GMT-0700 2004`, which is the smaller of the evaluated expressions.

```
var date1:Date = new Date(2004, 11, 25);
var date2:Date = new Date(2004, 11, 30);
var minDate:Number = Math.min(date1.getTime(), date2.getTime());
trace(new Date(minDate).toString());
```

See Also

`Math.max()`

Math.PI

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

```
Math.PI:Number
```

Parameters

None.

Returns

Nothing.

Description

Constant; a mathematical constant for the ratio of the circumference of a circle to its diameter, expressed as pi, with a value of 3.141592653589793.

Example

The following example draws a circle using the mathematical constant pi and the Drawing API.

```
drawCircle(this, 100, 100, 50);
//
function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
  mc.lineStyle(2, 0xFF0000, 100);
  mc.moveTo(x+r, y);
  mc.curveTo(r+x, Math.tan(Math.PI/8)*r+y, Math.sin(Math.PI/4)*r+x,
  Math.sin(Math.PI/4)*r+y);
  mc.curveTo(Math.tan(Math.PI/8)*r+x, r+y, x, r+y);
  mc.curveTo(-Math.tan(Math.PI/8)*r+x, r+y, -Math.sin(Math.PI/4)*r+x,
  Math.sin(Math.PI/4)*r+y);
  mc.curveTo(-r+x, Math.tan(Math.PI/8)*r+y, -r+x, y);
  mc.curveTo(-r+x, -Math.tan(Math.PI/8)*r+y, -Math.sin(Math.PI/4)*r+x, -
  Math.sin(Math.PI/4)*r+y);
  mc.curveTo(-Math.tan(Math.PI/8)*r+x, -r+y, x, -r+y);
  mc.curveTo(Math.tan(Math.PI/8)*r+x, -r+y, Math.sin(Math.PI/4)*r+x, -
  Math.sin(Math.PI/4)*r+y);
  mc.curveTo(r+x, -Math.tan(Math.PI/8)*r+y, r+x, y);
}
```

Math.pow()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

`Math.pow(`*`x:Number`*` , `*`y:Number`*`) : `*`Number`*

Parameters

x A number to be raised to a power.

y A number specifying a power the parameter *x* is raised to.

Returns

A number.

Description

Method; computes and returns *x* to the power of *y*.

Example

The following example uses `Math.pow` and `Math.sqrt` to calculate the length of a line.

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
  this.origX = _xmouse;
  this.origY = _ymouse;
};
mouseListener.onMouseUp = function() {
  this.newX = _xmouse;
  this.newY = _ymouse;
  var minY = Math.min(this.origY, this.newY);
  var nextDepth:Number = canvas_mc.getNextHighestDepth();
  var line_mc:MovieClip =
  canvas_mc.createEmptyMovieClip("line"+nextDepth+"_mc", nextDepth);
  line_mc.moveTo(this.origX, this.origY);
  line_mc.lineStyle(2, 0x000000, 100);
  line_mc.lineTo(this.newX, this.newY);
  var hypLen:Number = Math.sqrt(Math.pow(line_mc._width,
  2)+Math.pow(line_mc._height, 2));
  line_mc.createTextField("length"+nextDepth+"_txt",
  canvas_mc.getNextHighestDepth(), this.origX, this.origY-22, 100, 22);
  line_mc['length'+nextDepth+'_txt'].text = Math.round(hypLen) +" pixels";
};
Mouse.addListener(mouseListener);
```

Math.random()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

```
Math.random() : Number
```

Parameters

None.

Returns

A number.

Description

Method; returns a pseudo-random number n, where 0 <= n < 1. The number returned is a pseudo-random number because it is not generated by a truly random natural phenomenon such as radioactive decay.

Example

The following example returns a random number between two specified integers.

```
function randRange(min:Number, max:Number):Number {
  var randomNum:Number = Math.round(Math.random()*(max-min))+min;
  return randomNum;
}
for (var i = 0; i<25; i++) {
  trace(randRange(4, 11));
}
```

Math.round()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

`Math.round(x:Number) : Number`

Parameters

x A number.

Returns

A number; an integer.

Description

Method; rounds the value of the parameter *x* up or down to the nearest integer and returns the value. If parameter *x* is equidistant from its two nearest integers (that is, the number ends in .5), the value is rounded up to the next higher integer.

Example

The following example returns a random number between two specified integers.

```
function randRange(min:Number, max:Number):Number {
  var randomNum:Number = Math.round(Math.random()*(max-min))+min;
  return randomNum;
}
for (var i = 0; i<25; i++) {
  trace(randRange(4, 11));
}
```

See Also

`Math.ceil()`, `Math.floor()`

Math.sin()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

```
Math.sin(x:Number) : Number
```

Parameters

x A number that represents an angle measured in radians.

Returns

A number; the sine of the specified angle (between -1.0 and 1.0).

Description

Method; computes and returns the sine of the specified angle in radians. To calculate a radian, see "Description" on page 419 of the Math class entry.

Example

The following example draws a circle using the mathematical constant pi, the sine of an angle, and the Drawing API.

```
drawCircle(this, 100, 100, 50);
//
function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
  mc.lineStyle(2, 0xFF0000, 100);
  mc.moveTo(x+r, y);
  mc.curveTo(r+x, Math.tan(Math.PI/8)*r+y, Math.sin(Math.PI/4)*r+x,
  Math.sin(Math.PI/4)*r+y);
  mc.curveTo(Math.tan(Math.PI/8)*r+x, r+y, x, r+y);
  mc.curveTo(-Math.tan(Math.PI/8)*r+x, r+y, -Math.sin(Math.PI/4)*r+x,
  Math.sin(Math.PI/4)*r+y);
  mc.curveTo(-r+x, Math.tan(Math.PI/8)*r+y, -r+x, y);
  mc.curveTo(-r+x, -Math.tan(Math.PI/8)*r+y, -Math.sin(Math.PI/4)*r+x, -
  Math.sin(Math.PI/4)*r+y);
  mc.curveTo(-Math.tan(Math.PI/8)*r+x, -r+y, x, -r+y);
  mc.curveTo(Math.tan(Math.PI/8)*r+x, -r+y, Math.sin(Math.PI/4)*r+x, -
  Math.sin(Math.PI/4)*r+y);
  mc.curveTo(r+x, -Math.tan(Math.PI/8)*r+y, r+x, y);
}
```

See also

`Math.acos()`, `Math.asin()`, `Math.atan()`, `Math.atan2()`, `Math.cos()`, `Math.tan()`

Math.sqrt()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

`Math.sqrt(`*`x:Number`*`) : `*`Number`*

Parameters

x A number or expression greater than or equal to 0.

Returns

A number if parameter *x* is greater than or equal to zero; NaN (not a number) otherwise.

Description

Method; computes and returns the square root of the specified number.

Example

The following example uses `Math.pow` and `Math.sqrt` to calculate the length of a line.

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
  this.origX = _xmouse;
  this.origY = _ymouse;
};
mouseListener.onMouseUp = function() {
  this.newX = _xmouse;
  this.newY = _ymouse;
  var minY = Math.min(this.origY, this.newY);
  var nextDepth:Number = canvas_mc.getNextHighestDepth();
  var line_mc:MovieClip =
  canvas_mc.createEmptyMovieClip("line"+nextDepth+"_mc", nextDepth);
  line_mc.moveTo(this.origX, this.origY);
  line_mc.lineStyle(2, 0x000000, 100);
  line_mc.lineTo(this.newX, this.newY);
  var hypLen:Number = Math.sqrt(Math.pow(line_mc._width,
  2)+Math.pow(line_mc._height, 2));
  line_mc.createTextField("length"+nextDepth+"_txt",
  canvas_mc.getNextHighestDepth(), this.origX, this.origY-22, 100, 22);
  line_mc['length'+nextDepth+'_txt'].text = Math.round(hypLen) +" pixels";
};
Mouse.addListener(mouseListener);
```

Math.SQRT1_2

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

`Math.SQRT1_2:`*`Number`*

Description

Constant; a mathematical constant for the square root of one-half, with an approximate value of 0.7071067811865476.

Example

This example traces the value of `Math.SQRT1_2`.

```
trace(Math.SQRT1_2);
// Output: 0.707106781186548
```

Math.SQRT2

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

```
Math.SQRT2
```

Description

Constant; a mathematical constant for the square root of 2, with an approximate value of 1.4142135623730951.

Example

This example traces the value of `Math.SQRT2`.

```
trace(Math.SQRT2);
// Output: 1.4142135623731
```

Math.tan()

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math class are emulated using approximations and might not be as accurate as the non-emulated math functions that Flash Player 5 supports.

Usage

```
Math.tan(x:Number)
```

Parameters

x A number that represents an angle measured in radians.

Returns

A number; tangent of parameter *x*.

Description

Method; computes and returns the tangent of the specified angle. To calculate a radian, use the information outlined in the introduction to the Math class.

Example

The following example draws a circle using the mathematical constant pi, the tangent of an angle, and the Drawing API.

```
drawCircle(this, 100, 100, 50);
//
function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
  mc.lineStyle(2, 0xFF0000, 100);
  mc.moveTo(x+r, y);
  mc.curveTo(r+x, Math.tan(Math.PI/8)*r+y, Math.sin(Math.PI/4)*r+x,
  Math.sin(Math.PI/4)*r+y);
  mc.curveTo(Math.tan(Math.PI/8)*r+x, r+y, x, r+y);
  mc.curveTo(-Math.tan(Math.PI/8)*r+x, r+y, -Math.sin(Math.PI/4)*r+x,
  Math.sin(Math.PI/4)*r+y);
  mc.curveTo(-r+x, Math.tan(Math.PI/8)*r+y, -r+x, y);
  mc.curveTo(-r+x, -Math.tan(Math.PI/8)*r+y, -Math.sin(Math.PI/4)*r+x, -
  Math.sin(Math.PI/4)*r+y);
  mc.curveTo(-Math.tan(Math.PI/8)*r+x, -r+y, x, -r+y);
  mc.curveTo(Math.tan(Math.PI/8)*r+x, -r+y, Math.sin(Math.PI/4)*r+x, -
  Math.sin(Math.PI/4)*r+y);
  mc.curveTo(r+x, -Math.tan(Math.PI/8)*r+y, r+x, y);
}
```

See also

`Math.acos()`, `Math.asin()`, `Math.atan()`, `Math.atan2()`, `Math.cos()`, `Math.sin()`

Microphone class

Availability

Flash Player 6.

Description

The Microphone class lets you capture audio from a microphone attached to the computer that is running Flash Player.

The Microphone class is primarily for use with Flash Communication Server but can be used in a limited fashion without the server—for example, to transmit sound from your microphone through the speakers on your local system.

Caution: Flash Player displays a Privacy dialog box that lets the user choose whether to allow or deny access to the microphone. Make sure your Stage size is at least 215 x 138 pixels; this is the minimum size Flash requires to display the dialog box.

To create or reference a Microphone object, use the `Microphone.get()` method.

Method summary for the Microphone class

Method	Description
`Microphone.get()`	Returns a default or specified Microphone object, or `null` if the microphone is not available.
`Microphone.setGain()`	Specifies the amount by which the microphone should boost the signal.
`Microphone.setRate()`	Specifies the rate at which the microphone should capture sound, in kHz.
`Microphone.setSilenceLevel()`	Specifies the amount of sound required to activate the microphone.
`Microphone.setUseEchoSuppression()`	Specifies whether to use the echo suppression feature of the audio codec.

Property summary for the Microphone class

Property (read-only)	Description
`Microphone.activityLevel`	The amount of sound the microphone is detecting.
`Microphone.gain`	The amount by which the microphone boosts the signal before transmitting it.
`Microphone.index`	The index of the current microphone.
`Microphone.muted`	A Boolean value that specifies whether the user has allowed or denied access to the microphone.
`Microphone.name`	The name of the current sound capture device, as returned by the sound capture hardware.

Property (read-only)	Description
`Microphone.names`	Class property; an array of strings reflecting the names of all available sound capture devices, including sound cards and microphones.
`Microphone.rate`	The sound capture rate, in kHz.
`Microphone.silenceLevel`	The amount of sound required to activate the microphone.
`Microphone.silenceTimeOut`	The number of milliseconds between the time the microphone stops detecting sound and `Microphone.onActivity(false)` is called.
`Microphone.useEchoSuppression`	A Boolean value that specifies whether echo suppression is being used.

Event handler summary for the Microphone class

Event handler	Description
`Microphone.onActivity`	Invoked when the microphone starts or stops detecting sound.
`Microphone.onStatus`	Invoked when the user allows or denies access to the microphone.

Constructor for the Microphone class

See `Microphone.get()`.

Microphone.activityLevel

Availability

Flash Player 6.

Usage

active_mic.activityLevel*:Number*

Description

Read-only property; a numeric value that specifies the amount of sound the microphone is detecting. Values range from 0 (no sound is being detected) to 100 (very loud sound is being detected). The value of this property can help you determine a good value to pass to the `Microphone.setSilenceLevel()` method.

If the microphone is available but is not yet being used because `Microphone.get()` has not been called, this property is set to -1.

Example

The following example displays the activity level of the current microphone in a ProgressBar instance called `activityLevel_pb`.

```
var activityLevel_pb:mx.controls.ProgressBar;
activityLevel_pb.mode = "manual";
activityLevel_pb.label = "Activity Level: %3%%";
activityLevel_pb.setStyle("themeColor", "0xFF0000");
this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);
this.onEnterFrame = function() {
  activityLevel_pb.setProgress(active_mic.activityLevel, 100);
};
active_mic.onActivity = function(active:Boolean) {
  if (active) {
    var haloTheme_str:String = "haloGreen";
  } else {
    var haloTheme_str:String = "0xFF0000";
  }
  activityLevel_pb.setStyle("themeColor", haloTheme_str);
};
```

See also

`Microphone.setGain()`

Microphone.gain

Availability

Flash Player 6.

Usage

active_mic.gain:*Number*

Description

Read-only property; the amount by which the microphone boosts the signal. Valid values are 0 to 100. The default value is 50.

Example

The following example uses a ProgressBar instance called `gain_pb` to display and a NumericStepper instance called `gain_nstep` to set the microphone's gain value.

```
this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);

gain_pb.label = "Gain: %3";
gain_pb.mode = "manual";
gain_pb.setProgress(active_mic.gain, 100);
gain_nstep.value = active_mic.gain;

function changeGain() {
  active_mic.setGain(gain_nstep.value);
  gain_pb.setProgress(active_mic.gain, 100);
}
gain_nstep.addEventListener("change", changeGain);
```

See also

`Microphone.setGain()`

Microphone.get()

Availability

Flash Player 6.

Usage

`Microphone.get([index:Number]) : Microphone`

Note: The correct syntax is `Microphone.get()`. To assign the Microphone object to a variable, use syntax like `active_mic = Microphone.get()`.

Parameters

`index` An optional zero-based integer that specifies which microphone to get, as determined from the array that Microphone.names contains. To get the default microphone (which is recommended for most applications), omit this parameter.

Returns

- If *`index`* is not specified, this method returns a reference to the default microphone or, if it is not available, to the first available microphone. If no microphones are available or installed, the method returns `null`.
- If *`index`* is specified, this method returns a reference to the requested microphone, or `null` if it is not available.

Description

Method; returns a reference to a Microphone object for capturing audio. To actually begin capturing the audio, you must attach the Microphone object to a MovieClip object (see `MovieClip.attachAudio()`).

Unlike objects that you create using the `new` constructor, multiple calls to `Microphone.get()` reference the same microphone. Thus, if your script contains the lines `mic1 = Microphone.get()` and `mic2 = Microphone.get()`, both `mic1` and `mic2` reference the same (default) microphone.

In general, you shouldn't pass a value for *`index`*; simply use the `Microphone.get()` method to return a reference to the default microphone. By means of the Microphone settings panel (discussed later in this section), the user can specify the default microphone Flash should use. If you pass a value for *`index`*, you might be trying to reference a microphone other than the one the user prefers. You might use *`index`* in rare cases—for example, if your application is capturing audio from two microphones at the same time.

When a SWF file tries to access the microphone returned by the `Microphone.get()` method—for example, when you issue `MovieClip.attachAudio()`—Flash Player displays a Privacy dialog box that lets the user choose whether to allow or deny access to the microphone. (Make sure your Stage size is at least 215 x 138 pixels; this is the minimum size Flash requires to display the dialog box.)

When the user responds to this dialog box, the `Microphone.onStatus` event handler returns an information object that indicates the user's response. To determine whether the user has denied or allowed access to the camera without processing this event handler, use `Microphone.muted`.

The user can also specify permanent privacy settings for a particular domain by right-clicking (Windows) or Control-clicking (Macintosh) while a SWF file is playing, choosing Settings, opening the Privacy panel, and selecting Remember.

You can't use ActionScript to set the Allow or Deny value for a user, but you can display the Privacy panel for the user by using `System.showSettings(0)`. If the user selects Remember, Flash Player no longer displays the Privacy dialog box for SWF files from this domain.

If `Microphone.get()` returns `null`, either the microphone is in use by another application, or there are no microphones installed on the system. To determine whether any microphones are installed, use `Microphones.names.length`. To display the Flash Player Microphone Settings panel, which lets the user choose the microphone to be referenced by `Microphone.get()`, use `System.showSettings(2)`.

Example

The following example lets the user specify the default microphone, and then captures audio and plays it back locally. To avoid feedback, you may want to test this code while wearing headphones.

```
this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
System.showSettings(2);
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);
```

See also

`Microphone.index`, `Microphone.muted`, `Microphone.names`, `Microphone.onStatus`, `MovieClip.attachAudio()`, `System.showSettings()`

Microphone.index

Availability

Flash Player 6.

Usage

```
active_mic.index:Number
```

Description

Read-only property; a zero-based integer that specifies the index of the microphone, as reflected in the array returned by Microphone.names.

Example

The following example displays the names of the sound capturing devices available on your computer system in a ComboBox instance called `mic_cb`. An instance of the Label component, called `mic_lbl`, displays the index microphone. You can use the ComboBox to switch between the devices.

```
var mic_lbl:mx.controls.Label;
var mic_cb:mx.controls.ComboBox;

this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);
mic_lbl.text = "["+active_mic.index+"] "+active_mic.name;
mic_cb.dataProvider = Microphone.names;
mic_cb.selectedIndex = active_mic.index;

var cbListener:Object = new Object();
cbListener.change = function(evt:Object) {
  active_mic = Microphone.get(evt.target.selectedIndex);
  sound_mc.attachAudio(active_mic);
  mic_lbl.text = "["+active_mic.index+"] "+active_mic.name;
};
mic_cb.addEventListener("change", cbListener);
```

See also

`Microphone.get()`, `Microphone.names`

Microphone.muted

Availability

Flash Player 6.

Usage

active_mic.muted:*Boolean*

Description

Read-only property; a Boolean value that specifies whether the user has denied access to the microphone (`true`) or allowed access (`false`). When this value changes, Microphone.onStatus is invoked. For more information, see Microphone.get().

Example

This example gets the default microphone and checks whether it is muted.

```
var active_mic:Microphone = Microphone.get();
trace(active_mic.muted);
```

See also

`Microphone.get()`, `Microphone.onStatus`

Microphone.name

Availability

Flash Player 6.

Usage

active_mic.name:*String*

Description

Read-only property; a string that specifies the name of the current sound capture device, as returned by the sound capture hardware.

Example

The following example displays information about the sound capturing device(s) on your computer system, including an array of names and the default device.

```
var status_ta:mx.controls.TextArea;
status_ta.html = false;
status_ta.setStyle("fontSize", 9);
var microphone_array:Array = Microphone.names;
var active_mic:Microphone = Microphone.get();
status_ta.text = "The default device is: "+active_mic.name+newline+newline;
status_ta.text += "You have "+microphone_array.length+" device(s)
  installed."+newline+newline;
for (var i = 0; i<microphone_array.length; i++) {
  status_ta.text += "["+i+"] "+microphone_array[i]+newline;
}
```

See also

Microphone.get(), Microphone.names

Microphone.names

Availability

Flash Player 6.

Usage

`Microphone.names:Array`

Note: The correct syntax is `Microphone.names`. To assign the return value to a variable, use syntax like *`mic_array`* `= Microphone.names`. To determine the name of the current microphone, use *`active_mic`*`.name`.

Description

Read-only class property; retrieves an array of strings reflecting the names of all available sound capture devices without displaying the Flash Player Privacy Settings panel. This array behaves the same as any other ActionScript array, implicitly providing the zero-based index of each sound capture device and the number of sound capture devices on the system (by means of `Microphone.names.length`). For more information, see the Array class entry.

Calling `Microphone.names` requires an extensive examination of the hardware, and it may take several seconds to build the array. In most cases, you can just use the default microphone.

Example

The following example displays information about the sound capturing device(s) on your computer system, including an array of names and the default device.

```
var status_ta:mx.controls.TextArea;
status_ta.html = false;
status_ta.setStyle("fontSize", 9);
var microphone_array:Array = Microphone.names;
var active_mic:Microphone = Microphone.get();
status_ta.text = "The default device is: "+active_mic.name+newline+newline;
status_ta.text += "You have "+microphone_array.length+" device(s)
  installed."+newline+newline;
for (var i = 0; i<microphone_array.length; i++) {
  status_ta.text += "["+i+"] "+microphone_array[i]+newline;
}
```

For example, the following information could be displayed:

```
The default device is: Logitech USB Headset
You have 2 device(s) installed.
[0] Logitech USB Headset
[1] YAMAHA AC-XG WDM Audio
```

See also

`Array class`, `Microphone.name`

Microphone.onActivity

Availability

Flash Player 6.

Usage

```
active_mic.onActivity = function(activity:Boolean) : Void {
  // your statements here
}
```

Parameters

activity A Boolean value set to `true` when the microphone starts detecting sound, and `false` when it stops.

Returns

Nothing.

Description

Event handler; invoked when the microphone starts or stops detecting sound. If you want to respond to this event handler, you must create a function to process its *activity* value.

To specify the amount of sound required to invoke `Microphone.onActivity(true)`, and the amount of time that must elapse without sound before `Microphone.onActivity(false)` is invoked, use Microphone.setSilenceLevel().

Example

The following example displays the amount of activity level in a ProgressBar instance called `activityLevel_pb`. When the microphone detects sound, it invokes the `onActivity` function, which modifies the ProgressBar instance.

```
var activityLevel_pb:mx.controls.ProgressBar;
activityLevel_pb.mode = "manual";
activityLevel_pb.label = "Activity Level: %3%%";
this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);
active_mic.onActivity = function(active:Boolean) {
  if (active) {
    activityLevel_pb.indeterminate = false;
    activityLevel_pb.label = "Activity Level: %3%%";
  } else {
    activityLevel_pb.indeterminate = true;
    activityLevel_pb.label = "Activity Level: (inactive)";
  }
};
this.onEnterFrame = function() {
  activityLevel_pb.setProgress(active_mic.activityLevel, 100);
};
```

See also

`Microphone.setSilenceLevel()`

Microphone.onStatus

Availability

Flash Player 6.

Usage

```
active_mic.onStatus = function(infoObject:Object) : Void {
  // your statements here
}
```

Parameters

`infoObject` A parameter defined according to the status message.

Returns

Nothing.

Description

Event handler; invoked when the user allows or denies access to the microphone. If you want to respond to this event handler, you must create a function to process the information object generated by the microphone.

When a SWF file tries to access the microphone, Flash Player displays a Privacy dialog box that lets the user choose whether to allow or deny access.

- If the user allows access, the Microphone.muted property is set to `false`, and this event handler is invoked with an information object whose `code` property is `"Microphone.Unmuted"` and whose `level` property is `"Status"`.
- If the user denies access, the Microphone.muted property is set to `true`, and this event handler is invoked with an information object whose `code` property is `"Microphone.Muted"` and whose `level` property is `"Status"`.

To determine whether the user has denied or allowed access to the microphone without processing this event handler, use Microphone.muted.

Note: If the user chooses to permanently allow or deny access to all SWF files from a specified domain, this method is not invoked for SWF files from that domain unless the user later changes the privacy setting. For more information, see `Microphone.get()`.

Example

The following example launches the Privacy dialog box that lets the user choose whether to allow or deny access to the microphone when they click a hyperlink. If the user chooses to deny access, *muted* is displayed in large red text. If microphone access is allowed, the user does not see this text.

```
this.createTextField("muted_txt", this.getNextHighestDepth(), 10, 10, 100,
  22);
muted_txt.autoSize = true;
muted_txt.html = true;
muted_txt.selectable = false;
muted_txt.htmlText = "<a href=\"asfunction:System.showSettings\"><u>Click
  Here</u></a> to Allow/Deny access.";
this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
```

```
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);
active_mic.onStatus = function(infoObj:Object) {
  status_txt._visible = active_mic.muted;
  muted_txt.htmlText = "Status: <a
  href=\"asfunction:System.showSettings\"><u>"+infoObj.code+"</u></a>";
};
this.createTextField("status_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
status_txt.html = true;
status_txt.autoSize = true;
status_txt.htmlText = "<font size='72' color='#FF0000'>muted</font>";
status_txt._x = (Stage.width-status_txt._width)/2;
status_txt._y = (Stage.height-status_txt._height)/2;
status_txt._visible = active_mic.muted;
```

See also

Microphone.get(), Microphone.muted

Microphone.rate

Availability

Flash Player 6.

Usage

active_mic.rate:*Number*

Description

Read-only property; the rate at which the microphone is capturing sound, in kHz. The default value is 8 kHz if your sound capture device supports this value. Otherwise, the default value is the next available capture level above 8 kHz that your sound capture device supports, usually 11 kHz.

To set this value, use Microphone.setRate().

Example

The following code lets you use a ComboBox instance, called `rate_cb`, to change the rate at which your microphone captures sound. The current rate displays in a Label instance called `rate_lbl`.

```
this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);
var rate_array:Array = new Array(5, 8, 11, 22, 44);
rate_cb.dataProvider = rate_array;
rate_cb.labelFunction = function(item:Object) {
  return (item+" kHz");
};
for (var i = 0; i<rate_array.length; i++) {
  if (rate_cb.getItemAt(i) == active_mic.rate) {
    rate_cb.selectedIndex = i;
    break;
  }
}
function changeRate() {
  active_mic.setRate(rate_cb.selectedItem);
  rate_lbl.text = "Current rate: "+active_mic.rate+" kHz";
}
rate_cb.addEventListener("change", changeRate);
rate_lbl.text = "Current rate: "+active_mic.rate+" kHz";
```

See also

`Microphone.setRate()`

Microphone.setGain()

Availability

Flash Player 6.

Usage

`active_mic``.setGain(`*`gain:Number`*`) : Void`

Parameters

`gain` An integer that specifies the amount by which the microphone should boost the signal. Valid values are 0 to 100. The default value is 50; however, the user may change this value in the Flash Player Microphone Settings panel.

Returns

Nothing.

Description

Method; sets the microphone gain—that is, the amount by which the microphone should multiply the signal before transmitting it. A value of 0 tells Flash to multiply by 0; that is, the microphone transmits no sound.

You can think of this setting like a volume knob on a stereo: 0 is no volume and 50 is normal volume; numbers below 50 specify lower than normal volume, while numbers above 50 specify higher than normal volume.

Example

The following example uses a ProgressBar instance called `gain_pb` to display and a NumericStepper instance called `gain_nstep` to set the microphone's gain value.

```
this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);

gain_pb.label = "Gain: %3";
gain_pb.mode = "manual";
gain_pb.setProgress(active_mic.gain, 100);
gain_nstep.value = active_mic.gain;

function changeGain() {
  active_mic.setGain(gain_nstep.value);
  gain_pb.setProgress(active_mic.gain, 100);
}
gain_nstep.addEventListener("change", changeGain);
```

See also

`Microphone.gain`, `Microphone.setUseEchoSuppression()`

Microphone.setRate()

Availability

Flash Player 6.

Usage

active_mic.setRate(*kHz:Number*) *: Void*

Parameters

kHz The rate at which the microphone should capture sound, in kHz. Acceptable values are 5, 8, 11, 22, and 44. The default value is 8 kHz if your sound capture device supports this value. Otherwise, the default value is the next available capture level above 8 kHz that your sound capture device supports, usually 11 kHz.

Returns

Nothing.

Description

Method; sets the rate, in kHz, at which the microphone should capture sound.

Example

The following example sets the microphone rate to the user's preference (which you have assigned to the userRate variable) if it is one of the following values: 5, 8, 11, 22, or 44. If it is not, the value is rounded to the nearest acceptable value that the sound capture device supports.

```
active_mic.setRate(userRate);
```

The following example lets you use a ComboBox instance, called rate_cb, to change the rate at which your microphone captures sound. The current rate displays in a Label instance called rate_lbl.

```
this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);
var rate_array:Array = new Array(5, 8, 11, 22, 44);
rate_cb.dataProvider = rate_array;
rate_cb.labelFunction = function(item:Object) {
  return (item+" kHz");
};
for (var i = 0; i<rate_array.length; i++) {
  if (rate_cb.getItemAt(i) == active_mic.rate) {
    rate_cb.selectedIndex = i;
    break;
  }
}
function changeRate() {
  active_mic.setRate(rate_cb.selectedItem);
  rate_lbl.text = "Current rate: "+active_mic.rate+" kHz";
}
rate_cb.addEventListener("change", changeRate);
rate_lbl.text = "Current rate: "+active_mic.rate+" kHz";
```

See also

`Microphone.rate`

Microphone.setSilenceLevel()

Availability

Flash Player 6.

Usage

```
active_mic.setSilenceLevel(level:Number [, timeout:Number]) : Void
```

Parameters

`level` An integer that specifies the amount of sound required to activate the microphone and invoke `Microphone.onActivity(true)`. Acceptable values range from 0 to 100. The default value is 10.

`timeout` An optional integer parameter that specifies how many milliseconds must elapse without activity before Flash considers sound to have stopped and invokes `Microphone.onActivity(false)`. The default value is 2000 (2 seconds).

Returns

Nothing.

Description

Method; sets the minimum input level that should be considered sound and (optionally) the amount of silent time signifying that silence has actually begun.

- To prevent the microphone from detecting sound at all, pass a value of 100 for `level`; Microphone.onActivity is never invoked.
- To determine the amount of sound the microphone is currently detecting, use Microphone.activityLevel.

Activity detection is the ability to detect when audio levels suggest that a person is talking. When someone is not talking, bandwidth can be saved because there is no need to send the associated audio stream. This information can also be used for visual feedback so that users know they (or others) are silent.

Silence values correspond directly to activity values. Complete silence is an activity value of 0. Constant loud noise (as loud as can be registered based on the current gain setting) is an activity value of 100. After gain is appropriately adjusted, your activity value is less than your silence value when you're not talking; when you are talking, the activity value exceeds your silence value.

This method is similar in purpose to Camera.setMotionLevel(); both methods are used to specify when the `onActivity` event handler should be invoked. However, these methods have a significantly different impact on publishing streams:

- Camera.setMotionLevel() is designed to detect motion and does not affect bandwidth usage. Even if a video stream does not detect motion, video is still sent.
- Microphone.setSilenceLevel() is designed to optimize bandwidth. When an audio stream is considered silent, no audio data is sent. Instead, a single message is sent, indicating that silence has started.

Example

The following example changes the silence level based on the user's input in a NumericStepper instance called `silenceLevel_nstep`. The ProgressBar instance called `silenceLevel_pb` modifies its appearance depending on whether the audio stream is considered silent. Otherwise, it displays the activity level of the audio stream.

```
var silenceLevel_pb:mx.controls.ProgressBar;
var silenceLevel_nstep:mx.controls.NumericStepper;

this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);

silenceLevel_pb.label = "Activity level: %3";
silenceLevel_pb.mode = "manual";
silenceLevel_nstep.minimum = 0;
silenceLevel_nstep.maximum = 100;
silenceLevel_nstep.value = active_mic.silenceLevel;

var nstepListener:Object = new Object();
nstepListener.change = function(evt:Object) {
  active_mic.setSilenceLevel(evt.target.value, active_mic.silenceTimeOut);
};
silenceLevel_nstep.addEventListener("change", nstepListener);

this.onEnterFrame = function() {
  silenceLevel_pb.setProgress(active_mic.activityLevel, 100);
};
active_mic.onActivity = function(active:Boolean) {
  if (active) {
    silenceLevel_pb.indeterminate = false;
    silenceLevel_pb.setStyle("themeColor", "haloGreen");
    silenceLevel_pb.label = "Activity level: %3";
  } else {
    silenceLevel_pb.indeterminate = true;
    silenceLevel_pb.setStyle("themeColor", "0xFF0000");
    silenceLevel_pb.label = "Activity level: (inactive)";
  }
};
```

For more information, see the example for Camera.setMotionLevel().

See also

`Microphone.activityLevel`, `Microphone.onActivity`, `Microphone.setGain()`, `Microphone.silenceLevel`, `Microphone.silenceTimeOut`

Microphone.setUseEchoSuppression()

Availability

Flash Player 6.

Usage

`active_mic.setUseEchoSuppression(suppress:Boolean) : Void`

Parameters

suppress A Boolean value indicating whether echo suppression should be used (`true`) or not (`false`).

Returns

Nothing.

Description

Method; specifies whether to use the echo suppression feature of the audio codec. The default value is `false` unless the user has selected Reduce Echo in the Flash Player Microphone Settings panel.

Echo suppression is an effort to reduce the effects of audio feedback, which is caused when sound going out the speaker is picked up by the microphone on the same computer. (This is different from echo cancellation, which completely removes the feedback.)

Generally, echo suppression is advisable when the sound being captured is played through speakers—instead of a headset—on the same computer. If your SWF file allows users to specify the sound output device, you may want to call `Microphone.setUseEchoSuppression(true)` if they indicate they are using speakers and will be using the microphone as well.

Users can also adjust these settings in the Flash Player Microphone Settings panel.

Example

The following example turns on echo suppression if the user selects a CheckBox instance called `useEchoSuppression_ch`. The ProgressBar instance called `activityLevel_pb` displays the current activity level of the audio stream.

```
var useEchoSuppression_ch:mx.controls.CheckBox;
var activityLevel_pb:mx.controls.ProgressBar;

this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);

activityLevel_pb.mode = "manual";
activityLevel_pb.label = "Activity Level: %3";
useEchoSuppression_ch.selected = active_mic.useEchoSuppression;
this.onEnterFrame = function() {
  activityLevel_pb.setProgress(active_mic.activityLevel, 100);
};
var chListener:Object = new Object();
chListener.click = function(evt:Object) {
  active_mic.setUseEchoSuppression(evt.target.selected);
};
useEchoSuppression_ch.addEventListener("click", chListener);
```

See also

`Microphone.setGain()`, `Microphone.useEchoSuppression`

Microphone.silenceLevel

Availability

Flash Player 6.

Usage

active_mic.silenceLevel*:Number*

Description

Read-only property; an integer that specifies the amount of sound required to activate the microphone and invoke `Microphone.onActivity(true)`. The default value is 10.

Example

The following example changes the silence level based on the user's input in a NumericStepper instance called `silenceLevel_nstep`. The ProgressBar instance called `silenceLevel_pb` modifies its appearance depending on whether the audio stream is considered silent. Otherwise, it displays the activity level of the audio stream.

```
var silenceLevel_pb:mx.controls.ProgressBar;
var silenceLevel_nstep:mx.controls.NumericStepper;

this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);

silenceLevel_pb.label = "Activity level: %3";
silenceLevel_pb.mode = "manual";
silenceLevel_nstep.minimum = 0;
silenceLevel_nstep.maximum = 100;
silenceLevel_nstep.value = active_mic.silenceLevel;

var nstepListener:Object = new Object();
nstepListener.change = function(evt:Object) {
  active_mic.setSilenceLevel(evt.target.value, active_mic.silenceTimeOut);
};
silenceLevel_nstep.addEventListener("change", nstepListener);

this.onEnterFrame = function() {
  silenceLevel_pb.setProgress(active_mic.activityLevel, 100);
};
active_mic.onActivity = function(active:Boolean) {
  if (active) {
    silenceLevel_pb.indeterminate = false;
    silenceLevel_pb.setStyle("themeColor", "haloGreen");
    silenceLevel_pb.label = "Activity level: %3";
  } else {
    silenceLevel_pb.indeterminate = true;
    silenceLevel_pb.setStyle("themeColor", "0xFF0000");
    silenceLevel_pb.label = "Activity level: (inactive)";
  }
};
```

See also

`Microphone.gain`, `Microphone.setSilenceLevel()`

Microphone.silenceTimeOut

Availability

Flash Player 6.

Usage

`active_mic.silenceTimeOut:Number`

Description

Read-only property; a numeric value representing the number of milliseconds between the time the microphone stops detecting sound and the time `Microphone.onActivity(false)` is invoked. The default value is 2000 (2 seconds).

To set this value, use Microphone.setSilenceLevel().

Example

The following example enables the user to control the amount of time between when the microphone stops detecting sound and when `Microphone.onActivity(false)` is invoked. The user controls this value using a NumericStepper instance called `silenceTimeOut_nstep`. The ProgressBar instance called `silenceLevel_pb` modifies its appearance depending on whether the audio stream is considered silent. Otherwise, it displays the activity level of the audio stream.

```
var silenceLevel_pb:mx.controls.ProgressBar;
var silenceTimeOut_nstep:mx.controls.NumericStepper;

this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);

silenceLevel_pb.label = "Activity level: %3";
silenceLevel_pb.mode = "manual";
silenceTimeOut_nstep.minimum = 0;
silenceTimeOut_nstep.maximum = 10;
silenceTimeOut_nstep.value = active_mic.silenceTimeOut/1000;

var nstepListener:Object = new Object();
nstepListener.change = function(evt:Object) {
  active_mic.setSilenceLevel(active_mic.silenceLevel, evt.target.value*1000);
};
silenceTimeOut_nstep.addEventListener("change", nstepListener);

this.onEnterFrame = function() {
  silenceLevel_pb.setProgress(active_mic.activityLevel, 100);
};
active_mic.onActivity = function(active:Boolean) {
  if (active) {
    silenceLevel_pb.indeterminate = false;
    silenceLevel_pb.setStyle("themeColor", "haloGreen");
    silenceLevel_pb.label = "Activity level: %3";
  } else {
    silenceLevel_pb.indeterminate = true;
    silenceLevel_pb.setStyle("themeColor", "0xFF0000");
```

```
        silenceLevel_pb.label = "Activity level: (inactive)";
      }
    };
```

See also

Microphone.setSilenceLevel()

Microphone.useEchoSuppression

Availability

Flash Player 6.

Usage

`active_mic.useEchoSuppression:Boolean`

Description

Property (read-only); a Boolean value of `true` if echo suppression is enabled, `false` otherwise. The default value is `false` unless the user has selected Reduce Echo in the Flash Player Microphone Settings panel.

Example

The following example turns on echo suppression if the user selects a CheckBox instance called `useEchoSuppression_ch`. The ProgressBar instance called `activityLevel_pb` displays the current activity level of the audio stream.

```
var useEchoSuppression_ch:mx.controls.CheckBox;
var activityLevel_pb:mx.controls.ProgressBar;

this.createEmptyMovieClip("sound_mc", this.getNextHighestDepth());
var active_mic:Microphone = Microphone.get();
sound_mc.attachAudio(active_mic);

activityLevel_pb.mode = "manual";
activityLevel_pb.label = "Activity Level: %3";
useEchoSuppression_ch.selected = active_mic.useEchoSuppression;
this.onEnterFrame = function() {
  activityLevel_pb.setProgress(active_mic.activityLevel, 100);
};
var chListener:Object = new Object();
chListener.click = function(evt:Object) {
  active_mic.setUseEchoSuppression(evt.target.selected);
};
useEchoSuppression_ch.addEventListener("click", chListener);
```

See also

`Microphone.setUseEchoSuppression()`

MMExecute()

Availability

Flash Player 7.

Usage

`MMExecute("`*`Flash JavaScript API command`*`;":`*`String`*`)`

Parameters

`Flash JavaScript API command` Any command that you can use in a Flash JavaScript (JSFL) file.

Returns

A string representation of the result, if any, sent by the JavaScript statement.

Description

Function; lets you issue Flash JavaScript API (JSAPI) commands from ActionScript. In Flash MX2004 the `MMExecute` function can be called only by a movie that is used as a Flash Panel (file is stored in WindowSWF directory), by an XMLtoUI dialog box, or by the Custom UI of a component. JSAPI commands have no effect in the player, in test movie mode, or outside the authoring environment.

The Flash JSAPI provides several objects, methods, and properties to duplicate or emulate commands that a user can enter in the authoring environment. Using the JSAPI, you can write scripts that extend Flash in several ways: adding commands to menus, manipulating objects on the Stage, repeating sequences of commands, and so on.

In general, a user runs a JSAPI script by selecting Commands > Run Command. However, you can use this function in an ActionScript script to call a JSAPI command directly. If you use `MMExecute()` in a script on Frame 1 of your file, the command executes when the SWF file is loaded.

For more information on the JSAPI, see www.macromedia.com/go/jsapi_info_en.

Example

The following command will output the number of items in the library of the current document to the trace window. You must run this example as a Flash panel because Flash files can't call `MMExecute` if they are run in either test movie or the browser.

- Place the following code into frame 1 of the main Timeline of an empty Flash document:
```
var numLibItems = MMExecute("fl.getDocumentDOM().library.items.length");
var message = numLibItems + " items in library";
MMExecute('fl.trace("' + message + '");');
```
- Save the FLA file in the WindowSWF directory that is located in your Configuration directory, and then select File > Publish (or save it elsewhere and either publish the SWF file directly to that directory, or move the SWF file to that directory).
- Quit and restart the application (you need to do this step the first time you add your file to the WindowSWF directory).

Now you can select your file from the bottom of the Window > Other Panels menu.

The ActionScript trace function does not work from a Flash panel; this example uses the JavaScript `fl.trace` version to get the output. It might be easier to copy the results of `MMExecute` to a text field that is part of your Flash Panel file.

Mouse class

Availability

Flash Player 5.

Description

The Mouse class is a top-level class whose properties and methods you can access without using a constructor. You can use the methods of the Mouse class to hide and show the mouse pointer (cursor) in the SWF file. The mouse pointer is visible by default, but you can hide it and implement a custom pointer that you create using a movie clip (see "Creating a custom mouse pointer" in *Using Flash*).

Method summary for the Mouse class

Method	Description
`Mouse.addListener()`	Registers an object to receive `onMouseDown`, `onMouseMove`, `onMouseWheel`, and `onMouseUp` notification.
`Mouse.hide()`	Hides the mouse pointer in the SWF file.
`Mouse.removeListener()`	Removes an object that was registered with `addListener()`.
`Mouse.show()`	Displays the mouse pointer in the SWF file.

Listener summary for the Mouse class

Method	Description
`Mouse.onMouseDown`	Notified when the mouse button is pressed down.
`Mouse.onMouseMove`	Notified when the mouse is moved.
`Mouse.onMouseUp`	Notified when the mouse button is released.
`Mouse.onMouseWheel`	Notified when the user rolls the mouse wheel.

Mouse.addListener()

Availability

Flash Player 6.

Usage

`Mouse.addListener (newListener:Object)`

Parameters

newListener An object.

Returns

Nothing.

Description

Method; registers an object to receive notifications of the `onMouseDown`, `onMouseMove`, `onMouseUp`, and `onMouseWheel` listeners. (The `onMouseWheel` listener is supported only in Windows.)

The *newListener* parameter should contain an object that has a defined method for at least one of the listeners.

When the mouse is pressed, moved, released, or used to scroll, regardless of the input focus, all listening objects that are registered with this method have their `onMouseDown`, `onMouseMove`, `onMouseUp`, or `onMouseWheel` method invoked. Multiple objects can listen for mouse notifications. If the listener *newListener* is already registered, no change occurs.

See also

`Mouse.onMouseDown`, `Mouse.onMouseMove`, `Mouse.onMouseUp`, `Mouse.onMouseWheel`

Example

This example is excerpted from the *animation.fla* file in the HelpExamples Folder.

```
// Create a mouse listener object
var mouseListener:Object = new Object();

/* Every time the mouse cursor moves within the SWF file,
  update the position of the crosshair movie clip
  instance on the Stage.*/
mouseListener.onMouseMove = function() {
  crosshair_mc._x = _xmouse;
  crosshair_mc._y = _ymouse;
};

/* When you click the mouse, check to see if the cursor is within the
  boundaries of the Stage. If so, increment the number of shots. */
mouseListener.onMouseDown = function() {
  if (bg_mc.hitTest(_xmouse, _ymouse, false)) {
    _global.shots++;
  }
};
Mouse.addListener(mouseListener);
```

To view the entire script, see the *animation.fla* file in the HelpExamples Folder. The following list shows typical paths to the HelpExamples Folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/

Mouse.hide()

Availability

Flash Player 5.

Usage

```
Mouse.hide() : Boolean
```

Parameters

None.

Returns

A Boolean value: `true` if the pointer is visible; `false` otherwise.

Description

Method; hides the pointer in a SWF file. The pointer is visible by default.

Example:

The following code hides the standard mouse pointer, and sets the *x* and *y* positions of the `cursor_mc` movie clip instance to the *x* and *y* cursor position. Create a movie clip and set its Linkage identifier to `cursor_id`. Add the following ActionScript to Frame 1 of the Timeline:

```
// to use this script you need a symbol
// in your library with a Linkage Identifier of "pointer_id".
this.attachMovie("pointer_id", "pointer_mc", this.getNextHighestDepth());
Mouse.hide();
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
  pointer_mc._x = _xmouse;
  pointer_mc._y = _ymouse;
  updateAfterEvent();
};
Mouse.addListener(mouseListener);
```

See also

`Mouse.show()`, `MovieClip._xmouse`, `MovieClip._ymouse`

Mouse.onMouseDown

Availability

Flash Player 6.

Usage

```
someListener.onMouseDown
```

Parameters

None.

Description

Listener; notified when the mouse is pressed. To use the `onMouseDown` listener, you must create a listener object. You can then define a function for `onMouseDown` and use `addListener()` to register the listener with the Mouse object, as shown in the following code:

```
var someListener:Object = new Object();
someListener.onMouseDown = function () { ... };
Mouse.addListener(someListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

Example

The following example uses the Drawing API to draw a rectangle whenever the user clicks, drags and releases the mouse at runtime.

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
  this.isDrawing = true;
  this.orig_x = _xmouse;
  this.orig_y = _ymouse;
  this.target_mc = canvas_mc.createEmptyMovieClip("",
  canvas_mc.getNextHighestDepth());
};
mouseListener.onMouseMove = function() {
  if (this.isDrawing) {
    this.target_mc.clear();
    this.target_mc.lineStyle(1, 0xFF0000, 100);
    this.target_mc.moveTo(this.orig_x, this.orig_y);
    this.target_mc.lineTo(_xmouse, this.orig_y);
    this.target_mc.lineTo(_xmouse, _ymouse);
    this.target_mc.lineTo(this.orig_x, _ymouse);
    this.target_mc.lineTo(this.orig_x, this.orig_y);
  }
  updateAfterEvent();
};
mouseListener.onMouseUp = function() {
  this.isDrawing = false;
};
Mouse.addListener(mouseListener);
```

See also

`Mouse.addListener()`

Mouse.onMouseMove

Availability

Flash Player 6.

Usage

someListener.onMouseMove

Parameters

None.

Returns

Nothing.

Description

Listener; notified when the mouse moves. To use the `onMouseMove` listener, you must create a listener object. You can then define a function for `onMouseMove` and use `addListener()` to register the listener with the Mouse object, as shown in the following code:

```
var someListener:Object = new Object();
someListener.onMouseMove = function () { ... };
Mouse.addListener(someListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

Example

The following example uses the mouse pointer as a tool to draw lines using `onMouseMove` and the Drawing API. The user draws a line when they drag the mouse pointer.

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
  this.isDrawing = true;
  canvas_mc.lineStyle(2, 0xFF0000, 100);
  canvas_mc.moveTo(_xmouse, _ymouse);
};
mouseListener.onMouseMove = function() {
  if (this.isDrawing) {
    canvas_mc.lineTo(_xmouse, _ymouse);
  }
  updateAfterEvent();
};
mouseListener.onMouseUp = function() {
  this.isDrawing = false;
};
Mouse.addListener(mouseListener);
```

The following example hides the standard mouse pointer, and sets the *x* and *y* positions of the `pointer_mc` movie clip instance to the *x* and *y* pointer position. Create a movie clip and set its Linkage identifier to `pointer_id`. Add the following ActionScript to Frame 1 of the Timeline:

```
// to use this script you need a symbol
// in your library with a Linkage Identifier of "pointer_id".
this.attachMovie("pointer_id", "pointer_mc", this.getNextHighestDepth());
Mouse.hide();
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
  pointer_mc._x = _xmouse;
  pointer_mc._y = _ymouse;
  updateAfterEvent();
};
Mouse.addListener(mouseListener);
```

See also

`Mouse.addListener()`

Mouse.onMouseUp

Availability

Flash Player 6.

Usage

someListener.onMouseUp

Parameters

None.

Returns

Nothing.

Description

Listener; notified when the mouse is released. To use the onMouseUp listener, you must create a listener object. You can then define a function for onMouseUp and use addListener() to register the listener with the Mouse object, as shown in the following code:

```
var someListener:Object = new Object();
someListener.onMouseUp = function () { ... };
Mouse.addListener(someListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

Example

The following example uses the mouse pointer as a tool to draw lines using onMouseMove and the Drawing API. The user draws a line when they drag the mouse pointer. The user stops drawing the line when they release the mouse button.

```
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
  this.isDrawing = true;
  canvas_mc.lineStyle(2, 0xFF0000, 100);
  canvas_mc.moveTo(_xmouse, _ymouse);
};
mouseListener.onMouseMove = function() {
  if (this.isDrawing) {
    canvas_mc.lineTo(_xmouse, _ymouse);
  }
  updateAfterEvent();
};
mouseListener.onMouseUp = function() {
  this.isDrawing = false;
};
Mouse.addListener(mouseListener);
```

See also

Mouse.addListener()

Mouse.onMouseWheel

Availability

Flash Player 6 (Windows only).

Usage

```
someListener.onMouseWheel = function ( [ delta [, scrollTarget ] ] ) {
  // your statements here
}
```

Parameters

delta An optional number indicating how many lines should be scrolled for each notch the user rolls the mouse wheel. A positive *delta* value indicates an upward scroll; a negative value indicates a downward scroll. Typical values are from 1 to 3; faster scrolling can produce larger values.

scrollTarget An optional parameter that indicates the topmost movie clip instance under the mouse pointer when the mouse wheel is rolled. If you want to specify a value for *scrollTarget* but don't want to specify a value for *delta*, pass `null` for *delta*.

Returns

Nothing.

Description

Listener; notified when the user rolls the mouse wheel. To use the `onMouseWheel` listener, you must create a listener object. You can then define a function for `onMouseWheel` and use `addListener()` to register the listener with the Mouse object.

Note: Mouse wheel event listeners are available only in Windows versions of Flash Player.

Example

The following example shows how to create a listener object that responds to mouse wheel events. In this example, the *x* coordinate of a movie clip object named `clip_mc` changes each time the user rotates the mouse wheel:

```
var mouseListener:Object = new Object();
mouseListener.onMouseWheel = function(delta) {
  clip_mc._x += delta;
}
Mouse.addListener(mouseListener);
```

The following example draws a line that rotates when you rotate the mouse wheel. Click the SWF file at runtime and then rotate your mouse wheel to see the movie clip in action.

```
this.createEmptyMovieClip("line_mc", this.getNextHighestDepth());
line_mc.lineStyle(2, 0xFF0000, 100);
line_mc.moveTo(0, 100);
line_mc.lineTo(0, 0);
line_mc._x = 200;
line_mc._y = 200;

var mouseListener:Object = new Object();
```

```
mouseListener.onMouseWheel = function(delta:Number) {
  line_mc._rotation += delta;
};
mouseListener.onMouseDown = function() {
  trace("Down");
};
Mouse.addListener(mouseListener);
```

See also

Mouse.addListener(), TextField.mouseWheelEnabled

Mouse.removeListener()

Availability

Flash Player 6.

Usage

`Mouse.removeListener (`*`listener:Object`*`) :`*` Boolean`*

Parameters

`listener` An object.

Returns

If the *`listener`* object is successfully removed, the method returns `true`; if the *`listener`* is not successfully removed (for example, if the *`listener`* was not on the Mouse object's listener list), the method returns `false`.

Description

Method; removes an object that was previously registered with `addListener()`.

Example

The following example attaches three buttons to the Stage, and lets the user draw lines in the SWF file at runtime, using the mouse pointer. One button clears all of the lines from the SWF file. The second button removes the mouse listener so the user cannot draw lines. The third button adds the mouse listener after it is removed, so the user can draw lines again. Add the following ActionScript to Frame 1 of the Timeline:

```
/* Add an instance of the Button
   component to be placed in the Library.
   This example attaches and positions three Button
   instances on the Stage. */
this.createClassObject(mx.controls.Button, "clear_button",
  this.getNextHighestDepth(), {_x:10, _y:10, label:'clear'});
this.createClassObject(mx.controls.Button, "stopDrawing_button",
  this.getNextHighestDepth(), {_x:120, _y:10, label:'stop drawing'});
this.createClassObject(mx.controls.Button, "startDrawing_button",
  this.getNextHighestDepth(), {_x:230, _y:10, label:'start drawing'});
startDrawing_button.enabled = false;
//
this.createEmptyMovieClip("canvas_mc", this.getNextHighestDepth());
var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
  this.isDrawing = true;
  canvas_mc.lineStyle(2, 0xFF0000, 100);
  canvas_mc.moveTo(_xmouse, _ymouse);
};
mouseListener.onMouseMove = function() {
  if (this.isDrawing) {
    canvas_mc.lineTo(_xmouse, _ymouse);
  }
  updateAfterEvent();
};
```

```
mouseListener.onMouseUp = function() {
  this.isDrawing = false;
};
Mouse.addListener(mouseListener);
var clearListener:Object = new Object();
clearListener.click = function() {
  canvas_mc.clear();
};
clear_button.addEventListener("click", clearListener);
//
var stopDrawingListener:Object = new Object();
stopDrawingListener.click = function(evt:Object) {
  Mouse.removeListener(mouseListener);
  evt.target.enabled = false;
  startDrawing_button.enabled = true;
};
stopDrawing_button.addEventListener("click", stopDrawingListener);
var startDrawingListener:Object = new Object();
startDrawingListener.click = function(evt:Object) {
  Mouse.addListener(mouseListener);
  evt.target.enabled = false;
  stopDrawing_button.enabled = true;
};
startDrawing_button.addEventListener("click", startDrawingListener);
```

Mouse.show()

Availability

Flash Player 5.

Usage

```
Mouse.show() : Number
```

Parameters

None.

Returns

An integer; either `0` or `1`. If the mouse pointer was hidden before the call to `Mouse.show()`, then the return value is `0`. If the mouse pointer was visible before the call to `Mouse.show()`, then the return value is `1`.

Description

Method; displays the mouse pointer in a SWF file. The pointer is visible by default.

Example

The following example attaches a custom cursor from the library when it rolls over a movie clip called `my_mc`. Give a movie clip in the Library a Linkage identifier of `cursor_help_id`, and add the following ActionScript to Frame 1 of the Timeline:

```
my_mc.onRollOver = function() {
  Mouse.hide();
  this.attachMovie("cursor_help_id", "cursor_mc", this.getNextHighestDepth(),
  {_x:this._xmouse, _y:this._ymouse});
};
my_mc.onMouseMove = function() {
  this.cursor_mc._x = this._xmouse;
  this.cursor_mc._y = this._ymouse;
};
my_mc.onRollOut = function() {
  Mouse.show();
  this.cursor_mc.removeMovieClip();
};
```

See also

`Mouse.hide()`, `MovieClip._xmouse`, `MovieClip._ymouse`

MovieClip class

Availability

Flash Player 3.

Description

The methods for the MovieClip class provide the same functionality as actions that target movie clips. There are also additional methods that do not have equivalent actions in the Actions toolbox in the Actions panel.

You do not need to use a constructor method to call the methods of the MovieClip class; instead, you reference movie clip instances by name, using the following syntax:

```
my_mc.play();
my_mc.gotoAndPlay(3);
```

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Method summary for the MovieClip class

Method	Description
`MovieClip.attachAudio()`	Captures and plays local audio from the microphone hardware.
`MovieClip.attachMovie()`	Attaches a SWF file or movie clip in the library.
`MovieClip.createEmptyMovieClip()`	Creates an empty movie clip.
`MovieClip.createTextField()`	Creates an empty text field.
`MovieClip.duplicateMovieClip()`	Duplicates the specified movie clip.
`MovieClip.getBounds()`	Returns the minimum and maximum *x* and *y* coordinates of a SWF file in a specified coordinate space.
`MovieClip.getBytesLoaded()`	Returns the number of bytes loaded for the specified movie clip.
`MovieClip.getBytesTotal()`	Returns the size of the movie clip, in bytes.
`MovieClip.getDepth()`	Returns the depth of a movie clip.
`MovieClip.getInstanceAtDepth()`	Returns the movie clip instance from a particular depth if it exists.
`MovieClip.getNextHighestDepth()`	Returns the next available depth that can be passed to other methods. This ensures that Flash renders the movie clip in front of all other objects in the current movie clip.
`MovieClip.getSWFVersion()`	Returns an integer that indicates the Flash Player version for which the movie clip was published.
`MovieClip.getTextSnapshot()`	Returns a TextSnapshot object that contains the text in the static text fields in the specified movie clip.
`MovieClip.getURL()`	Retrieves a document from a URL.

Method	Description
`MovieClip.globalToLocal()`	Converts Stage coordinates to the local coordinates of the specified movie clip.
`MovieClip.gotoAndPlay()`	Sends the playhead to a specific frame in the movie clip and plays the movie clip.
`MovieClip.gotoAndStop()`	Sends the playhead to a specific frame in the movie clip and stops the movie clip.
`MovieClip.hitTest()`	Returns `true` if bounding box of the specified movie clip intersects the bounding box of the target movie clip.
`MovieClip.loadMovie()`	Loads the specified SWF or JPEG file into the movie clip.
`MovieClip.loadVariables()`	Loads variables from a URL or other location into the movie clip.
`MovieClip.localToGlobal()`	Converts local coordinates of the movie clip to the global Stage.
`MovieClip.nextFrame()`	Sends the playhead to the next frame of the movie clip.
`MovieClip.play()`	Plays the specified movie clip.
`MovieClip.prevFrame()`	Sends the playhead to the previous frame of the movie clip.
`MovieClip.removeMovieClip()`	Removes the movie clip from the Timeline if it was created with `duplicateMovieClip()`, `MovieClip.duplicateMovieClip()`, or `MovieClip.attachMovie()`.
`MovieClip.setMask()`	Sets a movie clip as the mask for the specified movie clip.
`MovieClip.startDrag()`	Specifies a movie clip as draggable and begins dragging the movie clip.
`MovieClip.stop()`	Stops the currently playing movie clip.
`MovieClip.stopDrag()`	Stops the dragging of any movie clip that is being dragged.
`MovieClip.swapDepths()`	Swaps the depth level of two movie clips.
`MovieClip.unloadMovie()`	Removes a SWF file that was loaded with `MovieClip.loadMovie()`.

Drawing method summary for the MovieClip class

Method	Description
`MovieClip.beginFill()`	Begins drawing a fill for the specified movie clip.
`MovieClip.beginGradientFill()`	Begins drawing a gradient fill for the specified movie clip.
`MovieClip.clear()`	Removes all the drawing commands associated with a movie clip instance.
`MovieClip.curveTo()`	Draws a curve using the current line style.
`MovieClip.endFill()`	Ends the fill specified by `MovieClip.beginFill()` or `MovieClip.beginGradientFill()`.

Method	Description
`MovieClip.lineStyle()`	Defines the stroke of lines created with the `MovieClip.lineTo()` and MovieClip.curveTo() methods.
`MovieClip.lineTo()`	Draws a line using the current line style.
`MovieClip.moveTo()`	Moves the current drawing position to specified coordinates.

Property summary for the MovieClip class

Property	Description
`MovieClip._alpha`	The transparency value of a movie clip instance.
`MovieClip._currentframe`	Read-only; the frame number in which the playhead is currently located.
`MovieClip._droptarget`	Read-only; the absolute path in slash syntax notation of the movie clip instance on which a draggable movie clip was dropped.
`MovieClip.enabled`	A Boolean value that indicates whether a movie clip is enabled.
`MovieClip.focusEnabled`	A Boolean value that enables a movie clip to receive focus.
`MovieClip._focusrect`	A Boolean value that indicates whether a focused movie clip has a yellow rectangle around it.
`MovieClip._framesloaded`	Read-only; the number of frames that have been loaded from a streaming SWF file.
`MovieClip._height`	The height of a movie clip instance, in pixels.
`MovieClip.hitArea`	A reference to a movie clip that serves as the hit area for another movie clip.
`MovieClip._lockroot`	The specification of what _root refers to when a SWF file is loaded into a movie clip.
`MovieClip.menu`	An object that associates a ContextMenu object with a movie clip.
`MovieClip._name`	The instance name of a movie clip instance.
`MovieClip._parent`	A reference to the movie clip that encloses the movie clip.
`MovieClip._quality`	A string that sets the rendering quality of a SWF file.
`MovieClip._rotation`	The degree of rotation of a movie clip instance.
`MovieClip._soundbuftime`	The number of seconds before a sound starts to stream.
`MovieClip.tabChildren`	A Boolean value that indicates whether the children of a movie clip are included in automatic tab ordering.
`MovieClip.tabEnabled`	A Boolean value that indicates whether a movie clip is included in tab ordering.
`MovieClip.tabIndex`	A number that indicates the tab order of an object.

Property	Description
`MovieClip._target`	Read-only; the target path of a movie clip instance.
`MovieClip._totalframes`	Read-only; the total number of frames in a movie clip instance.
`MovieClip.trackAsMenu`	A Boolean value that indicates whether other movie clips can receive mouse release events.
`MovieClip._url`	Read-only; the URL of the SWF file from which a movie clip was downloaded.
`MovieClip.useHandCursor`	A Boolean value that determines whether the hand is displayed when the mouse rolls over a movie clip.
`MovieClip._visible`	A Boolean value that determines whether a movie clip instance is hidden or visible.
`MovieClip._width`	The width of a movie clip instance, in pixels.
`MovieClip._x`	The *x* coordinate of a movie clip instance
`MovieClip._xmouse`	Read-only; the *x* coordinate of the mouse pointer within a movie clip instance.
`MovieClip._xscale`	The value specifying the percentage that the movie clip is scaled horizontally.
`MovieClip._y`	The *y* coordinate of a movie clip instance.
`MovieClip._ymouse`	Read-only; the *y* coordinate of the mouse pointer within a movie clip instance.
`MovieClip._yscale`	The value specifying the percentage for vertically scaling a movie clip.

Event handler summary for the MovieClip class

Event handler	Description
`MovieClip.onData`	Invoked when all the data is loaded into a movie clip.
`MovieClip.onDragOut`	Invoked when the mouse button is pressed inside the movie clip area and then rolled outside the movie clip area.
`MovieClip.onDragOver`	Invoked when the mouse pointer is dragged over the movie clip.
`MovieClip.onEnterFrame`	Invoked continually at the frame rate of the SWF file. The actions associated with the `enterFrame` event are processed before any frame actions that are attached to the affected frames.
`MovieClip.onKeyDown`	Invoked when a key is pressed. Use the `Key.getCode()` and `Key.getAscii()` methods to retrieve information about the last key pressed.
`MovieClip.onKeyUp`	Invoked when a key is released.
`MovieClip.onKillFocus`	Invoked when focus is removed from a movie clip.

Event handler	Description
`MovieClip.onLoad`	Invoked when the movie clip is instantiated and appears in the Timeline.
`MovieClip.onMouseDown`	Invoked when the left mouse button is pressed.
`MovieClip.onMouseMove`	Invoked every time the mouse is moved.
`MovieClip.onMouseUp`	Invoked when the left mouse button is released.
`MovieClip.onPress`	Invoked when the mouse is pressed while the pointer is over a movie clip.
`MovieClip.onRelease`	Invoked when the mouse is released while the pointer is over a movie clip.
`MovieClip.onReleaseOutside`	Invoked when the mouse is clicked over a movie clip and released while the pointer is outside the movie clip's area.
`MovieClip.onRollOut`	Invoked when the pointer rolls outside of a movie clip area.
`MovieClip.onRollOver`	Invoked when the mouse pointer rolls over a movie clip.
`MovieClip.onSetFocus`	Invoked when a movie clip has input focus and a key is released.
`MovieClip.onUnload`	Invoked in the first frame after the movie clip is removed from the Timeline. The actions associated with the Unload movie clip event are processed before any actions are attached to the affected frame.

MovieClip._alpha

Availability

Flash Player 4.

Usage

my_mc._alpha:*Number*

Description

Property; the alpha transparency value of the movie clip specified by *my_mc*. Valid values are 0 (fully transparent) to 100 (fully opaque). The default value is 100. Objects in a movie clip with _alpha set to 0 are active, even though they are invisible. For example, you can still click a button in a movie clip whose _alpha property is set to 0. To disable the button completely, you can set the movie clip's _visible property to false.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following code sets the _alpha property of a dynamically created movie clip named holder_mc to 50% when the mouse rolls over the movie clip. Add the following ActionScript to your FLA or AS file:

```
this.createEmptyMovieClip("holder_mc", this.getNextHighestDepth());
holder_mc.createEmptyMovieClip("image_mc", holder_mc.getNextHighestDepth());
// replace with your own image or use the following
holder_mc.image_mc.loadMovie("http://www.macromedia.com/devnet/mx/blueprint/
  articles/nielsen/spotlight_jnielsen.jpg");
holder_mc.onRollOver = function() {
  this._alpha = 50;
};
holder_mc.onRollOut = function() {
  this._alpha = 100;
};
```

See also

Button._alpha, TextField._alpha, MovieClip._visible

MovieClip.attachAudio()

Availability

Flash Player 6; the ability to attach audio from Flash Video (FLV) files was added in Flash Player 7.

Usage

my_mc.attachAudio(*source:Object*) : *Void*

Parameters

source The object containing the audio to play. Valid values are a Microphone object, a NetStream object that is playing an FLV file, and `false` (stops playing the audio).

Returns

Nothing.

Description

Method; specifies the audio source to be played. To stop playing the audio source, pass `false` for *source*.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following code creates a new NetStream connection. Add a new Video symbol by opening the Library panel and selecting New Video from the Library options menu. Give it the instance name `my_video`. Dynamically load the FLV video at runtime. Use the `attachAudio()` method to attach the audio from the FLV file to a movie clip on the Stage. Then you can control the audio in the movie clip using the Sound class and two buttons called `volUp_btn` and `volDown_btn`:

```
var my_nc:NetConnection = new NetConnection();
my_nc.connect(null);
var my_ns:NetStream = new NetStream(my_nc);
my_video.attachVideo(my_ns);
my_ns.play("yourVideo.flv");
this.createEmptyMovieClip("flv_mc", this.getNextHighestDepth());
flv_mc.attachAudio(my_ns);
var audio_sound:Sound = new Sound(flv_mc);
// add volume buttons
volUp_btn.onRelease = function() {
  if (audio_sound.getVolume()<100) {
    audio_sound.setVolume(audio_sound.getVolume()+10);
    updateVolume();
  }
};
volDown_btn.onRelease = function() {
  if (audio_sound.getVolume()>0) {
    audio_sound.setVolume(audio_sound.getVolume()-10);
    updateVolume();
  }
};
```

```
// updates the volume
this.createTextField("volume_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
updateVolume();
function updateVolume() {
  volume_txt.text = "Volume: "+audio_sound.getVolume();
}
```

The following example specifies a microphone as the audio source for a dynamically created movie clip instance called `audio_mc`:

```
var active_mic:Microphone = Microphone.get();
this.createEmptyMovieClip("audio_mc", this.getNextHighestDepth());
audio_mc.attachAudio(active_mic);
```

See also

Microphone class, `NetStream.play()`, Sound class, `Video.attachVideo()`

MovieClip.attachMovie()

Availability

Flash Player 5.

Usage

```
my_mc.attachMovie(idName:String, newName:String, depth:Number [,
  initObject:Object]) : MovieClip
```

Parameters

idName The linkage name of the movie clip symbol in the library to attach to a movie clip on the Stage. This is the name entered in the Identifier field in the Linkage Properties dialog box.

newname A unique instance name for the movie clip being attached to the movie clip.

depth An integer specifying the depth level where the SWF file is placed.

initObject (Supported for Flash Player 6 and later) An object containing properties with which to populate the newly attached movie clip. This parameter allows dynamically created movie clips to receive clip parameters. If *initObject* is not an object, it is ignored. All properties of *initObject* are copied into the new instance. The properties specified with *initObject* are available to the constructor function. This parameter is optional.

Returns

A reference to the newly created instance.

Description

Method; takes a symbol from the library and attaches it to the SWF file on the Stage specified by *my_mc*. Use `MovieClip.removeMovieClip()` or `MovieClip.unloadMovie()` to remove a SWF file attached with `attachMovie()`.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example attaches the symbol with the linkage identifier "circle" to the movie clip instance, which is on the Stage in the SWF file:

```
this.attachMovie("circle", "circle1_mc", this.getNextHighestDepth());
this.attachMovie("circle", "circle2_mc", this.getNextHighestDepth(), {_x:100,
  _y:100});
```

See also

`MovieClip.removeMovieClip()`, `MovieClip.unloadMovie()`, `removeMovieClip()`

MovieClip.beginFill()

Availability

Flash Player 6.

Usage

```
my_mc.beginFill([rgb:Number[, alpha:Number]]) : Void
```

Parameter

`rgb` A hex color value (for example, red is 0xFF0000, blue is 0x0000FF, and so on). If this value is not provided or is undefined, a fill is not created.

`alpha` An integer between 0–100 that specifies the alpha value of the fill. If this value is not provided, 100 (solid) is used. If the value is less than 0, Flash uses 0. If the value is greater than 100, Flash uses 100.

Returns

Nothing.

Description

Method; indicates the beginning of a new drawing path. If an open path exists (that is, if the current drawing position does not equal the previous position specified in a `MovieClip.moveTo()` method) and it has a fill associated with it, that path is closed with a line and then filled. This is similar to what happens when `MovieClip.endFill()` is called.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example creates a square with red fill on the Stage.

```
this.createEmptyMovieClip("square_mc", this.getNextHighestDepth());
square_mc.beginFill(0xFF0000);
square_mc.moveTo(10, 10);
square_mc.lineTo(100, 10);
square_mc.lineTo(100, 100);
square_mc.lineTo(10, 100);
square_mc.lineTo(10, 10);
square_mc.endFill();
```

An example is also in the *drawingapi.fla* file in the HelpExamples folder. The following list gives typical paths to this folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/

See also

`MovieClip.beginGradientFill()`, `MovieClip.endFill()`

MovieClip.beginGradientFill()

Availability

Flash Player 6.

Usage

```
my_mc.beginGradientFill(fillType:String, colors:Array, alphas:Array,
ratios:Array, matrix:Object) : Void
```

Parameter

fillType Either the string "linear" or the string "radial".

colors An array of RGB hex color values to be used in the gradient (for example, red is 0xFF0000, blue is 0x0000FF, and so on).

alphas An array of alpha values for the corresponding colors in the *colors* array; valid values are 0–100. If the value is less than 0, Flash uses 0. If the value is greater than 100, Flash uses 100.

ratios An array of color distribution ratios; valid values are 0–255. This value defines the percentage of the width where the color is sampled at 100 percent.

matrix A transformation matrix that is an object with either of the following two sets of properties.

- *a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *i*, which can be used to describe a 3 x 3 matrix of the following form:

  ```
  a  b  c
  d  e  f
  g  h  i
  ```

  The following example uses a `beginGradientFill()` method with a *matrix* parameter that is an object with these properties.

  ```
  this.createEmptyMovieClip("gradient_mc", 1);
  with (gradient_mc) {
    colors = [0xFF0000, 0x0000FF];
    alphas = [100, 100];
    ratios = [0, 0xFF];
    matrix = {a:200, b:0, c:0, d:0, e:200, f:0, g:200, h:200, i:1};
    beginGradientFill("linear", colors, alphas, ratios, matrix);
    moveTo(100, 100);
    lineTo(100, 300);
    lineTo(300, 300);
    lineTo(300, 100);
    lineTo(100, 100);
    endFill();
  }
  ```

If a *matrixType* property does not exist then the remaining parameters are all required; the function fails if any of them are missing. This matrix scales, translates, rotates, and skews the unit gradient, which is defined at (-1,-1) and (1,1).

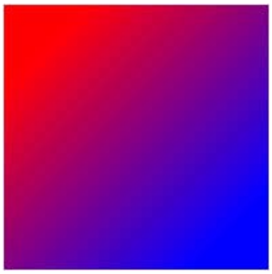
- *matrixType*, *x*, *y*, *w*, *h*, *r*.

  The properties indicate the following: *matrixType* is the string `"box"`, *x* is the horizontal position relative to the registration point of the parent clip for the upper left corner of the gradient, *y* is the vertical position relative to the registration point of the parent clip for the upper left corner of the gradient, *w* is the width of the gradient, *h* is the height of the gradient, and *r* is the rotation in radians of the gradient.

  The following example uses a `beginGradientFill()` method with a *matrix* parameter that is an object with these properties.

```
this.createEmptyMovieClip("gradient_mc", 1);
with (gradient_mc) {
  colors = [0xFF0000, 0x0000FF];
  alphas = [100, 100];
  ratios = [0, 0xFF];
  matrix = {matrixType:"box", x:100, y:100, w:200, h:200, r:(45/
  180)*Math.PI};
  beginGradientFill("linear", colors, alphas, ratios, matrix);
  moveTo(100, 100);
  lineTo(100, 300);
  lineTo(300, 300);
  lineTo(300, 100);
  lineTo(100, 100);
  endFill();
}
```

If a *matrixType* property exists then it must equal `"box"` and the remaining parameters are all required. The function fails if any of these conditions are not met.



Returns

Nothing.

Description

Method; indicates the beginning of a new drawing path. If the first parameter is `undefined`, or if no parameters are passed, the path has no fill. If an open path exists (that is if the current drawing position does not equal the previous position specified in a `MovieClip.moveTo()` method), and it has a fill associated with it, that path is closed with a line and then filled. This is similar to what happens when you call `MovieClip.endFill()`.

This method fails if any of the following conditions exist:

- The number of items in the *`colors`*, *`alphas`*, and *`ratios`* parameters are not equal.
- The *`fillType`* parameter is not `"linear"` or `"radial"`.
- Any of the fields in the object for the *`matrix`* parameter are missing or invalid.

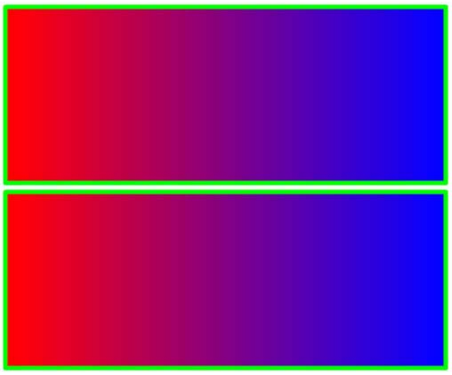
You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following code uses both methods to draw two stacked rectangles with a red-blue gradient fill and a 5-pixel solid lime green stroke:

```
this.createEmptyMovieClip("gradient_mc", 1);
with (gradient_mc) {
  colors = [0xFF0000, 0x0000FF];
  alphas = [100, 100];
  ratios = [0, 0xFF];
  lineStyle(5, 0x00ff00);
  matrix = {a:500, b:0, c:0, d:0, e:200, f:0, g:350, h:200, i:1};
  beginGradientFill("linear", colors, alphas, ratios, matrix);
  moveTo(100, 100);
  lineTo(100, 300);
  lineTo(600, 300);
  lineTo(600, 100);
  lineTo(100, 100);
  endFill();
  matrix = {matrixType:"box", x:100, y:310, w:500, h:200, r:(0/180)*Math.PI};
  beginGradientFill("linear", colors, alphas, ratios, matrix);
  moveTo(100, 310);
  lineTo(100, 510);
  lineTo(600, 510);
  lineTo(600, 310);
  lineTo(100, 310);
```

```
    endFill();
}
```



See also

`MovieClip.beginFill()`, `MovieClip.endFill()`, `MovieClip.lineStyle()`, `MovieClip.lineTo()`, `MovieClip.moveTo()`

MovieClip.clear()

Availability

Flash Player 6.

Usage

```
my_mc.clear() : Void
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all the graphics created during runtime using the movie clip draw methods, including line styles specified with `MovieClip.lineStyle()`. Shapes and lines that are manually drawn during authoring time (with the Flash drawing tools) are unaffected.

Example

The following example draws a box on the Stage. When the user clicks the box graphic, it removes the graphic from the Stage.

```
this.createEmptyMovieClip("box_mc", this.getNextHighestDepth());
box_mc.onRelease = function() {
  this.clear();
};
drawBox(box_mc, 10, 10, 320, 240);
function drawBox(mc:MovieClip, x:Number, y:Number, w:Number, h:Number):Void {
  mc.lineStyle(0);
  mc.beginFill(0xEEEEEE);
  mc.moveTo(x, y);
  mc.lineTo(x+w, y);
  mc.lineTo(x+w, y+h);
  mc.lineTo(x, y+h);
  mc.lineTo(x, y);
  mc.endFill();
}
```

An example is also in the *drawingapi.fla* file in the HelpExamples folder. The following list gives typical paths to this folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/

See also

`MovieClip.lineStyle()`

MovieClip.createEmptyMovieClip()

Availability

Flash Player 6.

Usage

```
my_mc.createEmptyMovieClip(instanceName:String, depth:Number) : MovieClip
```

Parameters

`instanceName` A string that identifies the instance name of the new movie clip.

`depth` An integer that specifies the depth of the new movie clip.

Returns

A reference to the newly created movie clip.

Description

Method; creates an empty movie clip as a child of an existing movie clip. This method behaves similarly to the `attachMovie()` method, but you don't need to provide an external linkage identifier for the new movie clip. The registration point for a newly created empty movie clip is the upper left corner. This method fails if any of the parameters are missing.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following ActionScript creates a new movie clip at runtime and loads a JPEG image into the movie clip.

```
this.createEmptyMovieClip("logo_mc", this.getNextHighestDepth());
logo_mc.loadMovie("http://www.macromedia.com/images/shared/product_boxes/
  80x92/studio_flashpro.jpg");
```

See also

`MovieClip.attachMovie()`

MovieClip.createTextField()

Availability

Flash Player 6.

Usage

```
my_mc.createTextField(instanceName:String, depth:Number, x:Number, y:Number,
width:Number, height:Number) : Void
```

Parameters

`instanceName` A string that identifies the instance name of the new text field.

`depth` A positive integer that specifies the depth of the new text field.

`x` An integer that specifies the *x* coordinate of the new text field.

`y` An integer that specifies the *y* coordinate of the new text field.

`width` A positive integer that specifies the width of the new text field.

`height` A positive integer that specifies the height of the new text field.

Returns

Nothing.

Description

Method; creates a new, empty text field as a child of the movie clip specified by `my_mc`. You can use `createTextField()` to create text fields while a SWF file plays. The `depth` parameter determines the new text field's z-order position in the movie clip. Each position in the z-order can contain only one object. If you create a new text field on a depth that already has a text field, the new text field will replace the existing text field. To avoid overwriting existing text fields, use the `MovieClip.getInstanceAtDepth()` to determine whether a specific depth is already occupied, or `MovieClip.getNextHighestDepth()`, to determine the highest unoccupied depth. The text field is positioned at (*x*, *y*) with dimensions `width` by `height`. The `x` and `y` parameters are relative to the container movie clip; these parameters correspond to the `_x` and `_y` properties of the text field. The `width` and `height` parameters correspond to the `_width` and `_height` properties of the text field.

The default properties of a text field are as follows:

```
type = "dynamic"
border = false
background = false
password = false
multiline = false
html = false
embedFonts = false
selectable = true
wordWrap = false
mouseWheelEnabled = true
condenseWhite = false
restrict = null
```

```
variable = null
maxChars = null
styleSheet = undefined
tabInded = undefined
```

A text field created with `createTextField()` receives the following default TextFormat object:

```
font = "Times New Roman"
size = 12
color = 0x000000
bold = false
italic = false
underline = false
url = ""
target = ""
align = "left"
leftMargin = 0
rightMargin = 0
indent = 0
leading = 0
blockIndent = 0
bullet = false
display = block
tabStops = [] (empty array)
```

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example creates a text field with a width of 300, a height of 100, an *x* coordinate of 100, a *y* coordinate of 100, no border, red, and underlined text:

```
this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
var my_fmt:TextFormat = new TextFormat();
my_fmt.color = 0xFF0000;
my_fmt.underline = true;
my_txt.text = "This is my first test field object text.";
my_txt.setTextFormat(my_fmt);
```

An example is also in the *animations.fla* file in the HelpExamples folder. The following list gives typical paths to this folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/HelpExamples/

See also

TextFormat class

MovieClip._currentframe

Availability

Flash Player 4.

Usage

my_mc`._currentframe`*:Number*

Description

Read-only property; returns the number of the frame in which the playhead is located in the Timeline specified by *my_mc*.

Example

The following example uses the `_currentframe` property to direct the playhead of the movie clip `actionClip_mc` to advance five frames ahead of its current location:

```
actionClip_mc.gotoAndStop(actionClip_mc._currentframe + 5);
```

MovieClip.curveTo()

Availability

Flash Player 6.

Usage

```
my_mc.curveTo(controlX:Number, controlY:Number, anchorX:Number, anchorY:Number)
: Void
```

Parameters

`controlX` An integer that specifies the horizontal position of the control point relative to the registration point of the parent movie clip.

`controlY` An integer that specifies the vertical position of the control point relative to the registration point of the parent movie clip.

`anchorX` An integer that specifies the horizontal position of the next anchor point relative to the registration. point of the parent movie clip.

`anchorY` An integer that specifies the vertical position of the next anchor point relative to the registration point of the parent movie clip.

Returns

Nothing.

Description

Method; draws a curve using the current line style from the current drawing position to (`anchorX`, `anchorY`) using the control point specified by (`controlX`, `controlY`). The current drawing position is then set to (`anchorX`, `anchorY`). If the movie clip you are drawing in contains content created with the Flash drawing tools, calls to `curveTo()` are drawn underneath this content. If you call `curveTo()` before any calls to `moveTo()`, the current drawing position defaults to (0, 0). If any of the parameters are missing, this method fails and the current drawing position is not changed.

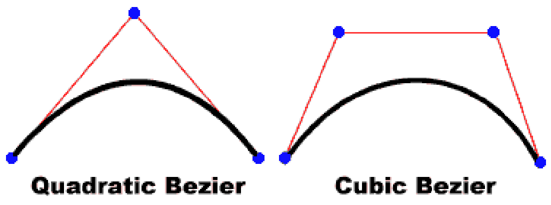
You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example draws a circle with a solid blue hairline stroke and a solid red fill:

```
this.createEmptyMovieClip("circle_mc", 1);
with (circle_mc) {
  lineStyle(0, 0x0000FF, 100);
  beginFill(0xFF0000);
  moveTo(500, 500);
  curveTo(600, 500, 600, 400);
  curveTo(600, 300, 500, 300);
  curveTo(400, 300, 400, 400);
  curveTo(400, 500, 500, 500);
  endFill();
}
```

The curve drawn in this example is a quadratic bezier curve. Quadratic bezier curves consist of two anchor points and a control point. The curve interpolates the two anchor points, and curves toward the control point.



The following ActionScript creates a circle using `MovieClip.curveTo()` and the Math class:

```
this.createEmptyMovieClip("circle2_mc", 2);
circle2_mc.lineStyle(0, 0x000000);
drawCircle(circle2_mc, 10, 10, 100);
function drawCircle(mc:MovieClip, x:Number, y:Number, r:Number):Void {
  mc.moveTo(x+r, y);
  mc.curveTo(r+x, Math.tan(Math.PI/8)*r+y, Math.sin(Math.PI/4)*r+x,
  Math.sin(Math.PI/4)*r+y);
  mc.curveTo(Math.tan(Math.PI/8)*r+x, r+y, x, r+y);
  mc.curveTo(-Math.tan(Math.PI/8)*r+x, r+y, -Math.sin(Math.PI/4)*r+x,
  Math.sin(Math.PI/4)*r+y);
  mc.curveTo(-r+x, Math.tan(Math.PI/8)*r+y, -r+x, y);
  mc.curveTo(-r+x, -Math.tan(Math.PI/8)*r+y, -Math.sin(Math.PI/4)*r+x, -
  Math.sin(Math.PI/4)*r+y);
  mc.curveTo(-Math.tan(Math.PI/8)*r+x, -r+y, x, -r+y);
  mc.curveTo(Math.tan(Math.PI/8)*r+x, -r+y, Math.sin(Math.PI/4)*r+x, -
  Math.sin(Math.PI/4)*r+y);
  mc.curveTo(r+x, -Math.tan(Math.PI/8)*r+y, r+x, y);
}
```

An example is also in the *drawingapi.fla* file in the HelpExamples folder. The following list gives typical paths to this folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/

See also

`MovieClip.beginFill()`, `MovieClip.createEmptyMovieClip()`, `MovieClip.endFill()`, `MovieClip.lineStyle()`, `MovieClip.lineTo()`, `MovieClip.moveTo()`

MovieClip._droptarget

Availability

Flash Player 4.

Usage

`my_mc._droptarget:String`

Description

Read-only property; returns the absolute path in slash syntax notation of the movie clip instance on which `my_mc` was dropped. The `_droptarget` property always returns a path that starts with a slash (/). To compare the `_droptarget` property of an instance to a reference, use the `eval()` function to convert the returned value from slash syntax to a dot syntax reference.

Note: You must perform this conversion if you are using ActionScript 2.0, which does not support slash syntax.

Example

The following example evaluates the `_droptarget` property of the `garbage_mc` movie clip instance and uses `eval()` to convert it from slash syntax to a dot syntax reference. The `garbage_mc` reference is then compared to the reference to the `trashcan_mc` movie clip instance. If the two references are equivalent, the visibility of `garbage_mc` is set to `false`. If they are not equivalent, the `garbage` instance resets to its original position.

```
origX = garbage_mc._x;
origY = garbage_mc._y;
garbage_mc.onPress = function() {
  this.startDrag();
};
garbage_mc.onRelease = function() {
  this.stopDrag();
  if (eval(this._droptarget) == trashcan_mc) {
    this._visible = false;
  } else {
    this._x = origX;
    this._y = origY;
  }
};
```

See also

`startDrag()`, `stopDrag()`

MovieClip.duplicateMovieClip()

Availability

Flash Player 5.

Usage

```
my_mc.duplicateMovieClip(newname:String, depth:Number [,initObject:Object]) :
  MovieClip
```

Parameters

newname A unique identifier for the duplicate movie clip.

depth A unique number specifying the depth at which the SWF file specified is to be placed.

initObject (Supported for Flash Player 6 and later.) An object containing properties with which to populate the duplicated movie clip. This parameter allows dynamically created movie clips to receive clip parameters. If `initObject` is not an object, it is ignored. All properties of `initObject` are copied into the new instance. The properties specified with `initObject` are available to the constructor function. This parameter is optional.

Returns

A reference to the duplicated movie clip.

Description

Method; creates an instance of the specified movie clip while the SWF file is playing. Duplicated movie clips always start playing at Frame 1, no matter what frame the original movie clip is on when the `duplicateMovieClip()` method is called. Variables in the parent movie clip are not copied into the duplicate movie clip. Movie clips that have been created using `duplicateMovieClip()` are not duplicated if you call `duplicateMovieClip()` on their parent. If the parent movie clip is deleted, the duplicate movie clip is also deleted. If you have loaded a movie clip using `MovieClip.loadMovie()` or the MovieClipLoader class, the contents of the SWF file are not duplicated. This means that you cannot save bandwidth by loading a JPEG or SWF file and then duplicating the movie clip.

Example

The following example duplicates the `circle_mc` movie clip. The code creates the movie clip, called `circle1_mc`, at the x, y coordinates 20,20.

```
circle_mc.duplicateMovieClip("circle1_mc", this.getNextHighestDepth(), {_x:20,
  _y:20});
```

See also

`duplicateMovieClip()`, `MovieClip.removeMovieClip()`

MovieClip.enabled

Availability

Flash Player 6.

Usage

my_mc.enabled:*Boolean*

Description

Property; a Boolean value that indicates whether a movie clip is enabled. The default value of enabled is true. If enabled is set to false, the movie clip's callback methods and on *action* event handlers are no longer invoked, and the Over, Down, and Up frames are disabled. The enabled property does not affect the Timeline of the movie clip; if a movie clip is playing, it continues to play. The movie clip continues to receive movie clip events (for example, mouseDown, mouseUp, keyDown, and keyUp).

The enabled property only governs the button-like properties of a movie clip. You can change the enabled property at any time; the modified movie clip is immediately enabled or disabled. The enabled property can be read out of a prototype object. If enabled is set to false, the object is not included in automatic tab ordering.

Example

The following example disables the circle_mc movie clip when the user clicks it.

```
circle_mc.onRelease = function() {
  trace("disabling the "+this._name+" movie clip.");
  this.enabled = false;
};
```

MovieClip.endFill()

Availability

Flash Player 6.

Usage

```
my_mc.endFill() : Void
```

Parameters

None.

Returns

Nothing.

Description

Method; applies a fill to the lines and curves added since the last call to `beginFill()` or `beginGradientFill()`. Flash uses the fill that was specified in the previous call to `beginFill()` or `beginGradientFill()`. If the current drawing position does not equal the previous position specified in a `moveTo()` method and a fill is defined, the path is closed with a line and then filled.

Example

The following example creates a square with red fill on the Stage.

```
this.createEmptyMovieClip("square_mc", this.getNextHighestDepth());
square_mc.beginFill(0xFF0000);
square_mc.moveTo(10, 10);
square_mc.lineTo(100, 10);
square_mc.lineTo(100, 100);
square_mc.lineTo(10, 100);
square_mc.lineTo(10, 10);
square_mc.endFill();
```

An example is also in the *drawingapi.fla* file in the HelpExamples folder. The following list gives typical paths to this folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/

See Also

`MovieClip.beginFill()`, `MovieClip.beginGradientFill()`, `MovieClip.moveTo()`

MovieClip.focusEnabled

Availability

Flash Player 6.

Usage

```
my_mc.focusEnabled:Boolean
```

Description

Property; if the value is `undefined` or `false`, a movie clip cannot receive input focus unless it is a button. If the `focusEnabled` property value is `true`, a movie clip can receive input focus even if it is not a button.

Example

The following example sets the `focusEnabled` property for the movie clip `my_mc` to `false`:

```
my_mc.focusEnabled = false;
```

MovieClip._focusrect

Availability

Flash Player 6.

Usage

```
my_mc._focusrect:Boolean
```

Description

Property; a Boolean value that specifies whether a movie clip has a yellow rectangle around it when it has keyboard focus. This property can override the global `_focusrect` property. The default value of the `_focusrect` property of a movie clip instance is `null`; meaning, the movie clip instance does not override the global `_focusrect` property. If the `_focusrect` of a movie clip instance is set to `true` or `false`, it overrides the setting of the global `_focusrect` property for the single movie clip instance.

In Flash Player 4 or Flash Player 5 SWF files, the `_focusrect` property controls the global `_focusrect` property. It is a Boolean value. This behavior was changed in Flash Player 6 and later to permit the customization of `_focusrect` on an individual movie clip basis.

Example

This example demonstrates how to hide the yellow rectangle around a specified movie clip instance in a SWF file when it has focus in a browser window. Create three movie clips called `mc1_mc`, `mc2_mc`, and `mc3_mc`, and add the following ActionScript in Frame 1 of the Timeline:

```
mc1_mc._focusrect = true;
mc2_mc._focusrect = false;
mc3_mc._focusrect = true;

mc1_mc.onRelease = traceOnRelease;
mc3_mc.onRelease = traceOnRelease;

function traceOnRelease() {
  trace(this._name);
}
```

Test the SWF file in a browser window by selecting File > Publish Preview > HTML. Give the SWF focus by clicking it in the browser window, and press Tab to focus each instance. You will not be able to execute code for this movie clip in the browser by pressing Enter or the Spacebar when `_focusrect` is disabled.

Additionally, you can test your SWF file in the test environment. Select Control > Disable Keyboard Shortcuts from the main menu in the test environment. This allows you to view the focus rectangle around the instances in the SWF file.

See Also

`Button._focusrect`, `_focusrect`

MovieClip._framesloaded

Availability

Flash Player 4.

Usage

my_mc._framesloaded:*Number*

Description

Read-only property; the number of frames that have been loaded from a streaming SWF file. This property is useful for determining whether the contents of a specific frame, and all the frames before it, have loaded and are available locally in the browser. It is also useful for monitoring the downloading of large SWF files. For example, you might want to display a message to users indicating that the SWF file is loading until a specified frame in the SWF file has finished loading.

Example

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

Enter the following ActionScript in Frame 1 of the Timeline:

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/
  this.getBytesTotal()*100);
bar_mc._xscale = pctLoaded;
```

Add the following code to Frame 2:

```
if (this._framesloaded<this._totalframes) {
  this.gotoAndPlay(1);
} else {
  this.gotoAndStop(3);
}
```

Place your content on or after Frame 3. Then add the following code to Frame 3:

```
stop();
```

See also

MovieClipLoader class

MovieClip.getBounds()

Availability

Flash Player 5.

Usage

my_mc.getBounds(*targetCoordinateSpace*:*Object*) : *Object*

Parameters

targetCoordinateSpace The target path of the Timeline whose coordinate system you want to use as a reference point.

Returns

An object with the properties xMin, xMax, yMin, and yMax.

Description

Method; returns properties that are the minimum and maximum *x* and *y* coordinate values of the instance specified by *my_mc* for the *targetCoordinateSpace* parameter.

Note: Use MovieClip.localToGlobal() and MovieClip.globalToLocal() to convert the movie clip's local coordinates to Stage coordinates, or Stage coordinates to local coordinates, respectively.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example creates a movie clip called square_mc. The code draws a square for that movie clip and uses MovieClip.getBounds() to display the coordinate values of the instance in the Output panel.

```
this.createEmptyMovieClip("square_mc", 1);
square_mc._x = 10;
square_mc._y = 10;
square_mc.beginFill(0xFF0000);
square_mc.moveTo(0, 0);
square_mc.lineTo(100, 0);
square_mc.lineTo(100, 100);
square_mc.lineTo(0, 100);
square_mc.lineTo(0, 0);
square_mc.endFill();

var bounds_obj:Object = square_mc.getBounds(this);
for (var i in bounds_obj) {
  trace(i+" --> "+bounds_obj[i]);
}
```

The following information displays in the Output panel:

```
yMax --> 110
yMin --> 10
xMax --> 110
xMin --> 10
```

See also

`MovieClip.globalToLocal()`, `MovieClip.localToGlobal()`

MovieClip.getBytesLoaded()

Availability

Flash Player 5.

Usage

```
my_mc.getBytesLoaded() : Number
```

Parameters

None.

Returns

An integer indicating the number of bytes loaded.

Description

Method; returns the number of bytes that have already loaded (streamed) for the movie clip specified by *my_mc*. You can compare this value with the value returned by `MovieClip.getBytesTotal()` to determine what percentage of a movie clip has loaded.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

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

Enter the following ActionScript in Frame 1 of the Timeline:

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/
  this.getBytesTotal()*100);
bar_mc._xscale = pctLoaded;
```

Add the following code to Frame 2:

```
if (this._framesloaded<this._totalframes) {
  this.gotoAndPlay(1);
} else {
  this.gotoAndStop(3);
}
```

Place your content on or after Frame 3. Then add the following code to Frame 3:

```
stop();
```

See also

`MovieClip.getBytesTotal()`

MovieClip.getBytesTotal()

Availability

Flash Player 5.

Usage

my_mc.getBytesTotal() : *Number*

Parameters

None.

Returns

An integer indicating the total size, in bytes, of *my_mc*.

Description

Method; returns the size, in bytes, of the movie clip specified by *my_mc*. For movie clips that are external (the root SWF file or a movie clip that is being loaded into a target or a level), the return value is the size of the SWF file.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

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

Enter the following ActionScript in Frame 1 of the Timeline:

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/
  this.getBytesTotal()*100);
bar_mc._xscale = pctLoaded;
```

Add the following code to Frame 2:

```
if (this._framesloaded<this._totalframes) {
  this.gotoAndPlay(1);
} else {
  this.gotoAndStop(3);
}
```

Place your content on or after Frame 3. Then add the following code to Frame 3:

```
stop();
```

See also

`MovieClip.getBytesLoaded()`

MovieClip.getDepth()

Availability

Flash Player 6.

Usage

```
my_mc.getDepth() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the depth of a movie clip instance. For more information, see “Managing movie clip depths” in *Using ActionScript in Flash*.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see “Assigning a class to a movie clip symbol” in *Using ActionScript in Flash*.

Example

The following code traces the depth of all movie clip instances on the Stage:

```
for (var i in this) {
  if (typeof (this[i]) == "movieclip") {
    trace("movie clip '"+this[i]._name+"' is at depth "+this[i].getDepth());
  }
}
```

See also

`MovieClip.getInstanceAtDepth()`, `MovieClip.getNextHighestDepth()`,
  `MovieClip.swapDepths()`

MovieClip.getInstanceAtDepth()

Availability

Flash Player 7.

Usage

```
my_mc.getInstanceAtDepth(depth:Number) : MovieClip
```

Parameters

`depth` An integer that specifies the depth level to query.

Returns

A reference to the MovieClip instance located at the specified depth, or `undefined` if there is no movie clip at that depth.

Description

Method; lets you determine if a particular depth is already occupied by a movie clip. You can use this method before using `MovieClip.attachMovie()`, `MovieClip.duplicateMovieClip()`, or `MovieClip.createEmptyMovieClip()` to determine if the depth parameter you want to pass to any of these methods already contains a movie clip. For more information, see “Managing movie clip depths” in *Using ActionScript in Flash*.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see “Assigning a class to a movie clip symbol” in *Using ActionScript in Flash*.

Example

The following example displays the depth occupied by the `logo_mc` movie clip instance in the Output panel:

```
this.createEmptyMovieClip("logo_mc", 1);
logo_mc.loadMovie("http://www.macromedia.com/devnet/mx/blueprint/articles/
   nielsen/spotlight_jnielsen.jpg");
trace(this.getInstanceAtDepth(1)); // output: _level0.logo_mc
```

See also

`MovieClip.getDepth()`, `MovieClip.getNextHighestDepth()`, `MovieClip.swapDepths()`

MovieClip.getNextHighestDepth()

Availability

Flash Player 7.

Usage

my_mc.getNextHighestDepth() *: Number*

Parameters

None.

Returns

An integer that reflects the next available depth index that would render above all other objects on the same level and layer within *my_mc*.

Description

Method; lets you determine a depth value that you can pass to MovieClip.attachMovie(), MovieClip.duplicateMovieClip(), or MovieClip.createEmptyMovieClip() to ensure that Flash renders the movie clip in front of all other objects on the same level and layer in the current movie clip. The value returned is 0 or higher (that is, negative numbers are not returned).

For more information, see "Managing movie clip depths" in *Using ActionScript in Flash*.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example creates a new movie clip instance, logo_mc, at the next highest depth available. At runtime, logo_mc renders in front of all other instances at the same level.

```
this.createEmptyMovieClip("logo_mc", this.getNextHighestDepth());
var logo_mcl:MovieClipLoader = new MovieClipLoader();
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  target_mc.onPress = function() {
    this.startDrag();
  };
  target_mc.onRelease = function() {
    this.stopDrag();
  };
};
logo_mcl.addListener(mclListener);
logo_mcl.loadClip("http://www.macromedia.com/devnet/mx/blueprint/articles/
  nielsen/spotlight_jnielsen.jpg", logo_mc);
```

See also

MovieClip.getDepth(), MovieClip.getInstanceAtDepth(), MovieClip.swapDepths()

MovieClip.getSWFVersion()

Availability

Flash Player 7.

Usage

my_mc.getSWFVersion() *: Number*

Parameters

None.

Returns

An integer that specifies the Flash Player version that was targeted when the SWF file loaded into *my_mc* was published.

Description

Method; returns an integer that indicates the Flash Player version for which *my_mc* was published. If *my_mc* is a JPEG file, or if an error occurs and Flash can't determine the SWF version of *my_mc*, -1 is returned.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following ActionScript displays the Flash Player version for the SWF file and for a SWF file that loads into a movie clip instance called fp6_mc.

```
trace(this.getSWFVersion()); // output: 7
fp6_mc.loadMovie("flashplayer6.swf");
my_btn.onRelease = function(){
  trace("Loaded Flash Player " + fp6_mc.getSWFVersion() + " file.");
  //output: Loaded Flash Player 6 file.
};
```

MovieClip.getTextSnapshot()

Availability

Authoring: Flash MX 2004.

Playback: SWF files published for Flash Player 6 or later, playing in Flash Player 7 or later.

Usage

```
my_mc.getTextSnapshot() : TextSnapshot
```

Parameters

None.

Returns

A TextSnapshot object that contains the static text from `my_mc`.

Description

Method; returns a TextSnapshot object that contains the text in all the static text fields in the specified movie clip; text in child movie clips is not included. This method always returns a TextSnapshot object.

Flash concatenates text and places it in the TextSnapshot object in an order that reflects the tab index order of the static text fields in the movie clip. Text fields that don't have tab index values are placed in a random order in the object, and precede any text from fields that do have tab index values. No line breaks or formatting indicates where one field ends and the next begins.

Note: You can't specify a tab index value for static text in Flash. However, other products may do so (for example, Macromedia FlashPaper).

The contents of the TextSnapshot object aren't dynamic; that is, if the movie clip moves to a different frame, or is altered in some way (for example, objects in the movie clip are added or removed), the TextSnapshot object might not represent the current text in the movie clip. To ensure that the object's contents are current, reissue this command as needed.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following code dynamically creates a TextSnapshot object called `textShapshot_txt` inside a movie clip called `text_mc`.

```
this.createEmptyMovieClip("text_mc", this.getNextHighestDepth());
text_mc.createTextField("textSnapshot_txt", text_mc.getNextHighestDepth(), 10,
  10, 240, 160);
text_mc.textSnapshot_txt.multiline = true;
text_mc.textSnapshot_txt.wordWrap = true;
text_mc.textSnapshot_txt.autoSize = true;
text_mc.textSnapshot_txt.border = true;
text_mc.textSnapshot_txt.html = true;
//
var textSnap:TextSnapshot = text_mc.getTextSnapshot();
text_mc.textSnapshot_txt.htmlText = "<textformat tabstops='[150]'>";
```

```
for (var i in textSnap) {
  text_mc.textSnapshot_txt.htmlText += "<b>"+i+"</b>\t"+textSnap[i];
}
text_mc.textSnapshot_txt.htmlText += "</textformat>";
```

The following text appears in `text_mc.textSnapshot_txt`:

```
getTextRunInfo[type Function]
setSelectColor[type Function]
findText[type Function]
hitTestTextNearPos[type Function]
getSelectedText[type Function]
getText  [type Function]
getSelected[type Function]
setSelected[type Function]
getCount[type Function]
```

See also

TextSnapshot object

MovieClip.getURL()

Availability

Flash Player 5.

Usage

```
my_mc.getURL(URL:String [,window:String, variables:String]) : Void
```

Parameters

`URL` String; the URL from which to obtain the document.

`window` String; an optional parameter specifying the name, frame, or expression that specifies the window or HTML frame that the document is loaded into. You can also use one of the following reserved target names: `_self` specifies the current frame in the current window, `_blank` specifies a new window, `_parent` specifies the parent of the current frame, and `_top` specifies the top-level frame in the current window.

`variables` String (either `"GET"` or `"POST"`); an optional parameter specifying a method for sending variables associated with the SWF file to load. If there are no variables, omit this parameter; otherwise, specify whether to load variables using a `GET` or `POST` method. `GET` appends the variables to the end of the URL and is used for a small numbers of variables. `POST` sends the variables in a separate HTTP header and is used for long strings of variables.

Returns

Nothing.

Description

Method; loads a document from the specified URL into the specified window. The `getURL` method can also be used to pass variables to another application defined at the URL using a `GET` or `POST` method.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following ActionScript creates a new movie clip instance and opens the Macromedia website in a new browser window:

```
this.createEmptyMovieClip("loader_mc", this.getNextHighestDepth());
loader_mc.getURL("http://www.macromedia.com", “_blank”);
```

The `getURL()` method also allows you to send variables to a remove server-side script, as seen in the following code:

```
this.createEmptyMovieClip("loader_mc", this.getNextHighestDepth());
loader_mc.username = "some user input";
loader_mc.password = "random string";
loader_mc.getURL("http://www.flash-mx.com/mm/viewscope.cfm", "_blank", "GET");
```

See also

`getURL()`, `LoadVars.sendAndLoad()`, `LoadVars.send()`

MovieClip.globalToLocal()

Availability

Flash Player 5.

Usage

my_mc.globalToLocal(*point:Object*) : *Void*

Parameters

point The name or identifier of an object created with the generic Object class. The object specifies the *x* and *y* coordinates as properties.

Returns

Nothing.

Description

Method; converts the *point* object from Stage (global) coordinates to the movie clip's (local) coordinates.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

Add the following ActionScript to a FLA or AS file in the same directory as an image called photo1.jpg:

```
this.createTextField("coords_txt", this.getNextHighestDepth(), 10, 10, 100,
  22);
coords_txt.html = true;
coords_txt.multiline = true;
coords_txt.autoSize = true;
this.createEmptyMovieClip("target_mc", this.getNextHighestDepth());
target_mc._x = 100;
target_mc._y = 100;
target_mc.loadMovie("photo1.jpg");

var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
  var point:Object = {x:_xmouse, y:_ymouse};
  target_mc.globalToLocal(point);
  var rowHeaders = "<b> &nbsp; \t</b><b>_x\t</b><b>_y</b>";
  var row_1 = "_root\t"+_xmouse+"\t"+_ymouse;
  var row_2 = "target_mc\t"+point.x+"\t"+point.y;
  coords_txt.htmlText = "<textformat tabstops='[100, 150]'>";
  coords_txt.htmlText += rowHeaders;
  coords_txt.htmlText += row_1;
  coords_txt.htmlText += row_2;
  coords_txt.htmlText += "</textformat>";
};
Mouse.addListener(mouseListener);
```

See also

`MovieClip.getBounds()`, `MovieClip.localToGlobal()`

MovieClip.gotoAndPlay()

Availability

Flash Player 5.

Usage

```
my_mc.gotoAndPlay(frame:Object) : Void
```

Parameters

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

Returns

Nothing.

Description

Method; starts playing the SWF file at the specified frame. If you want to specify a scene as well as a frame, use `gotoAndPlay()`.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

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

Enter the following ActionScript in Frame 1 of the Timeline:

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/
  this.getBytesTotal()*100);
bar_mc._xscale = pctLoaded;
```

Add the following code to Frame 2:

```
if (this._framesloaded<this._totalframes) {
  this.gotoAndPlay(1);
} else {
  this.gotoAndStop(3);
}
```

Place your content on or after Frame 3. Then add the following code to Frame 3:

```
stop();
```

See also

`gotoAndPlay()`, `play()`

MovieClip.gotoAndStop()

Availability

Flash Player 5.

Usage

```
my_mc.gotoAndStop(frame:Object) : Void
```

Parameters

frame The frame number to which the playhead is sent.

Returns

Nothing.

Description

Method; brings the playhead to the specified frame of the movie clip and stops it there. If you want to specify a scene in addition to a frame, use `gotoAndStop()`.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

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

Enter the following ActionScript in Frame 1 of the Timeline:

```
var pctLoaded:Number = Math.round(this.getBytesLoaded()/
  this.getBytesTotal()*100);
bar_mc._xscale = pctLoaded;
```

Add the following code to Frame 2:

```
if (this._framesloaded<this._totalframes) {
  this.gotoAndPlay(1);
} else {
  this.gotoAndStop(3);
}
```

Place your content on or after Frame 3. Then add the following code to Frame 3:

```
stop();
```

See also

`gotoAndStop()`, `stop()`

MovieClip._height

Availability

Flash Player 4.

Usage

my_mc._height:*Number*

Description

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

Example

The following code example displays the height and width of a movie clip in the Output panel:

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var image_mcl:MovieClipLoader = new MovieClipLoader();
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  trace(target_mc._name+" = "+target_mc._width+" X "+target_mc._height+"
  pixels");
};
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.macromedia.com/devnet/mx/blueprint/articles/
  nielsen/spotlight_jnielsen.jpg", image_mc);
```

See Also

MovieClip._width

MovieClip.hitArea

Availability

Flash Player 6.

Usage

my_mc.hitArea:*Object*

Returns

A reference to a movie clip.

Description

Property; designates another movie clip to serve as the hit area for a movie clip. If the `hitArea` property does not exist or is `null` or `undefined`, the movie clip itself is used as the hit area. The value of the `hitArea` property may be a reference to a movie clip object.

You can change the `hitArea` property at any time; the modified movie clip immediately takes on the new hit area behavior. The movie clip designated as the hit area does not need to be visible; its graphical shape, although not visible, is hit-tested. The `hitArea` property can be read out of a prototype object.

Example

The following example sets the `circle_mc` movie clip as the hit area for the `square_mc` movie clip. Place these two movie clips on the Stage and test the document. When you click `circle_mc`, the `square_mc` movie clip traces that it has been clicked.

```
square_mc.hitArea = circle_mc;
square_mc.onRelease = function() {
  trace("hit! "+this._name);
};
```

You can also set the `circle_mc` movie clip `visible` property to `false` to hide the hit area for `square_mc`.

```
circle_mc._visible = false;
```

See Also

`MovieClip.hitTest()`

MovieClip.hitTest()

Availability

Flash Player 5.

Usage

my_mc`.hitTest(`*x:Number, y:Number, shapeFlag:Boolean*`)` *: Boolean*

my_mc`.hitTest(`*target:Object*`)` *: Boolean*

Parameters

x The *x* coordinate of the hit area on the Stage.

y The *y* coordinate of the hit area on the Stage.

The *x* and *y* coordinates are defined in the global coordinate space.

target The target path of the hit area that may intersect or overlap with the instance specified by *my_mc*. The *target* parameter usually represents a button or text-entry field.

shapeFlag A Boolean value specifying whether to evaluate the entire shape of the specified instance (*true*), or just the bounding box (*false*). This parameter can be specified only if the hit area is identified using *x* and *y* coordinate parameters.

Returns

A Boolean value of `true` if *my_mc* overlaps with the specified hit area, `false` otherwise.

Description

Method; evaluates the instance specified by *my_mc* to see if it overlaps or intersects with the hit area identified by the *target* or *x* and *y* coordinate parameters.

Usage 1: Compares the *x* and *y* coordinates to the shape or bounding box of the specified instance, according to the *shapeFlag* setting. If *shapeFlag* is set to `true`, only the area actually occupied by the instance on the Stage is evaluated, and if *x* and *y* overlap at any point, a value of `true` is returned. This is useful for determining if the movie clip is within a specified hit or hotspot area.

Usage 2: Evaluates the bounding boxes of the *target* and specified instance, and returns `true` if they overlap or intersect at any point.

Example

The following example uses `hitTest()` to determine if the movie clip `circle_mc` overlaps or intersects the movie clip `square_mc` when the user releases the mouse button:

```
square_mc.onPress = function() {
  this.startDrag();
};
square_mc.onRelease = function() {
  this.stopDrag();
  if (this.hitTest(circle_mc)) {
    trace("you hit the circle");
  }
};
```

See also

`MovieClip.getBounds()`, `MovieClip.globalToLocal()`, `MovieClip.localToGlobal()`

MovieClip.lineStyle()

Availability

Flash Player 6.

Usage

```
my_mc.lineStyle([thickness:Number[, rgb:Number[, alpha:Number]]]) : Void
```

Parameters

`thickness` An integer that indicates the thickness of the line in points; valid values are 0 to 255. If a number is not specified, or if the parameter is `undefined`, a line is not drawn. If a value of less than 0 is passed, Flash uses 0. The value 0 indicates hairline thickness; the maximum thickness is 255. If a value greater than 255 is passed, the Flash interpreter uses 255.

`rgb` A hex color value (for example, red is 0xFF0000, blue is 0x0000FF, and so on) of the line. If a value isn't indicated, Flash uses 0x000000 (black).

`alpha` An integer that indicates the alpha value of the line's color; valid values are 0–100. If a value isn't indicated, Flash uses 100 (solid). If the value is less than 0, Flash uses 0; if the value is greater than 100, Flash uses 100.

Returns

Nothing.

Description

Method; specifies a line style that Flash uses for subsequent calls to `lineTo()` and `curveTo()` until you call `lineStyle()` with different parameters. You can call `lineStyle()` in the middle of drawing a path to specify different styles for different line segments within a path.

Note: Calls to `clear()` will set the line style back to `undefined`.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following code draws a triangle with a 5-pixel, solid magenta line with no fill:

```
this.createEmptyMovieClip("triangle_mc", 1);
triangle_mc.lineStyle(5, 0xff00ff, 100);
triangle_mc.moveTo(200, 200);
triangle_mc.lineTo(300, 300);
triangle_mc.lineTo(100, 300);
triangle_mc.lineTo(200, 200);
```

See also

`MovieClip.beginFill()`, `MovieClip.beginGradientFill()`, `MovieClip.clear()`, `MovieClip.curveTo()`, `MovieClip.lineTo()`, `MovieClip.moveTo()`

MovieClip.lineTo()

Availability

Flash Player 6.

Usage

```
my_mc.lineTo(x:Number, y:Number) : Void
```

Parameters

x An integer indicating the horizontal position relative to the registration point of the parent movie clip.

y An integer indicating the vertical position relative to the registration point of the parent movie clip.

Returns

Nothing.

Description

Method; draws a line using the current line style from the current drawing position to (*x*, *y*); the current drawing position is then set to (*x*, *y*). If the movie clip that you are drawing in contains content that was created with the Flash drawing tools, calls to `lineTo()` are drawn underneath the content. If you call `lineTo()` before any calls to the `moveTo()` method, the current drawing position defaults to (`0`, `0`). If any of the parameters are missing, this method fails and the current drawing position is not changed.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example draws a triangle with a 5-pixel, solid magenta line and a partially transparent blue fill:

```
this.createEmptyMovieClip("triangle_mc", 1);
triangle_mc.beginFill(0x0000FF, 30);
triangle_mc.lineStyle(5, 0xFF00FF, 100);
triangle_mc.moveTo(200, 200);
triangle_mc.lineTo(300, 300);
triangle_mc.lineTo(100, 300);
triangle_mc.lineTo(200, 200);
triangle_mc.endFill();
```

See also

`MovieClip.beginFill()`, `MovieClip.createEmptyMovieClip()`, `MovieClip.endFill()`, `MovieClip.lineStyle()`, `MovieClip.moveTo()`

MovieClip.loadMovie()

Availability

Flash Player 5.

Usage

```
my_mc.loadMovie(url:String [,variables:String]) : Void
```

Parameters

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

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

Returns

Nothing.

Description

Method; loads SWF or JPEG files into a movie clip in Flash Player while the original SWF file is playing.

Tip: If you want to monitor the progress of the download, use `MovieClipLoader.loadClip()` instead of this function.

Without the `loadMovie()` method, Flash Player displays a single SWF file and then closes. The `loadMovie()` method lets you display several SWF files at once and switch between SWF files without loading another HTML document.

A SWF file or image loaded into a movie clip inherits the position, rotation, and scale properties of the movie clip. You can use the target path of the movie clip to target the loaded SWF file.

When calling `loadMovie()`, set the `MovieClip._lockroot` property to `true` in the loader movie, as shown in the following code. If you don't set `_lockroot` to `true` in the loader movie, the loader has access only to its own library, but not the library in the loaded movie:

```
myMovieClip._lockroot = true;
```

Use the `MovieClip.unloadMovie()` method to remove SWF files or images loaded with the `loadMovie()` method. Use the `MovieClip.loadVariables()` method to keep the active SWF file, and update the variables of the SWF file with new values.

Using event handlers with MovieClip.loadMovie() can be unpredictable. If you attach an event handler to a button using on() or if you create a dynamic handler using an event handler method such as MovieClip.onPress, and then you call loadMovie(), the event handler does not remain after the new content is loaded. However, if you attach an event handler to a movie clip using onClipEvent() or on(), and then call loadMovie() on that movie clip, the event handler remains after the new content is loaded.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see “Assigning a class to a movie clip symbol” in *Using ActionScript in Flash*.

Example

As shown in the following example, you can use loadMovie() to load the image picture.jpg into a movie clip and use the MovieClip.onPress() method to make the image act like a button. Loading a JPEG using loadMovie() replaces the movie clip with the image but doesn’t give you access to movie clip methods. To get access to movie clip methods, you must create an empty parent movie clip and a container child movie clip. Load the image into the container and place the event handler on the parent movie clip.

```
// Creates a parent movie clip to hold the container
this.createEmptyMovieClip("logo_mc", this.getNextHighestDepth());
// creates a child movie clip inside of "mc_1"
// this is the movie clip the image will replace
logo_mc.createEmptyMovieClip("container_mc",0);
logo_mc.container_mc.loadMovie("http://www.macromedia.com/images/shared/
  product_boxes/80x92/studio_flashpro.jpg");
// put event handler on the parent movie clip mc_1
logo_mc.onPress = function() {
  trace("It works");
};
```

See also

MovieClip.loadMovie(), loadMovieNum(), MovieClip.loadVariables(),
  MovieClip.unloadMovie(), unloadMovie(), unloadMovieNum()

MovieClip.loadVariables()

Availability

Flash Player 5; behavior changed in Flash Player 7.

Usage

`my_mc.loadVariables(url:String [, variables:String]) : Void`

Parameters

`url` The absolute or relative URL for the external file that contains the variables to be loaded. If the SWF file issuing this call is running in a web browser, `url` must be in the same domain as the SWF file; for details, see "Description," below.

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

Returns

Nothing.

Description

Method; reads data from an external file and sets the values for variables in `my_mc`. The external file can be a text file generated by ColdFusion, a CGI script, Active Server Page (ASP), or PHP script and can contain any number of variables.

This method can also be used to update variables in the active movie clip with new values.

This method requires that the text of the URL be in the standard MIME format: *application/x-www-form-urlencoded* (CGI script format).

In SWF files running in a version earlier than Flash Player 7, `url` must be in the same superdomain as the SWF file that is issuing this call. A superdomain is derived by removing the left-most component of a file's URL. For example, a SWF file at www.someDomain.com can load data from a source at store.someDomain.com because both files are in the same superdomain of someDomain.com.

In SWF files of any version running in Flash Player 7 or later, `url` must be in exactly the same domain as the SWF file that is issuing this call (see "Flash Player security features" in *Using ActionScript in Flash*). For example, a SWF file at www.someDomain.com can load data only from sources that are also at www.someDomain.com. If you want to load data from a different domain, you can place a *cross-domain policy file* on the server hosting the data source that is being accessed. For more information, see "About allowing cross-domain data loading" in *Using ActionScript in Flash*.

If you want to load variables into a specific level, use `loadVariablesNum()` instead of `loadVariables()`.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example loads information from a text file called params.txt into the `target_mc` movie clip that is created using `createEmptyMovieClip()`. The `setInterval()` function is used to check the loading progress. The script checks for a variable in the params.txt file named `done`.

```
this.createEmptyMovieClip("target_mc", this.getNextHighestDepth());
target_mc.loadVariables("params.txt");
function checkParamsLoaded() {
  if (target_mc.done == undefined) {
    trace("not yet.");
  } else {
    trace("finished loading. killing interval.");
    trace("-------------");
    for (i in target_mc) {
      trace(i+": "+target_mc[i]);
    }
    trace("-------------");
    clearInterval(param_interval);
  }
}
var param_interval = setInterval(checkParamsLoaded, 100);

/* params.txt includes the following text
var1="hello"&var2="goodbye"&done="done"
*/
```

See also

`MovieClip.loadMovie()`, `MovieClip.loadVariables()`, `loadVariablesNum()`,
  `MovieClip.unloadMovie()`

MovieClip.localToGlobal()

Availability

Flash Player 5.

Usage

`my_mc.localToGlobal(point:Object) : Void`

Parameters

`point` The name or identifier of an object created with the Object class, specifying the *x* and *y* coordinates as properties.

Returns

Nothing.

Description

Method; converts the `point` object from the movie clip's (local) coordinates to the Stage (global) coordinates.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example converts *x* and *y* coordinates of the `my_mc` object, from the movie clip's (local) coordinates to the Stage (global) coordinates. The center point of the movie clip is reflected after you click and drag the instance.

```
this.createTextField("point_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
  var point:Object = {x:my_mc._width/2, y:my_mc._height/2};
  my_mc.localToGlobal(point);
  point_txt.text = "x:"+point.x+", y:"+point.y;
};
Mouse.addListener(mouseListener);
my_mc.onPress = function() {
  this.startDrag();
};
my_mc.onRelease = function() {
  this.stopDrag();
};
}
```

See also

`MovieClip.globalToLocal()`

MovieClip._lockroot

Availability

Flash Player 7.

Usage

`my_mc._lockroot:Boolean`

Description

Property; specifies what _root refers to when a SWF file is loaded into a movie clip. The `_lockroot` property is `undefined` by default. You can set this property within the SWF file that is being loaded or in the handler that is loading the movie clip.

For example, suppose you have a document called Games.fla that lets a user choose a game to play, and loads the game (for example, Chess.swf) into the `game_mc` movie clip. You want to make sure that, if `_root` is used in Chess.swf, it still refers to `_root` in Chess.swf after being loaded into Games.swf. If you have access to Chess.fla and publish it to Flash Player 7 or later, you can add this statement to Chess.fla on the main Timeline:

```
this._lockroot = true;
```

If you don't have access to Chess.fla (for example, if you are loading Chess.swf from someone else's site into `chess_mc`), you can set its `_lockroot` property when you load it, as shown below. In this case, Chess.swf can be published for any version of Flash Player, as long as Games.swf is published for Flash Player 7 or later. Place the following ActionScript on the main Timeline:

```
chess_mc._lockroot = true;
```

When calling `loadMovie()`, set the `MovieClip._lockroot` property to `true` in the loader movie, as shown in the following code. If you don't set `_lockroot` to `true` in the loader movie, the loader has access only to its own library, but not the library in the loaded movie:

```
myMovieClip._lockroot = true;
```

Example

In the following example, lockroot.fla has `_lockroot` applied to the main SWF file. If it is loaded into another FLA document, `_root` will always refer to the scope of lockroot.swf, which helps prevent conflicts. Place the following ActionScript on the main Timeline of lockroot.fla.

```
this._lockroot = true;
_root.myVar = 1;
_root.myOtherVar = 2;
trace("from lockroot.swf");
for (i in _root) {
  trace("  "+i+" -> "+_root[i]);
}
trace("");
```

Which traces the following information:

```
from lockroot.swf
myOtherVar -> 2
myVar -> 1
```

```
_lockroot -> true
$version -> WIN 7,0,19,0
```

The following example loads two SWF files, lockroot.swf and nolockroot.swf. The lockroot.fla document contains the previous ActionScript. The nolockroot FLA file has the following code placed on Frame 1 of the Timeline:

```
_root.myVar = 1;
_root.myOtherVar = 2;
trace("from nolockroot.swf");
for (i in _root) {
  trace("  "+i+" -> "+_root[i]);
}
trace("");
```

The lockroot.swf file has `_lockroot` applied to it, and nolockroot.swf does not. After the files are loaded, each file dumps variables from their `_root` scopes. Place the following ActionScript on the main Timeline of a FLA document:

```
this.createEmptyMovieClip("lockroot_mc", this.getNextHighestDepth());
lockroot_mc.loadMovie("lockroot.swf");
this.createEmptyMovieClip("nolockroot_mc", this.getNextHighestDepth());
nolockroot_mc.loadMovie("nolockroot.swf");
function dumpRoot() {
  trace("from current SWF file");
  for (i in _root) {
    trace("  "+i+" -> "+_root[i]);
  }
  trace("");
}
dumpRoot();
```

which traces the following information:

```
from current SWF file
dumpRoot -> [type Function]
$version -> WIN 7,0,19,0
nolockroot_mc -> _level0.nolockroot_mc
lockroot_mc -> _level0.lockroot_mc

from nolockroot.swf
myVar -> 1
i -> lockroot_mc
dumpRoot -> [type Function]
$version -> WIN 7,0,19,0
nolockroot_mc -> _level0.nolockroot_mc
lockroot_mc -> _level0.lockroot_mc

from lockroot.swf
myOtherVar -> 2
myVar -> 1
```

The file with no `_lockroot` applied contains all of the other variables contained in the root SWF as well. If you don't have access to the nolockroot.fla, then you can change the `_lockroot` in the main FLA document above using the following ActionScript added to the main Timeline:

```
this.createEmptyMovieClip("nolockroot_mc", this.getNextHighestDepth());
```

```
nolockroot_mc._lockroot = true;
nolockroot_mc.loadMovie("nolockroot.swf");
```

which would then trace the following:

```
from current SWF file
dumpRoot -> [type Function]
$version -> WIN 7,0,19,0
nolockroot_mc -> _level0.nolockroot_mc
lockroot_mc -> _level0.lockroot_mc

from nolockroot.swf
myOtherVar -> 2
myVar -> 1

from lockroot.swf
myOtherVar -> 2
myVar -> 1
```

See also

`MovieClip.attachMovie()`, `MovieClip.loadMovie()`, `MovieClipLoader.onLoadInit`, _root, and "About loading components" in *Using Components*.

MovieClip.menu

Availability

Flash Player 7.

Usage

my_mc.menu:*ContextMenu* = *contextMenu*

Parameters

contextMenu A ContextMenu object.

Description

Property; associates the specified ContextMenu object with the movie clip *my_mc*. The ContextMenu class lets you modify the context menu that appears when the user right-clicks (Windows) or Control-clicks (Macintosh) in Flash Player.

Example

The following example assigns the ContextMenu object `menu_cm` to the movie clip `image_mc`. The ContextMenu object contains a custom menu item labeled "View Image in Browser..." that has an associated function named `viewImage()`.

```
var menu_cm:ContextMenu = new ContextMenu();
menu_cm.customItems.push(new ContextMenuItem("View Image in Browser...",
  viewImage));
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  target_mc.menu = menu_cm;
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("photo1.jpg", image_mc);

function viewImage(target_mc:MovieClip, obj:Object) {
  getURL(target_mc._url, "_blank");
}
```

When you right-click or Control-click the image at runtime, select View Image in Browser from the context menu to open the image in a browser window.

See also

`Button.menu`, `ContextMenu class`, `ContextMenuItem class`, `TextField.menu`

MovieClip.moveTo()

Availability

Flash Player 6.

Usage

```
my_mc.moveTo(x:Number, y:Number) : Void
```

Parameters

`x` An integer indicating the horizontal position relative to the registration point of the parent movie clip.

`y` An integer indicating the vertical position relative to the registration point of the parent movie clip.

Returns

Nothing.

Description

Method; moves the current drawing position to (*x*, *y*). If any of the parameters are missing, this method fails and the current drawing position is not changed.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example draws a triangle with a 5-pixel, solid magenta line and a partially transparent blue fill:

```
this.createEmptyMovieClip("triangle_mc", 1);
triangle_mc.beginFill(0x0000FF, 30);
triangle_mc.lineStyle(5, 0xFF00FF, 100);
triangle_mc.moveTo(200, 200);
triangle_mc.lineTo(300, 300);
triangle_mc.lineTo(100, 300);
triangle_mc.lineTo(200, 200);
triangle_mc.endFill();
```

See also

`MovieClip.createEmptyMovieClip()`, `MovieClip.lineStyle()`, `MovieClip.lineTo()`

MovieClip._name

Availability

Flash Player 4.

Usage

my_mc._name:*String*

Description

Property; the instance name of the movie clip specified by *my_mc*.

Example

The following example lets you right-click or Control-click a movie clip on the Stage and select "Info..." from the context menu to view information about that instance. Add several movie clips with instance names, and then add the following ActionScript to your AS or FLA file:

```
var menu_cm:ContextMenu = new ContextMenu();
menu_cm.customItems.push(new ContextMenuItem("Info...", getMCInfo));
function getMCInfo(target_mc:MovieClip, obj:Object) {
  trace("You clicked on the movie clip '"+target_mc._name+"'.");
  trace("\t width:"+target_mc._width+", height:"+target_mc._height);
  trace("");
}
for (var i in this) {
  if (typeof (this[i]) == 'movieclip') {
    this[i].menu = menu_cm;
  }
}
```

See Also

Button._name

MovieClip.nextFrame()

Availability

Flash Player 5.

Usage

`*my_mc*.nextFrame() : *Void*`

Parameters

None.

Returns

Nothing.

Description

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

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example loads content into a SWF file using `_framesloaded` and `nextFrame()`. Do not add any code to Frame 1, but add the following ActionScript to Frame 2 of the Timeline:

```
if (this._framesloaded >= 3) {
  this.nextFrame();
} else {
  this.gotoAndPlay(1);
}
```

Then, add the following code (and the content you want to load) on Frame 3:

```
stop();
```

See also

`nextFrame()`, `MovieClip.prevFrame()`, `prevFrame()`

MovieClip.onData

Availability

Flash Player 6.

Usage

```
my_mc.onData = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when a movie clip receives data from a `MovieClip.loadVariables()` or `MovieClip.loadMovie()` call. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

This handler can be used only with movie clips for which you have a symbol in the library that is associated with a class. If you want an event handler to be invoked when a specific movie clip receives data, you must use onClipEvent() instead of this handler. The latter handler is invoked when any movie clip receives data.

Example

The following example illustrates the correct use of `MovieClip.onData()` and `onClipEvent(data)`:

```
// symbol_mc is a movie clip symbol in the library.
// It is linked to the MovieClip class.
// The following function is triggered for each instance of symbol_mc
//   when it receives data.
symbol_mc.onData = function() {
   trace("The movie clip has received data");
}
// dynamic_mc is a movie clip that is being loaded with MovieClip.loadMovie().
//   This code attempts to call a function when the clip is loaded,
//   but it will not work, because the loaded SWF is not a symbol
//   in the library associated with the MovieClip class.
function output()
{
    trace("Will never be called.");
}
dynamic_mc.onData = output;
dynamic_mc.loadMovie("replacement.swf");
// The following function is invoked for any movie clip that
// receives data, whether it is in the library or not.
// Therefore, this function is invoked when symbol_mc is instantiated
//    and also when replacement.swf is loaded.
OnClipEvent( data ) {
   trace("The movie clip has received data");
}
```

See also

`onClipEvent()`

MovieClip.onDragOut

Availability

Flash Player 6.

Usage

```
my_mc.onDragOut = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the mouse button is pressed and the pointer rolls outside the object. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example defines a function for the `onDragOut` method that sends a `trace()` action to the Output panel:

```
my_mc.onDragOut = function () {
  trace ("onDragOut called");
};
```

See also

`MovieClip.onDragOver`

MovieClip.onDragOver

Availability

Flash Player 6.

Usage

```
my_mc.onDragOver = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the pointer is dragged outside and then over the movie clip. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example defines a function for the `onDragOver` method that sends a `trace()` action to the Output panel:

```
my_mc.onDragOver = function () {
  trace ("onDragOver called");
};
```

See also

`MovieClip.onDragOut`

MovieClip.onEnterFrame

Availability

Flash Player 6.

Usage

```
my_mc.onEnterFrame = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked repeatedly at the frame rate of the SWF file. The actions associated with the `enterFrame` event are processed before any frame actions that are attached to the affected frames.

You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see “Assigning a class to a movie clip symbol” in *Using ActionScript in Flash*.

Example

The following example defines a function for the `onEnterFrame` method that sends a `trace()` action to the Output panel:

```
my_mc.onEnterFrame = function () {
  trace ("onEnterFrame called");
};
```

MovieClip.onKeyDown

Availability

Flash Player 6.

Usage

```
my_mc.onKeyDown = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when a movie clip has input focus and a key is pressed. The `onKeyDown` event handler is invoked with no parameters. You can use the `Key.getAscii()` and `Key.getCode()` methods to determine which key was pressed. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

The `onKeyDown` event handler works only if the movie clip has input focus enabled and set. First, the `MovieClip.focusEnabled` property must be set to `true` for the movie clip. Then, the clip must be given focus. This can be done either by using `Selection.setFocus()` or by setting the Tab key to navigate to the clip.

If `Selection.setFocus()` is used, the path for the movie clip must be passed to `Selection.setFocus()`. It is very easy for other elements to take the focus back after the mouse is moved.

Example

The following example defines a function for the `onKeyDown()` method that sends a `trace()` action to the Output panel. Create a movie clip called my_mc and add the following ActionScript to your FLA or AS file:

```
my_mc.onKeyDown = function() {
  trace("key was pressed");
};
```

The movie clip must have focus for the `onKeyDown` event handler to work. Add the following ActionScript to set input focus.

```
my_mc.tabEnabled = true;
my_mc.focusEnabled = true;
Selection.setFocus(my_mc);
```

When you tab to the movie clip and press a key, `key was pressed` displays in the Output panel. However, this does not occur after you move the mouse, because the movie clip loses focus. Therefore, you should use `Key.onKeyDown` in most cases.

See also

`MovieClip.onKeyUp`

MovieClip.onKeyUp

Availability

Flash Player 6.

Usage

```
my_mc.onKeyUp = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when a key is released. The `onKeyUp` event handler is invoked with no parameters. You can use the `Key.getAscii()` and `Key.getCode()` methods to determine which key was pressed. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

The `onKeyUp` event handler works only if the movie clip has input focus enabled and set. First, the `MovieClip.focusEnabled` property must be set to `true` for the movie clip. Then, the clip must be given focus. This can be done either by using `Selection.setFocus()` or by setting the Tab key to navigate to the clip.

If `Selection.setFocus()` is used, the path for the movie clip must be passed to `Selection.setFocus()`. It is very easy for other elements to take the focus back after the mouse is moved.

Example

The following example defines a function for the `onKeyUp` method that sends a `trace()` action to the Output panel:

```
my_mc.onKeyUp = function () {
  trace ("onKeyUp called");
};
```

The following example sets input focus:

```
my_mc.focusEnabled = true;
Selection.setFocus(my_mc);
```

See Also

`MovieClip.onKeyDown`

MovieClip.onKillFocus

Availability

Flash Player 6.

Usage

```
my_mc.onKillFocus = function (newFocus:Object) {
  // your statements here
}
```

Parameters

`newFocus`	The object that is receiving the keyboard focus.

Returns

Nothing.

Description

Event handler; invoked when a movie clip loses keyboard focus. The `onKillFocus` method receives one parameter, *`newFocus`*, which is an object that represents the new object receiving the focus. If no object receives the focus, *`newFocus`* contains the value `null`.

You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example displays writes information about the movie clip that loses focus, and the instance that currently has focus. Two movie clips, called `my_mc` and `other_mc`, are on the Stage. Add the following ActionScript to your AS or FLA document:

```
my_mc.onRelease = Void;
other_mc.onRelease = Void;
my_mc.onKillFocus = function(newFocus) {
  trace("onKillFocus called, new focus is: "+newFocus);
};
```

Tab between the two instances, and information displays in the Output panel.

See Also

`MovieClip.onSetFocus`

MovieClip.onLoad

Availability

Flash Player 6.

Usage

```
my_mc.onLoad = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the movie clip is instantiated and appears in the Timeline. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

This handler can be used only with movie clips for which you have a symbol in the library that is associated with a class. If you want an event handler to be invoked when a specific movie clip loads, for example when you use `MovieClip.loadMovie()` to load a SWF file dynamically, you must use `onClipEvent(load)` instead of this handler. The latter handler is invoked when any movie clip loads.

Example

The following example illustrates one way to use `MovieClip.onLoad()` with `setInterval()` to check that a file has loaded into a movie clip that's created at runtime:

```
this.createEmptyMovieClip("tester_mc", 1);
tester_mc.loadMovie("http://www.yourserver.com/your_movie.swf");
function checkLoaded(target_mc:MovieClip) {
  var pctLoaded:Number = target_mc.getBytesLoaded()/
  target_mc.getBytesTotal()*100;
  if (!isNaN(pctLoaded) && (pctLoaded>0)) {
    target_mc.onLoad = doOnLoad;
    trace("clearing interval");
    clearInterval(myInterval);
  }
}
var myInterval:Number = setInterval(checkLoaded, 100, tester_mc);
function doOnLoad() {
  trace("movie loaded");
}
```

The following example displays the equivalent of the previous ActionScript example:

```
this.createEmptyMovieClip("tester_mc", 1);
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  trace("movie loaded");
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.yourserver.com/your_movie.swf", tester_mc);
```

See also

onClipEvent(), MovieClipLoader class

MovieClip.onMouseDown

Availability

Flash Player 6.

Usage

```
my_mc.onMouseDown = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the mouse button is pressed. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example defines a function for the `onMouseDown` method that sends a `trace()` action to the Output panel:

```
my_mc.onMouseDown = function () {
  trace ("onMouseDown called");
}
```

MovieClip.onMouseMove

Availability

Flash Player 6.

Usage

```
my_mc.onMouseMove = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the mouse moves. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example defines a function for the `onMouseMove` method that sends a `trace()` action to the Output panel:

```
my_mc.onMouseMove = function () {
  trace ("onMouseMove called");
};
```

MovieClip.onMouseUp

Availability

Flash Player 6.

Usage

```
my_mc.onMouseUp = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the mouse button is released. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example defines a function for the `onMouseUp` method that sends a `trace()` action to the Output panel:

```
my_mc.onMouseUp = function () {
  trace ("onMouseUp called");
};
```

MovieClip.onPress

Availability

Flash Player 6.

Usage

```
my_mc.onPress = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the user clicks the mouse while the pointer is over a movie clip. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example defines a function for the `onPress` method that sends a `trace()` action to the Output panel:

```
my_mc.onPress = function () {
  trace ("onPress called");
};
```

MovieClip.onRelease

Availability

Flash Player 6.

Usage

```
my_mc.onRelease = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked the mouse button is released over a movie clip. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example defines a function for the `onPress` method that sends a `trace()` action to the Output panel:

```
logo_mc.onRelease = function() {
  trace("onRelease called");
};
```

MovieClip.onReleaseOutside

Availability

Flash Player 6.

Usage

```
my_mc.onReleaseOutside = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked after the mouse button has been pressed inside the movie clip area and then released outside the movie clip area.

You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example defines a function for the `onReleaseOutside` method that sends a `trace()` action to the Output panel:

```
my_mc.onReleaseOutside = function () {
  trace ("onReleaseOutside called");
};
```

MovieClip.onRollOut

Availability

Flash Player 6.

Usage

```
my_mc.onRollOut = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the pointer moves outside a movie clip area.

You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example defines a function for the `onRollOut` method that sends a `trace()` action to the Output panel:

```
my_mc.onRollOut = function () {
  trace ("onRollOut called");
};
```

MovieClip.onRollOver

Availability

Flash Player 6.

Usage

```
my_mc.onRollOver = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the pointer moves over a movie clip area.

You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example defines a function for the `onRollOver` method that sends a `trace()` to the Output panel:

```
my_mc.onRollOver = function () {
  trace ("onRollOver called");
};
```

MovieClip.onSetFocus

Availability

Flash Player 6.

Usage

```
my_mc.onSetFocus = function(oldFocus){
  // your statements here
}
```

Parameters

oldFocus	The object to lose focus.

Returns

Nothing.

Description

Event handler; invoked when a movie clip receives keyboard focus. The *oldFocus* parameter is the object that loses the focus. For example, if the user presses the Tab key to move the input focus from a movie clip to a text field, *oldFocus* contains the movie clip instance.

If there is no previously focused object, *oldFocus* contains a `null` value.

You must define a function that executes when the event handler in invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example displays information about the movie clip that receives keyboard focus, and the instance that previously had focus. Two movie clips, called `my_mc` and `other_mc` are on the Stage. Add the following ActionScript to your AS or FLA document:

```
my_mc.onRelease = Void;
other_mc.onRelease = Void;
my_mc.onSetFocus = function(oldFocus) {
  trace("onSetFocus called, previous focus was: "+oldFocus);
};
```

Tab between the two instances, and information displays in the Output panel.

See Also

`MovieClip.onKillFocus`

MovieClip.onUnload

Availability

Flash Player 6.

Usage

```
my_mc.onUnload = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked in the first frame after the movie clip is removed from the Timeline. Flash processes the actions associated with the `onUnload` event handler before attaching any actions to the affected frame. You must define a function that executes when the event handler is invoked. You can define the function on the Timeline or in a class file that extends the MovieClip class or is linked to a symbol in the library. For more information, see “Assigning a class to a movie clip symbol” in *Using ActionScript in Flash*.

Example

The following example defines a function for the `MovieClip.onUnload` method that sends a `trace()` action to the Output panel:

```
my_mc.onUnload = function () {
  trace ("onUnload called");
};
```

MovieClip._parent

Availability

Flash Player 5.

Usage

`my_mc``._parent.`*`property`*

`_parent.`*`property`*

Description

Property; a reference to the movie clip or object that contains the current movie clip or object. The current object is the object containing the ActionScript code that references `_parent`. Use the `_parent` property to specify a relative path to movie clips or objects that are above the current movie clip or object.

You can use `_parent` to move up multiple levels in the display list as in the following:

```
this._parent._parent._alpha = 20;
```

Example

The following example traces the reference to a movie clip and its relationship to the main Timeline. Create a movie clip with the instance name `my_mc`, and add it to the main Timeline. Add the following ActionScript to your FLA or AS file:

```
my_mc.onRelease = function() {
  trace("You clicked the movie clip: "+this);
  trace("The parent of "+this._name+" is: "+this._parent);
}
```

When you click the movie clip, the following information displays in the Output panel.

```
You clicked the movie clip: _level0.my_mc
The parent of my_mc is: _level0
```

See also

`Button._parent`, `_root`, `targetPath()`, `TextField._parent`

MovieClip.play()

Availability

Flash Player 5.

Usage

my_mc.play() *: Void*

Parameters

None.

Returns

Nothing.

Description

Method; moves the playhead in the Timeline of the movie clip.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

Use the following ActionScript to play the main Timeline of a SWF file. This ActionScript is for a movie clip button called `my_mc` on the main Timeline:

```
stop();
my_mc.onRelease = function() {
  this._parent.play();
};
```

Use the following ActionScript to play the Timeline of a movie clip in a SWF file. This ActionScript is for a button called `my_btn` on the main Timeline that plays a movie clip called `animation_mc`:

```
animation_mc.stop();
my_btn.onRelease = function(){
  animation_mc.play();
};
```

See also

`play()`, `MovieClip.gotoAndPlay()`, `gotoAndPlay()`

MovieClip.prevFrame()

Availability

Flash Player 5.

Usage

```
my_mc.prevFrame() : Void
```

Parameters

None.

Returns

Nothing.

Description

Method; sends the playhead to the previous frame and stops it.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

In the following example, two movie clip buttons control the Timeline. The `prev_mc` button moves the playhead to the previous frame, and the `next_mc` button moves the playhead to the next frame. Add content to a series of frames on the Timeline, and add the following ActionScript to Frame 1 of the Timeline:

```
stop();
prev_mc.onRelease = function() {
  var parent_mc:MovieClip = this._parent;
  if (parent_mc._currentframe>1) {
    parent_mc.prevFrame();
  } else {
    parent_mc.gotoAndStop(parent_mc._totalframes);
  }
};
next_mc.onRelease = function() {
  var parent_mc:MovieClip = this._parent;
  if (parent_mc._currentframe<parent_mc._totalframes) {
    parent_mc.nextFrame();
  } else {
    parent_mc.gotoAndStop(1);
  }
};
```

See also

`prevFrame()`

MovieClip._quality

Availability

Flash Player 6.

Usage

my_mc`._quality:`*String*

Description

Property (global); sets or retrieves the rendering quality used for a SWF file. Device fonts are always aliased and therefore are unaffected by the `_quality` property.

The `_quality` property can be set to the following values:

- `"LOW"` Low rendering quality. Graphics are not anti-aliased, and bitmaps are not smoothed.
- `"MEDIUM"` Medium rendering quality. Graphics are anti-aliased using a 2 x 2 pixel grid, but bitmaps are not smoothed. This is suitable for movies that do not contain text.
- `"HIGH"` High rendering quality. Graphics are anti-aliased using a 4 x 4 pixel grid, and bitmaps are smoothed if the movie is static. This is the default rendering quality setting used by Flash.
- `"BEST"` Very high rendering quality. Graphics are anti-aliased using a 4 x 4 pixel grid and bitmaps are always smoothed.

Note: Although you can specify this property for a Movie Clip object, it is actually a global property, and you can specify its value simply as `_quality`. For more information, see `_quality`.

Example

This example sets the rendering quality of a movie clip named `my_mc` to `LOW`:

```
my_mc._quality = "LOW";
```

MovieClip.removeMovieClip()

Availability

Flash Player 5.

Usage

```
my_mc.removeMovieClip() : Void
```

Parameters

None.

Returns

Nothing.

Description

Method; removes a movie clip instance created with `duplicateMovieClip()`, `MovieClip.duplicateMovieClip()`, or `MovieClip.attachMovie()`.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

Each time you click a button in the following example, you attach a movie clip instance to the Stage in a random position. When you click a movie clip instance, you remove that instance from the SWF file.

```
function randRange(min:Number, max:Number):Number {
  var randNum:Number = Math.round(Math.random()*(max-min))+min;
  return randNum;
}
var bugNum:Number = 0;
addBug_btn.onRelease = addBug;
function addBug() {
  var thisBug:MovieClip = this._parent.attachMovie("bug_id",
  "bug"+bugNum+"_mc", bugNum, {_x:randRange(50, 500), _y:randRange(50, 350)});
  thisBug.onRelease = function() {
    this.removeMovieClip();
  };
  bugNum++;
}
```

MovieClip._rotation

Availability

Flash Player 4.

Usage

my_mc._rotation:*Number*

Description

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

Example

The following example creates a movie clip instance dynamically, rotates, and loads an image into the instance.

```
this.createEmptyMovieClip("image_mc", 1);
image_mc._rotation = 15;
image_mc.loadMovie("http://www.macromedia.com/devnet/mx/blueprint/articles/
  nielsen/spotlight_jnielsen.jpg");
```

See also

`Button._rotation`, `TextField._rotation`

MovieClip.setMask()

Availability

Flash Player 6.

Usage

my_mc.setMask(*mask_mc:Object*) : *Void*

Parameters

my_mc The instance name of a movie clip to be masked.

mask_mc The instance name of a movie clip to be a mask.

Returns

Nothing.

Description

Method; makes the movie clip in the parameter *mask_mc* a mask that reveals the movie clip specified by the *my_mc* parameter.

This method allows multiple-frame movie clips with complex, multilayered content to act as masks. You can shut masks on and off at runtime. However, you can't use the same mask for multiple masks (which is possible by using mask layers). If you have device fonts in a masked movie clip, they are drawn but not masked. You can't set a movie clip to be its own mask—for example, `my_mc.setMask(my_mc)`.

If you create a mask layer that contains a movie clip, and then apply the `setMask()` method to it, the `setMask()` call takes priority and this is not reversible. For example, you could have a movie clip in a mask layer called `UIMask` that masks another layer containing another movie clip called `UIMaskee`. If, as the SWF file plays, you call `UIMask.setMask(UIMaskee)`, from that point on, `UIMask` is masked by `UIMaskee`.

To cancel a mask created with ActionScript, pass the value `null` to the `setMask()` method. The following code cancels the mask without affecting the mask layer in the Timeline.

```
UIMask.setMask(null);
```

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following code uses the movie clip `circleMask_mc` to mask the movie clip `theMaskee_mc`:

```
theMaskee_mc.setMask(circleMask_mc);
```

MovieClip._soundbuftime

Availability

Flash Player 6.

Usage

```
my_mc._soundbuftime:Number
```

Description

Property (global); an integer that specifies the number of seconds a sound prebuffers before it starts to stream.

Note: Although you can specify this property for a MovieClip object, it is actually a global property, and you can specify its value simply as `_soundbuftime`. For more information and an example, see _soundbuftime.

MovieClip.startDrag()

Availability

Flash Player 5.

Usage

```
my_mc.startDrag([lock:Boolean, [left:Number, top:Number, right:Number,
  bottom:Number]]) : Void
```

Parameters

`lock` A Boolean value specifying whether the draggable movie clip is locked to the center of the mouse position (`true`), or locked to the point where the user first clicked on the movie clip (`false`). This parameter is optional.

`left`, `top`, `right`, `bottom` Values relative to the coordinates of the movie clip's parent that specify a constraint rectangle for the movie clip. These parameters are optional.

Returns

Nothing.

Description

Method; lets the user drag the specified movie clip. The movie clip remains draggable until explicitly stopped through a call to `MovieClip.stopDrag()`, or until another movie clip is made draggable. Only one movie clip is draggable at a time.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example creates a draggable movie clip instance called `image_mc`. A MovieClipLoader object is used to load an image into `image_mc`.

```
this.createEmptyMovieClip("image_mc", 1);
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  target_mc.onPress = function() {
    this.startDrag();
  };
  target_mc.onRelease = function() {
    this.stopDrag();
  };
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.macromedia.com/devnet/mx/blueprint/articles/
  nielsen/spotlight_jnielsen.jpg", image_mc);
```

See also

`MovieClip._droptarget`, `startDrag()`, `MovieClip.stopDrag()`

MovieClip.stop()

Availability

Flash Player 5.

Usage

`my_mc.stop() : Void`

Parameters

None.

Returns

Nothing.

Description

Method; stops the movie clip currently playing.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example shows how to stop a movie clip named `aMovieClip`:

```
aMovieClip.stop();
```

See also

`stop()`

MovieClip.stopDrag()

Availability

Flash Player 5.

Usage

my_mc.stopDrag() : *Void*

Parameters

None.

Returns

Nothing.

Description

Method; ends a MovieClip.startDrag() method. A movie clip that was made draggable with that method remains draggable until a stopDrag() method is added, or until another movie clip becomes draggable. Only one movie clip is draggable at a time.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example creates a draggable movie clip instance called image_mc. A MovieClipLoader object is used to load an image into image_mc.

```
this.createEmptyMovieClip("image_mc", 1);
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  target_mc.onPress = function() {
    this.startDrag();
  };
  target_mc.onRelease = function() {
    this.stopDrag();
  };
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.macromedia.com/devnet/mx/blueprint/articles/
  nielsen/spotlight_jnielsen.jpg", image_mc);
```

See also

MovieClip._droptarget, MovieClip.startDrag(), stopDrag()

MovieClip.swapDepths()

Availability

Flash Player 5.

Usage

my_mc.swapDepths(*depth:Number*)

my_mc.swapDepths(*target*:String)

Parameters

depth A number specifying the depth level where *my_mc* is to be placed.

target A string specifying the movie clip instance whose depth is swapped by the instance specified by *my_mc*. Both instances must have the same parent movie clip.

Returns

Nothing.

Description

Method; swaps the stacking, or *z*-order (depth level), of the specified instance (*my_mc*) with the movie clip specified by the *target* parameter, or with the movie clip that currently occupies the depth level specified in the *depth* parameter. Both movie clips must have the same parent movie clip. Swapping the depth level of movie clips has the effect of moving one movie clip in front of or behind the other. If a movie clip is tweening when this method is called, the tweening is stopped. For more information, see “Managing movie clip depths” in *Using ActionScript in Flash*.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see “Assigning a class to a movie clip symbol” in *Using ActionScript in Flash*.

Example

The following example swaps the stacking order of two movie clip instances. Overlap two movie clip instances on the Stage, called myMC1_mc and myMC2_mc and then add the following ActionScript to your AS or FLA file:

```
myMC1_mc.onRelease = function() {
  this.swapDepths(myMC2_mc);
};
myMC2_mc.onRelease = function() {
  this.swapDepths(myMC1_mc);
};
```

See also

_level, MovieClip.getDepth(), MovieClip.getInstanceAtDepth(),
  MovieClip.getNextHighestDepth()

MovieClip.tabChildren

Availability

Flash Player 6.

Usage

my_mc.tabChildren:*Boolean*

Description

Property; `undefined` by default. If `tabChildren` is `undefined` or `true`, the children of a movie clip are included in automatic tab ordering. If the value of `tabChildren` is `false`, the children of a movie clip are not included in automatic tab ordering.

Example

A list box UI widget built as a movie clip contains several items. The user can click each item to select it, so each item is a button. However, only the list box itself should be a tab stop. The items inside the list box should be excluded from tab ordering. To do this, the `tabChildren` property of the list box should be set to `false`.

The `tabChildren` property has no effect if the `tabIndex` property is used; the `tabChildren` property affects only automatic tab ordering.

The following example disables tabbing for all children movie clips inside a parent movie clip called `menu_mc`.

```
menu_mc.onRelease = function(){};
menu_mc.menu1_mc.onRelease = function(){};
menu_mc.menu2_mc.onRelease = function(){};
menu_mc.menu3_mc.onRelease = function(){};
menu_mc.menu4_mc.onRelease = function(){};

menu_mc.tabChildren = false;
```

Change the last line of code to the following, in order to include the children movie clip instances of `menu_mc` in the automatic tab ordering.

```
menu_mc.tabChildren = true;
```

See also

`Button.tabIndex`, `MovieClip.tabEnabled`, `MovieClip.tabIndex`, `TextField.tabIndex`

MovieClip.tabEnabled

Availability

Flash Player 6.

Usage

my_mc.tabEnabled:*Boolean*

Description

Property; specifies whether *my_mc* is included in automatic tab ordering. It is `undefined` by default.

If `tabEnabled` is `undefined`, the object is included in automatic tab ordering only if it defines at least one movie clip handler, such as `MovieClip.onRelease`. If `tabEnabled` is `true`, the object is included in automatic tab ordering. If the `tabIndex` property is also set to a value, the object is included in custom tab ordering as well.

If `tabEnabled` is `false`, the object is not included in automatic or custom tab ordering, even if the `tabIndex` property is set. However, if `MovieClip.tabChildren` is `true`, the movie clip's children can still be included in automatic tab ordering, even if `tabEnabled` is `false`.

Example

The following example does not include `myMC2_mc` in the automatic tab ordering.

```
myMC1_mc.onRelease = function() {};
myMC2_mc.onRelease = function() {};
myMC3_mc.onRelease = function() {};
myMC2_mc.tabEnabled = false;
```

See also

`Button.tabEnabled`, `MovieClip.tabChildren`, `MovieClip.tabIndex`,
  `TextField.tabEnabled`

MovieClip.tabIndex

Availability

Flash Player 6.

Usage

my_mc.tabIndex:*Number*

Description

Property; lets you customize the tab ordering of objects in a movie. The `tabIndex` property is `undefined` by default. You can set `tabIndex` on a button, movie clip, or text field instance.

If an object in a SWF file contains a `tabIndex` property, automatic tab ordering is disabled, and the tab ordering is calculated from the `tabIndex` properties of objects in the SWF file. The custom tab ordering includes only objects that have `tabIndex` properties.

The `tabIndex` property must be a positive integer. The objects are ordered according to their `tabIndex` properties, in ascending order. An object with a `tabIndex` value of `1` precedes an object with a `tabIndex` value of `2`. The custom tab ordering disregards the hierarchical relationships of objects in a SWF file. All objects in the SWF file with `tabIndex` properties are placed in the tab order. You shouldn't use the same `tabIndex` value for multiple objects.

Example

The following ActionScript sets a custom tab order for three movie clip instances.

```
myMC1_mc.onRelease = function() {};
myMC2_mc.onRelease = function() {};
myMC3_mc.onRelease = function() {};
myMC1_mc.tabIndex = 2;
myMC2_mc.tabIndex = 1;
myMC3_mc.tabIndex = 3;
```

See also

`Button.tabIndex`, `TextField.tabIndex`

MovieClip._target

Availability

Flash Player 4.

Usage

`my_mc._target:String`

Description

Read-only property; returns the target path of the movie clip instance specified by `my_mc` in slash notation. Use the `eval()` function to convert the target path to dot notation.

Example

The following example displays the target paths of movie clip instances in a SWF file, in both slash and dot notation.

```
for (var i in this) {
  if (typeof (this[i]) == "movieclip") {
    trace("name: "+this[i]._name+",\t target: "+this[i]._target+",\t
  target(2): "+eval(this[i]._target));
  }
}
```

MovieClip._totalframes

Availability

Flash Player 4.

Usage

`my_mc``._totalframes:`*`Number`*

Description

Read-only property; returns the total number of frames in the movie clip instance specified in the *`MovieClip`* parameter.

Example

In the following example, two movie clip buttons control the Timeline. The `prev_mc` button moves the playhead to the previous frame, and the `next_mc` button moves the playhead to the next frame. Add content to a series of frames on the Timeline, and add the following ActionScript to Frame 1 of the Timeline:

```
stop();
prev_mc.onRelease = function() {
  var parent_mc:MovieClip = this._parent;
  if (parent_mc._currentframe>1) {
    parent_mc.prevFrame();
  } else {
    parent_mc.gotoAndStop(parent_mc._totalframes);
  }
};
next_mc.onRelease = function() {
  var parent_mc:MovieClip = this._parent;
  if (parent_mc._currentframe<parent_mc._totalframes) {
    parent_mc.nextFrame();
  } else {
    parent_mc.gotoAndStop(1);
  }
};
```

MovieClip.trackAsMenu

Availability

Flash Player 6.

Usage

`my_mc``.trackAsMenu:`*`Boolean`*

Description

Property; a Boolean value that indicates whether or not other buttons or movie clips can receive mouse release events. This allows you to create menus. You can set the `trackAsMenu` property on any button or movie clip object. If the `trackAsMenu` property does not exist, the default behavior is `false`.

You can change the `trackAsMenu` property at any time; the modified movie clip immediately takes on the new behavior.

Example

The following example sets the `trackAsMenu` property for three movie clips on the Stage. Click a movie clip and release the mouse button on a second movie clip to see which instance receives the event.

```
myMC1_mc.trackAsMenu = true;
myMC2_mc.trackAsMenu = true;
myMC3_mc.trackAsMenu = false;

myMC1_mc.onRelease = clickMC;
myMC2_mc.onRelease = clickMC;
myMC3_mc.onRelease = clickMC;

function clickMC() {
  trace("you clicked the "+this._name+" movie clip.");
}
```

See also

`Button.trackAsMenu`

MovieClip.unloadMovie()

Availability

Flash Player 5.

Usage

`my_mc.unloadMovie() : Void`

Parameters

None.

Returns

Nothing.

Description

Method; removes the contents of a movie clip instance. The instance properties and clip handlers remain.

To remove the instance, including its properties and clip handlers, use `MovieClip.removeMovieClip()`.

You can extend the methods and event handlers of the MovieClip class by creating a subclass. For more information, see "Assigning a class to a movie clip symbol" in *Using ActionScript in Flash*.

Example

The following example unloads a movie clip instance called `image_mc` when a user clicks the `unloadMC_btn` instance.

```
this.createEmptyMovieClip("image_mc", 1);
image_mc.loadMovie("http://www.macromedia.com/images/shared/product_boxes/
  112x112/box_studio_112x112.jpg");
unloadMC_btn.onRelease = function() {
  image_mc.unloadMovie();
};
```

See also

`MovieClip.attachMovie()`, `MovieClip.loadMovie()`, `unloadMovie()`, `unloadMovieNum()`

MovieClip._url

Availability

Flash Player 4.

Usage

my_mc._url*:String*

Description

Read-only property; retrieves the URL of the SWF or JPEG file from which the movie clip was downloaded.

Example

The following example displays the URL of the image that is loaded into the `image_mc` instance in the Output panel.

```
this.createEmptyMovieClip("image_mc", 1);
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  trace("_url: "+target_mc._url);
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.macromedia.com/images/shared/product_boxes/
  112x112/box_studio_112x112.jpg", image_mc);
```

The following example assigns the ContextMenu object `menu_cm` to the movie clip `image_mc`. The ContextMenu object contains a custom menu item labeled “View Image in Browser...” that has an associated function named `viewImage()`.

```
var menu_cm:ContextMenu = new ContextMenu();
menu_cm.customItems.push(new ContextMenuItem("View Image in Browser...",
  viewImage));
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  target_mc.menu = menu_cm;
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("photo1.jpg", image_mc);

function viewImage(target_mc:MovieClip, obj:Object) {
  getURL(target_mc._url, "_blank");
}
```

When you right-click or Control-click the image at runtime, select View Image in Browser from the context menu to open the image in a browser window.

MovieClip.useHandCursor

Availability

Flash Player 6.

Usage

my_mc.useHandCursor:*Boolean*

Description

Property; a Boolean value that indicates whether the hand cursor (pointing hand) appears when the mouse rolls over a movie clip. The default value of useHandCursor is true. If useHandCursor is set to true, the pointing hand used for buttons is displayed when the mouse rolls over a button movie clip. If useHandCursor is false, the arrow pointer is used instead.

You can change the useHandCursor property at any time; the modified movie clip immediately takes on the new cursor behavior. The useHandCursor property can be read out of a prototype object.

Example

The following example sets the useHandCursor property for two movie clips called myMC1_mc and myMC2_mc. The property is set to true for one instance, and false for the other instance. Notice how both instances can still receive events.

```
myMC1_mc.onRelease = traceMC;
myMC2_mc.onRelease = traceMC;
myMC2_mc.useHandCursor = false;

function traceMC() {
  trace("you clicked: "+this._name);
}
```

MovieClip._visible

Availability

Flash Player 4.

Usage

my_mc._visible:*Boolean*

Description

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

Example

The following example sets the _visible property for two movie clips called myMC1_mc and myMC2_mc. The property is set to true for one instance, and false for the other. Notice that myMC1_mc instance cannot be clicked after the _visible property is set to false.

```
myMC1_mc.onRelease = function() {
  trace(this._name+"._visible = false");
  this._visible = false;
};
myMC2_mc.onRelease = function() {
  trace(this._name+"._alpha = 0");
  this._alpha = 0;
};
```

See also

Button._visible, TextField._visible

MovieClip._width

Availability

Flash Player 4 as a read-only property.

Usage

```
my_mc._width:Number
```

Description

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

Example

The following code example displays the height and width of a movie clip in the Output panel:

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var image_mcl:MovieClipLoader = new MovieClipLoader();
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  trace(target_mc._name+" = "+target_mc._width+" X "+target_mc._height+" 
  pixels");
};
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.macromedia.com/devnet/mx/blueprint/articles/
  nielsen/spotlight_jnielsen.jpg", image_mc);
```

See also

`MovieClip._height`

MovieClip._x

Availability

Flash Player 3.

Usage

my_mc._x:*Number*

Description

Property; an integer that sets the *x* coordinate of a movie clip relative to the local coordinates of the parent movie clip. If a movie clip is in the main Timeline, then its coordinate system refers to the upper left corner of the Stage as (0, 0). If the move clip is inside another movie clip that has transformations, the movie clip is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90° counterclockwise, the movie clip's children inherit a coordinate system that is rotated 90° counterclockwise. The movie clip's coordinates refer to the registration point position.

Example

The following example attaches a movie clip with the linkage identifier `cursor_id` to a SWF file. The movie clip is called `cursor_mc`, and it is used to replace the default mouse pointer. The following ActionScript sets the current coordinates of the movie clip instance to the position of the mouse pointer.

```
this.attachMovie("cursor_id", "cursor_mc", this.getNextHighestDepth(),
  {_x:_xmouse, _y:_ymouse});
Mouse.hide();
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
  cursor_mc._x = _xmouse;
  cursor_mc._y = _ymouse;
  updateAfterEvent();
};
Mouse.addListener(mouseListener);
```

See also

`MovieClip._xscale`, `MovieClip._y`, `MovieClip._yscale`

MovieClip._xmouse

Availability

Flash Player 5.

Usage

my_mc._xmouse:*Number*

Description

Read-only property; returns the *x* coordinate of the mouse position.

Example

The following example returns the current *x* and *y* coordinates of the mouse on the Stage (_level0) and in relation to a movie clip on the Stage called my_mc.

```
this.createTextField("mouse_txt", this.getNextHighestDepth, 0, 0, 150, 66);
mouse_txt.html = true;
mouse_txt.multiline = true;
var row1_str:String = "&nbsp;\t<b>_xmouse\t</b><b>_ymouse</b>";
my_mc.onMouseMove = function() {
  mouse_txt.htmlText = "<textformat tabStops='[50,100]'>";
  mouse_txt.htmlText += row1_str;
  mouse_txt.htmlText += "<b>_level0</b>\t"+_xmouse+"\t"+_ymouse;
  mouse_txt.htmlText += "<b>my_mc</b>\t"+this._xmouse+"\t"+this._ymouse;
  mouse_txt.htmlText += "</textformat>";
};
```

See also

Mouse class, MovieClip._ymouse

MovieClip._xscale

Availability

Flash Player 4.

Usage

`my_mc._xscale:Number`

Description

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

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

Example

The following example creates a movie clip at runtime called `box_mc`. The Drawing API is used to draw a box in this instance, and when the mouse rolls over the box, horizontal and vertical scaling is applied to the movie clip. When the mouse rolls off the instance, it returns to the previous scaling.

```
this.createEmptyMovieClip("box_mc", 1);
box_mc._x = 100;
box_mc._y = 100;
with (box_mc) {
  lineStyle(1, 0xCCCCCC);
  beginFill(0xEEEEEE);
  moveTo(0, 0);
  lineTo(80, 0);
  lineTo(80, 60);
  lineTo(0, 60);
  lineTo(0, 0);
  endFill();
}
box_mc.onRollOver = function() {
  this._x -= this._width/2;
  this._y -= this._height/2;
  this._xscale = 200;
  this._yscale = 200;
};
box_mc.onRollOut = function() {
  this._xscale = 100;
  this._yscale = 100;
  this._x += this._width/2;
  this._y += this._height/2;
};
```

See also

`MovieClip._x`, `MovieClip._y`, `MovieClip._yscale`

MovieClip._y

Availability

Flash Player 3.

Usage

my_mc._y:Number

Description

Property; sets the *y* coordinate of a movie clip relative to the local coordinates of the parent movie clip. If a movie clip is in the main Timeline, then its coordinate system refers to the upper left corner of the Stage as (0, 0). If the move clip is inside another movie clip that has transformations, the movie clip is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90° counterclockwise, the movie clip's children inherit a coordinate system that is rotated 90° counterclockwise. The movie clip's coordinates refer to the registration point position.

Example

The following example attaches a movie clip with the linkage identifier `cursor_id` to a SWF file. The movie clip is called `cursor_mc`, and it is used to replace the default mouse pointer. The following ActionScript sets the current coordinates of the movie clip instance to the position of the mouse pointer.

```
this.attachMovie("cursor_id", "cursor_mc", this.getNextHighestDepth(),
  {_x:_xmouse, _y:_ymouse});
Mouse.hide();
var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
  cursor_mc._x = _xmouse;
  cursor_mc._y = _ymouse;
  updateAfterEvent();
};
Mouse.addListener(mouseListener);
```

See also

`MovieClip._x`, `MovieClip._xscale`, `MovieClip._yscale`

MovieClip._ymouse

Availability

Flash Player 5.

Usage

my_mc._ymouse:*Number*

Description

Read-only property; indicates the *y* coordinate of the mouse position.

Example

The following example returns the current *x* and *y* coordinates of the mouse on the Stage (`_level0`) and in relation to a movie clip on the Stage called `my_mc`.

```
this.createTextField("mouse_txt", this.getNextHighestDepth, 0, 0, 150, 66);
mouse_txt.html = true;
mouse_txt.multiline = true;
var row1_str:String = "&nbsp;\t<b>_xmouse\t</b><b>_ymouse</b>";
my_mc.onMouseMove = function() {
  mouse_txt.htmlText = "<textformat tabStops='[50,100]'>";
  mouse_txt.htmlText += row1_str;
  mouse_txt.htmlText += "<b>_level0</b>\t"+_xmouse+"\t"+_ymouse;
  mouse_txt.htmlText += "<b>my_mc</b>\t"+this._xmouse+"\t"+this._ymouse;
  mouse_txt.htmlText += "</textformat>";
};
```

See also

`Mouse class`, `MovieClip._xmouse`

MovieClip._yscale

Availability

Flash Player 4.

Usage

my_mc._yscale:*Number*

Description

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

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

Example

The following example creates a movie clip at runtime called box_mc. The Drawing API is used to draw a box in this instance, and when the mouse rolls over the box, horizontal and vertical scaling is applied to the movie clip. When the mouse rolls off the instance, it returns to the previous scaling.

```
this.createEmptyMovieClip("box_mc", 1);
box_mc._x = 100;
box_mc._y = 100;
with (box_mc) {
  lineStyle(1, 0xCCCCCC);
  beginFill(0xEEEEEE);
  moveTo(0, 0);
  lineTo(80, 0);
  lineTo(80, 60);
  lineTo(0, 60);
  lineTo(0, 0);
  endFill();
}
box_mc.onRollOver = function() {
  this._x -= this._width/2;
  this._y -= this._height/2;
  this._xscale = 200;
  this._yscale = 200;
};
box_mc.onRollOut = function() {
  this._xscale = 100;
  this._yscale = 100;
  this._x += this._width/2;
  this._y += this._height/2;
};
```

See also

MovieClip._x, MovieClip._xscale, MovieClip._y

MovieClipLoader class

Availability

Flash Player 7.

Description

This class lets you implement listener callbacks that provide status information while SWF or JPEG files are being loaded (downloaded) into movie clips. To use MovieClipLoader features, use MovieClipLoader.loadClip() instead of `loadMovie()` or `MovieClip.loadMovie()` to load SWF files.

After you issue the `MovieClipLoader.loadClip()` command, the following events take place in the order listed:

- When the first bytes of the downloaded file have been written to disk, the MovieClipLoader.onLoadStart listener is invoked.
- If you have implemented the MovieClipLoader.onLoadProgress listener, it is invoked during the loading process.

  Note: You can call MovieClipLoader.getProgress() at any time during the load process.

- When the entire downloaded file has been written to disk, the MovieClipLoader.onLoadComplete listener is invoked.
- After the downloaded file's first frame actions have been executed, the MovieClipLoader.onLoadInit listener is invoked.

After `MovieClipLoader.onLoadInit` has been invoked, you can set properties, use methods, and otherwise interact with the loaded movie.

If the file fails to load completely, the MovieClipLoader.onLoadError listener is invoked.

Method summary for the MovieClipLoader class

Method	Description
`MovieClipLoader.addListener()`	Registers an object to receive notification when a MovieClipLoader event handler is invoked.
`MovieClipLoader.getProgress()`	Returns the number of bytes loaded and total number of bytes for a file that is being loaded using `MovieClipLoader.loadClip()`.
`MovieClipLoader.loadClip()`	Loads a SWF or JPEG file into a movie clip in Flash Player while the original movie is playing.
`MovieClipLoader.removeListener()`	Deletes an object that was registered using `MovieClipLoader.addListener()`.
`MovieClipLoader.unloadClip()`	Removes a movie clip that was loaded by means of `MovieClipLoader.loadClip()`.

Listener summary for the MovieClipLoader class

Listener	Description
`MovieClipLoader.onLoadComplete`	Invoked when a file loaded with `MovieClipLoader.loadClip()` has completely downloaded.
`MovieClipLoader.onLoadError`	Invoked when a file loaded with `MovieClipLoader.loadClip()` has failed to load.
`MovieClipLoader.onLoadInit`	Invoked when the actions on the first frame of the loaded clip have been executed.
`MovieClipLoader.onLoadProgress`	Invoked every time the loading content is written to disk during the loading process.
`MovieClipLoader.onLoadStart`	Invoked when a call to `MovieClipLoader.loadClip()` has successfully begun to download a file.

Constructor for the MovieClipLoader class

Availability

Flash Player 7.

Usage

```
new MovieClipLoader() : MovieClipLoader
```

Parameters

None.

Returns

A reference to a MovieClipLoader object.

Description

Constructor; creates a MovieClipLoader object that you can use to implement a number of listeners to respond to events while a SWF or JPEG file is downloading.

Example

See MovieClipLoader.loadClip().

See also

`MovieClipLoader.addListener()`

MovieClipLoader.addListener()

Availability

Flash Player 7.

Usage

```
my_mcl.addListener(listenerObject:Object) : Void
```

Parameters

`listenerObject` An object that listens for a callback notification from the MovieClipLoader event handlers.

Returns

Nothing.

Description

Method; registers an object to receive notification when a MovieClipLoader event handler is invoked.

Example

The following example loads an image into a movie clip called `image_mc`. The movie clip instance is rotated and centered on the Stage, and both the Stage and movie clip have a stroke drawn around their perimeters.

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  target_mc._x = Stage.width/2-target_mc._width/2;
  target_mc._y = Stage.height/2-target_mc._width/2;
  var w:Number = target_mc._width;
  var h:Number = target_mc._height;
  target_mc.lineStyle(4, 0x000000);
  target_mc.moveTo(0, 0);
  target_mc.lineTo(w, 0);
  target_mc.lineTo(w, h);
  target_mc.lineTo(0, h);
  target_mc.lineTo(0, 0);
  target_mc._rotation = 3;
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.macromedia.com/images/shared/product_boxes/
  112x112/box_studio_112x112.jpg", image_mc);
```

See also

`MovieClipLoader.onLoadComplete`, `MovieClipLoader.onLoadError`, `MovieClipLoader.onLoadInit`, `MovieClipLoader.onLoadProgress`, `MovieClipLoader.onLoadStart`, `MovieClipLoader.removeListener()`

MovieClipLoader.getProgress()

Availability

Flash Player 7.

Usage

my_mcl.getProgress(*target_mc*:*Object*) : *Object*

Parameters

target_mc A SWF or JPEG file that is loaded using MovieClipLoader.loadClip().

Returns

An object that has two integer properties: `bytesLoaded` and `bytesTotal`.

Description

Method; returns the number of bytes loaded and total number of bytes for a file that is being loaded using MovieClipLoader.loadClip(); for compressed movies, it reflects the number of compressed bytes. This method lets you explicitly request this information, instead of (or in addition to) writing a MovieClipLoader.onLoadProgress listener function.

Example

The following example loads an image into a draggable, dynamically created movie clip called `image_mc`. The number of bytes loaded and the total number of bytes for the loaded image display in a dynamically created text field called `filesize_txt`.

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  target_mc.onPress = function() {
    this.startDrag();
  };
  target_mc.onRelease = function() {
    this.stopDrag();
  };
  var mclProgress:Object = image_mcl.getProgress(target_mc);
  target_mc.createTextField("filesize_txt", target_mc.getNextHighestDepth(),
  0, target_mc._height, target_mc._width, 22);
  target_mc.filesize_txt.text = mclProgress.bytesLoaded+" of
  "+mclProgress.bytesTotal+" bytes loaded";
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.macromedia.com/images/shared/product_boxes/
  112x112/box_studio_112x112.jpg", image_mc);
```

See also

`MovieClipLoader.onLoadProgress`

MovieClipLoader.loadClip()

Availability

Flash Player 7.

Usage

`my_mcl.loadClip(url:String, target:Object ) : Boolean`

Parameters

`url` The absolute or relative URL of the SWF file or JPEG file to be loaded. A relative path must be relative to the SWF file at level 0. Absolute URLs must include the protocol reference, such as http:// or file:///. Filenames cannot include disk drive specifications.

`target` The target path of a movie clip, or an integer specifying the level in Flash Player into which the movie will be loaded. The target movie clip is replaced by the loaded SWF file or image.

Returns

A Boolean value. The return value is `true` if the URL request was sent successfully; otherwise the return value is `false`.

Description

Method; loads a SWF or JPEG file into a movie clip in Flash Player while the original movie is playing. Using this method lets you display several SWF files at once and switch between SWF files without loading another HTML document.

Using this method instead of `loadMovie()` or `MovieClip.loadMovie()` has a number of advantages. The following handlers are implemented by the use on a listener object, which is registered with the MovieClipLoader class using `MovieClipLoader.addListener(listenerObject)`.

- The MovieClipLoader.onLoadStart handler is invoked when loading begins.
- The MovieClipLoader.onLoadError handler is invoked if the clip cannot be loaded.
- The MovieClipLoader.onLoadProgress handler is invoked as the loading process progresses.
- The MovieClipLoader.onLoadInit handler is invoked after the actions in the first frame of the clip have executed, so you can begin manipulating the loaded clip.
- The MovieClipLoader.onLoadComplete handler is invoked when a file has completed downloading.

A SWF file or image loaded into a movie clip inherits the position, rotation, and scale properties of the movie clip. You can use the target path of the movie clip to target the loaded movie.

You can use the `loadClip()` method to load one or more files into a single movie clip or level; MovieClipLoader listener objects are passed the loading target movie clip instance as a parameter. Alternately, you can create a different MovieClipLoader object for each file you load.

Use `MovieClipLoader.unloadClip()` to remove movies or images loaded with this method or to cancel a load operation that is in progress.

`MovieClipLoader.getProgress()` and `MovieClipLoaderListener.onLoadProgress` do not report the actual `bytesLoaded` and `bytesTotal` values in the Authoring player when the files are local. When you use the Bandwidth Profiler feature in the authoring environment, `MovieClipLoader.getProgress()` and `MovieClipLoaderListener.onLoadProgress` report the download at the actual download rate, not at the reduced bandwidth rate that the Bandwidth Profiler provides.

Example

The following example illustrates the use of many of the MovieClipLoader class methods and listeners:

```
// first set of listeners
var my_mcl:MovieClipLoader = new MovieClipLoader();
var myListener:Object = new Object();
myListener.onLoadStart = function(target_mc:MovieClip) {
  trace("*********First my_mcl instance*********");
  trace("Your load has begun on movie clip = "+target_mc);
  var loadProgress:Object = my_mcl.getProgress(target_mc);
  trace(loadProgress.bytesLoaded+" = bytes loaded at start");
  trace(loadProgress.bytesTotal+" = bytes total at start");
};
myListener.onLoadProgress = function(target_mc:MovieClip, loadedBytes:Number,
  totalBytes:Number) {
  trace("*********First my_mcl instance Progress*********");
  trace("onLoadProgress() called back on movie clip "+target_mc);
  trace(loadedBytes+" = bytes loaded at progress callback");
  trace(totalBytes+" = bytes total at progress callback");
};
myListener.onLoadComplete = function(target_mc:MovieClip) {
  trace("*********First my_mcl instance*********");
  trace("Your load is done on movie clip = "+target_mc);
  var loadProgress:Object = my_mcl.getProgress(target_mc);
  trace(loadProgress.bytesLoaded+" = bytes loaded at end");
  trace(loadProgress.bytesTotal+" = bytes total at end");
};
myListener.onLoadInit = function(target_mc:MovieClip) {
  trace("*********First my_mcl instance*********");
  trace("Movie clip = "+target_mc+" is now initialized");
  // you can now do any setup required, for example:
  target_mc._width = 100;
  target_mc._height = 100;
};
myListener.onLoadError = function(target_mc:MovieClip, errorCode:String) {
  trace("*********First my_mcl instance*********");
  trace("ERROR CODE = "+errorCode);
  trace("Your load failed on movie clip = "+target_mc+"\n");
};
my_mcl.addListener(myListener);
// Now load the files into their targets.
// loads into movie clips
this.createEmptyMovieClip("clip1_mc", this.getNextHighestDepth());
clip1_mc._x = 400;
this.createEmptyMovieClip("clip2_mc", this.getNextHighestDepth());
```

```
my_mcl.loadClip("http://www.macromedia.com/software/drk/images/box_drk5.jpg",
  clip1_mc);
my_mcl.loadClip("http://www.macromedia.com/devnet/images/160x160/
  ben_forta.jpg", clip2_mc);
clip2_mc._x = 200;
// loads into _level1
my_mcl.loadClip("http://www.macromedia.com/devnet/images/160x160/
  mike_chambers.jpg", 1);
//
// Second set of listeners
var another_mcl:MovieClipLoader = new MovieClipLoader();
var myListener2:Object = new Object();
myListener2.onLoadStart = function(target_mc:MovieClip) {
  trace("*********Second my_mcl instance*********");
  trace("Your load has begun on movie = "+target_mc);
  var loadProgress:Object = my_mcl.getProgress(target_mc);
  trace(loadProgress.bytesLoaded+" = bytes loaded at start");
  trace(loadProgress.bytesTotal+" = bytes total at start");
};
myListener2.onLoadComplete = function(target_mc:MovieClip) {
  trace("*********Second my_mcl instance*********");
  trace("Your load is done on movie clip = "+target_mc);
  var loadProgress:Object = my_mcl.getProgress(target_mc);
  trace(loadProgress.bytesLoaded+" = bytes loaded at end");
  trace(loadProgress.bytesTotal+" = bytes total at end");
};
myListener2.onLoadError = function(target_mc:MovieClip, errorCode:String) {
  trace("*********Second my_mcl instance*********");
  trace("ERROR CODE = "+errorCode);
  trace("Your load failed on movie clip = "+target_mc+"\n");
};
another_mcl.addListener(myListener2);
/* Now load the files into their targets (using the second instance of
  MovieClipLoader) */
another_mcl.loadClip("http://www.macromedia.com/devnet/images/160x160/
  flex_logo.jpg", this.createEmptyMovieClip("clip4_mc",
  this.getNextHighestDepth()));
clip4_mc._y = 200;
// Issue the following statements after the download is complete,
// and after my_mcl.onLoadInit has been called.
// my_mcl.removeListener(myListener);
// my_mcl.removeListener(myListener2);
```

See also

`MovieClipLoader.unloadClip()`

MovieClipLoader.onLoadComplete

Availability

Flash Player 7.

Usage

```
listenerObject.onLoadComplete = function([target_mc:Object]) {
  // your statements here
}
```

Parameters

`listenerObject` A listener object that was added using MovieClipLoader.addListener().

`target_mc` A movie clip loaded by a MovieClipLoader.loadClip() method. This parameter is optional.

Returns

Nothing.

Description

Listener; invoked when a file loaded with MovieClipLoader.loadClip() has completely downloaded. The value for `target_mc` identifies the movie clip for which this call is being made. This is useful if multiple files are being loaded with the same set of listeners.

This parameter is passed by Flash to your code, but you do not have to implement all of the parameters in the listener function.

When you use the `onLoadComplete` and `onLoadInit` events with the MovieClipLoader class, it's important to understand how this differs from the way they work with your SWF file. The `onLoadComplete` event is called after the SWF or JPEG file has loaded, but before the application has been initialized. At this point it is impossible to access the loaded movie clip's methods and properties, and because of this you cannot call a function, move to a specific frame, and so on. In most situations, it's better to use the `onLoadInit` event instead, which is called after the content has loaded and is fully initialized.

Example

The following example loads an image into a movie clip instance called `image_mc`. The `onLoadInit` and `onLoadComplete` events are used to determine how long it takes to load the image. The information displays in a dynamically created text field called `timer_txt`.

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadStart = function(target_mc:MovieClip) {
  target_mc.startTimer = getTimer();
};
mclListener.onLoadComplete = function(target_mc:MovieClip) {
  target_mc.completeTimer = getTimer();
};
mclListener.onLoadInit = function(target_mc:MovieClip) {
  var timerMS:Number = target_mc.completeTimer-target_mc.startTimer;
```

```
    target_mc.createTextField("timer_txt", target_mc.getNextHighestDepth(), 0,
    target_mc._height, target_mc._width, 22);
    target_mc.timer_txt.text = "loaded in "+timerMS+" ms.";
  };
  var image_mcl:MovieClipLoader = new MovieClipLoader();
  image_mcl.addListener(mclListener);
  image_mcl.loadClip("http://www.macromedia.com/images/shared/product_boxes/
    112x112/box_studio_112x112.jpg", image_mc);
```

See also

`MovieClipLoader.addListener()`, `MovieClipLoader.onLoadStart`, `MovieClipLoader.onLoadError`

MovieClipLoader.onLoadError

Availability

Flash Player 7.

Usage

```
listenerObject.onLoadError = function(target_mc:Object, errorCode:String) {
  // your statements here
}
```

Parameters

listenerObject A listener object that was added using MovieClipLoader.addListener().

target_mc A movie clip loaded by a MovieClipLoader.loadClip() method.

errorCode A string that explains the reason for the failure.

Returns

One of two strings: `"URLNotFound"` or `"LoadNeverCompleted"`.

Description

Listener; invoked when a file loaded with MovieClipLoader.loadClip() has failed to load.

The string `"URLNotFound"` is returned if neither the MovieClipLoader.onLoadStart or MovieClipLoader.onLoadComplete listener has been called. For example, if a server is down or the file is not found, these listeners are not called.

The string `"LoadNeverCompleted"` is returned if `MovieClipLoader.onLoadStart` was called but `MovieClipLoader.onLoadComplete` was not called. For example, if `MovieClipLoader.onLoadStart` is called but the download is interrupted due to server overload, server crash, and so on, `MovieClipLoader.onLoadComplete` will not be called.

The value for *target_mc* identifies the movie clip this call is being made for. This is useful if you are loading multiple files with the same set of listeners. This optional parameter is passed to your ActionScript.

Example

The following example displays information in the Output panel when an image fails to load. This occurs when you test the following ActionScript, because the image does not exist in the specified location.

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadError = function(target_mc:MovieClip, errorCode:String) {
  trace("ERROR!");
  switch (errorCode) {
  case 'URLNotFound' :
    trace("\t Unable to connect to URL: "+target_mc._url);
    break;
  case 'LoadNeverCompleted' :
    trace("\t Unable to complete download: "+target_mc);
    break;
    }
};
mclListener.onLoadInit = function(target_mc:MovieClip) {
    trace("success");
    trace(image_mcl.getProgress(target_mc).bytesTotal+" bytes loaded");
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.fakedomain.com/images/bad_hair_day.jpg",
    image_mc);
```

MovieClipLoader.onLoadInit

Availability

Flash Player 7.

Usage

```
listenerObject.onLoadInit = function([target_mc:MovieClip]) {
  // your statements here
}
```

Parameters

listenerObject A listener object that was added using MovieClipLoader.addListener().

target_mc A movie clip loaded by a MovieClipLoader.loadClip() method. This parameter is optional.

Returns

Nothing.

Description

Listener; invoked when the actions on the first frame of the loaded clip have been executed. After this listener has been invoked, you can set properties, use methods, and otherwise interact with the loaded movie.

The value for *target_mc* identifies the movie clip this call is being made for. This is useful if you are loading multiple files with the same set of listeners. This optional parameter is passed to your ActionScript.

Example

The following example loads an image into a movie clip instance called `image_mc`. The `onLoadInit` and `onLoadComplete` events are used to determine how long it takes to load the image. This information displays in a text field called `timer_txt`.

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadStart = function(target_mc:MovieClip) {
  target_mc.startTimer = getTimer();
};
mclListener.onLoadComplete = function(target_mc:MovieClip) {
  target_mc.completeTimer = getTimer();
};
mclListener.onLoadInit = function(target_mc:MovieClip) {
  var timerMS:Number = target_mc.completeTimer-target_mc.startTimer;
  target_mc.createTextField("timer_txt", target_mc.getNextHighestDepth(), 0,
  target_mc._height, target_mc._width, 22);
  target_mc.timer_txt.text = "loaded in "+timerMS+" ms.";
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.macromedia.com/images/shared/product_boxes/
  112x112/box_studio_112x112.jpg", image_mc);
```

See also

`MovieClipLoader.onLoadStart`

MovieClipLoader.onLoadProgress

Availability

Flash Player 7.

Usage

```
listenerObject.onLoadProgress =
  function([target_mc:Object [, loadedBytes:Number [, totalBytes:Number ] ] ]
  ) {
  // your statements here
}
```

Parameters

listenerObject A listener object that was added using MovieClipLoader.addListener().

target_mc A movie clip loaded by a MovieClipLoader.loadClip() method. This parameter is optional.

loadedBytes The number of bytes that had been loaded when the listener was invoked.

totalBytes The total number of bytes in the file being loaded.

Returns

Nothing.

Description

Listener; invoked every time the loading content is written to disk during the loading process (that is, between `MovieClipLoader.onLoadStart` and `MovieClipLoader.onLoadComplete`). You can use this method to display information about the progress of the download, using the `loadedBytes` and `totalBytes` parameters.

The value for *target_mc* identifies the movie clip this call is being made for. This is useful if you are loading multiple files with the same set of listeners. This optional parameter is passed to your ActionScript.

Example

The following example creates a progress bar using the Drawing API. The progress bar displays the loading progress of an image using the `onLoadProgress` listener. When the image finishes loading, the progress bar is removed from the Stage. You must replace the URL parameter of the `image_mcl.loadClip()` command so that the parameter refers to a valid JPEG file using HTTP. If you attempt to use this example to load a local file that resides on your hard disk, this example will not work properly because, in test movie mode, Flash Player loads local files in their entirety. Add the following ActionScript to your FLA or AS file:

```
this.createEmptyMovieClip("progressBar_mc", 0);
progressBar_mc.createEmptyMovieClip("bar_mc", 1);
progressBar_mc.createEmptyMovieClip("stroke_mc", 2);
with (progressBar_mc.stroke_mc) {
  lineStyle(0, 0x000000);
  moveTo(0, 0);
  lineTo(100, 0);
```

```
    lineTo(100, 10);
    lineTo(0, 10);
    lineTo(0, 0);
  }
  with (progressBar_mc.bar_mc) {
    beginFill(0xFF0000, 100);
    moveTo(0, 0);
    lineTo(100, 0);
    lineTo(100, 10);
    lineTo(0, 10);
    lineTo(0, 0);
    endFill();
    _xscale = 0;
  }
  progressBar_mc._x = 2;
  progressBar_mc._y = 2;
  //
  var mclListener:Object = new Object();
  mclListener.onLoadStart = function(target_mc:MovieClip) {
    progressBar_mc.bar_mc._xscale = 0;
  };
  mclListener.onLoadProgress = function(target_mc:MovieClip, bytesLoaded:Number,
    bytesTotal:Number) {
    progressBar_mc.bar_mc._xscale = Math.round(bytesLoaded/bytesTotal*100);
  };
  mclListener.onLoadComplete = function(target_mc:MovieClip) {
    progressBar_mc.removeMovieClip();
  };
  mclListener.onLoadInit = function(target_mc:MovieClip) {
    target_mc._height = 320;
    target_mc._width = 240;
  };
  this.createEmptyMovieClip("image_mc", 100);
  var image_mcl:MovieClipLoader = new MovieClipLoader();
  image_mcl.addListener(mclListener);
  image_mcl.loadClip("[place a valid URL pointing to a JPEG file here]",
    image_mc);
```

See also

`MovieClipLoader.getProgress()`

MovieClipLoader.onLoadStart

Availability

Flash Player 7.

Usage

```
listenerObject.onLoadStart = function([target_mc:Object]) {
  // your statements here
}
```

Parameters

`listenerObject` A listener object that was added using MovieClipLoader.addListener().

`target_mc` A movie clip loaded by a MovieClipLoader.loadClip() method. This parameter is optional.

Returns

Nothing.

Description

Listener; invoked when a call to MovieClipLoader.loadClip() has successfully begun to download a file. The value for `target_mc` identifies the movie clip this call is being made for. This is useful if you are loading multiple files with the same set of listeners. This optional parameter is passed to your ActionScript.

Example

The following example loads an image into a movie clip instance called `image_mc`. The `onLoadInit` and `onLoadComplete` events are used to determine how long it takes to load the image. This information displays in a text field called `timer_txt`.

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadStart = function(target_mc:MovieClip) {
  target_mc.startTimer = getTimer();
};
mclListener.onLoadComplete = function(target_mc:MovieClip) {
  target_mc.completeTimer = getTimer();
};
mclListener.onLoadInit = function(target_mc:MovieClip) {
  var timerMS:Number = target_mc.completeTimer-target_mc.startTimer;
  target_mc.createTextField("timer_txt", target_mc.getNextHighestDepth(), 0,
  target_mc._height, target_mc._width, 22);
  target_mc.timer_txt.text = "loaded in "+timerMS+" ms.";
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.macromedia.com/images/shared/product_boxes/
  112x112/box_studio_112x112.jpg", image_mc);
```

See also

`MovieClipLoader.onLoadError`, `MovieClipLoader.onLoadInit`, `MovieClipLoader.onLoadComplete`

MovieClipLoader.removeListener()

Availability

Flash Player 7.

Usage

`my_mcl.removeListener(listenerObject:Object) : Void`

Parameters

`listenerObject` A listener object that was added using MovieClipLoader.addListener().

Returns

Nothing.

Description

Method; removes the listener that was used to receive notification when a `MovieClipLoader` event handler was invoked. No further loading messages will be received.

Example

The following example loads an image into a movie clip, and enables the user to start and stop the loading process using two buttons called `start_button` and `stop_button`. When the user starts or stops the progress, information is displayed in the Output panel.

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadStart = function(target_mc:MovieClip) {
  trace("\t onLoadStart");
};
mclListener.onLoadComplete = function(target_mc:MovieClip) {
  trace("\t onLoadComplete");
};
mclListener.onLoadError = function(target_mc:MovieClip, errorCode:String) {
  trace("\t onLoadError: "+errorCode);
};
mclListener.onLoadInit = function(target_mc:MovieClip) {
  trace("\t onLoadInit");
  start_button.enabled = true;
  stop_button.enabled = false;
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
//
start_button.clickHandler = function() {
  trace("Starting...");
  start_button.enabled = false;
  stop_button.enabled = true;
  //
  image_mcl.addListener(mclListener);
  image_mcl.loadClip("http://www.macromedia.com/devnet/images/160x160/
  flex_logo.jpg", image_mc);
};
stop_button.clickHandler = function() {
  trace("Stopping...");
```

```
    start_button.enabled = true;
    stop_button.enabled = false;
    //
    image_mcl.removeListener(mclListener);
};
stop_button.enabled = false;
```

MovieClipLoader.unloadClip()

Availability

Flash Player 7.

Usage

`my_mcl.unloadClip(target:Object)`

Parameters

`target` The string or integer passed to the corresponding call to `my_mcl.loadClip()`.

Returns

Nothing.

Description

Method; removes a movie clip that was loaded by means of MovieClipLoader.loadClip(). If you issue this command while a movie is loading, MovieClipLoader.onLoadError is invoked.

Example

The following example loads an image into a movie clip called `image_mc`. If you click the movie clip, the movie clip is removed and information is displayed in the Output panel.

```
this.createEmptyMovieClip("image_mc", this.getNextHighestDepth());
var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  target_mc._x = 100;
  target_mc._y = 100;
  target_mc.onRelease = function() {
    trace("Unloading clip...");
    trace("\t name: "+target_mc._name);
    trace("\t url:  "+target_mc._url);
    image_mcl.unloadClip(target_mc);
  };
};
var image_mcl:MovieClipLoader = new MovieClipLoader();
image_mcl.addListener(mclListener);
image_mcl.loadClip("http://www.macromedia.com/devnet/images/160x160/
  flex_logo.jpg", image_mc);
```

See also

`MovieClipLoader.loadClip()`

NaN

Availability

Flash Player 5.

Usage

`NaN`

Description

Variable; a predefined variable with the IEEE-754 value for `NaN` (not a number). To determine if a number is `NaN`, use `isNaN()`.

See also

`isNaN()`, `Number.NaN`

-Infinity

Availability

Flash Player 5.

Usage

```
-Infinity
```

Description

Constant; specifies the IEEE-754 value representing negative infinity. The value of this constant is the same as `Number.NEGATIVE_INFINITY`.

NetConnection class

Availability

Flash Player 7.

Note: This class is also supported in Flash Player 6 when used with Flash Communication Server. For more information, see the Flash Communication Server documentation.

Description

The NetConnection class provides the means to play back streaming FLV files from a local drive or HTTP address. For more information on video playback, see “Playing back external FLV files dynamically” in *Using ActionScript in Flash*.

Method summary for the NetConnection class

Method	Description
`NetConnection.connect()`	Opens a local connection through which you can play back video (FLV) files from an HTTP address or from the local file system.

Constructor for the NetConnection class

Availability

Flash Player 7.

Note: This class is also supported in Flash Player 6 when used with Flash Communication Server. For more information, see your Flash Communication Server documentation.

Usage

```
new NetConnection() : NetConnection
```

Parameters

None.

Returns

A reference to a NetConnection object.

Description

Constructor; creates a NetConnection object that you can use in conjunction with a NetStream object to play back local streaming video (FLV) files. After creating the NetConnection object, use NetConnection.connect() to make the actual connection.

Playing external FLV files provides several advantages over embedding video in a Flash document, such as better performance and memory management, and independent video and Flash frame rates. For more information on video playback, see “Playing back external FLV files dynamically” in *Using ActionScript in Flash*.

Example

See the example for NetConnection.connect().

See also

`NetStream class`, `Video.attachVideo()`

NetConnection.connect()

Availability

Flash Player 7.

Note: This method is also supported in Flash Player 6 when used with Flash Communication Server. For more information, see the Flash Communication Server documentation.

Usage

```
my_nc.connect(null) : Void
```

Parameters

None (you must pass `null`).

Returns

Nothing.

Description

Constructor; opens a local connection through which you can play back video (FLV) files from an HTTP address or from the local file system.

Example

The following example opens a connection to play the video2.flv file. Select New Video from the Library panel's options menu to create a new video object, and give it the instance name `my_video`.

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video2.flv");
```

See also

NetStream class

NetStream class

Availability

Flash Player 7.

Note: This class is also supported in Flash Player 6 when used with Flash Communication Server. For more information, see the Flash Communication Server documentation.

Description

The NetStream class provides methods and properties for playing Flash Video (FLV) files from the local file system or an HTTP address. You use a NetStream object to stream video through a NetConnection object. Playing external FLV files provides several advantages over embedding video in a Flash document, such as better performance and memory management, and independent video and Flash frame rates. This class provides a number of methods and properties you can use to track the progress of the file as it loads and plays, and to give the user control over playback (stopping, pausing, and so on).

For more information on video playback, see "Playing back external FLV files dynamically" in *Using ActionScript in Flash*.

Method summary for the NetStream class

The following methods and properties of the NetConnection and NetStream classes are used to control FLV playback.

Method	Purpose
`NetStream.close()`	Closes the stream but does not clear the video object.
`NetStream.pause()`	Pauses or resumes playback of a stream.
`NetStream.play()`	Begins playback of an external video (FLV) file.
`NetStream.seek()`	Seeks a specific position in the FLV file.
`NetStream.setBufferTime()`	Specifies how long to buffer data before starting to display the stream.

Property summary for the NetStream class

Property	Description
`NetStream.bufferLength`	The number of seconds of data currently in the buffer.
`NetStream.bufferTime`	Read-only; the number of seconds assigned to the buffer by `NetStream.setBufferTime()`.
`NetStream.bytesLoaded`	Read-only; the number of bytes of data that have been loaded into the player.
`NetStream.bytesTotal`	Read-only; the total size in bytes of the file being loaded into the player.
`NetStream.currentFps`	The number of frames per second being displayed.
`NetStream.time`	Read-only; the position of the playhead, in seconds.

Event handler summary for the NetStream class

Event handler	Description
`NetStream.onStatus`	Invoked every time a status change or error is posted for the NetStream object.

Constructor for the NetStream class

Availability

Flash Player 7.

Note: This class is also supported in Flash Player 6 when used with Flash Communication Server. For more information, see the Flash Communication Server documentation.

Usage

```
new NetStream(my_nc:NetConnection) : NetStream
```

Parameters

my_nc	A NetConnection object.

Returns

A reference to a NetStream object.

Description

Constructor; creates a stream that can be used for playing FLV files through the specified NetConnection object.

Example

The following code first constructs a new NetConnection object, `connection_nc`, and uses it to construct a new NetStream object called `stream_ns`. Select New Video from the Library options menu to create a video object instance, and give it an instance name `my_video`.Then add the following ActionScript to your FLA or AS file:

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");
```

See also

NetConnection class, NetStream class, Video.attachVideo()

NetStream.bufferLength

Availability

Flash Player 7.

Note: This property is also supported in Flash Player 6 when used with Flash Communication Server. For more information, see the Flash Communication Server documentation.

Usage

```
my_ns.bufferLength:Number
```

Description

Read-only property; the number of seconds of data currently in the buffer. You can use this property in conjunction with NetStream.bufferTime to estimate how close the buffer is to being full—for example, to display feedback to a user who is waiting for data to be loaded into the buffer.

Example

The following example dynamically creates a text field that displays information about the number of seconds that are currently in the buffer. The text field also displays the buffer length that the video is set to, and percentage of buffer that is filled.

```
this.createTextField("buffer_txt", this.getNextHighestDepth(), 10, 10, 300, 
  22);
buffer_txt.html = true;

var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
stream_ns.setBufferTime(3);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

var buffer_interval:Number = setInterval(checkBufferTime, 100, stream_ns);
function checkBufferTime(my_ns:NetStream):Void {
  var bufferPct:Number = Math.min(Math.round(my_ns.bufferLength/
  my_ns.bufferTime*100), 100);
  var output_str:String = "<textformat tabStops='[100,200]'>";
  output_str += "Length: "+my_ns.bufferLength+"\t"+"Time: 
  "+my_ns.bufferTime+"\t"+"Buffer:"+bufferPct+"%";
  output_str += "</textformat>";
  buffer_txt.htmlText = output_str;
}
```

See also

`NetStream.bytesLoaded`

NetStream.bufferTime

Availability

Flash Player 7.

Note: This property is also supported in Flash Player 6 when used with Flash Communication Server. For more information, see the Flash Communication Server documentation.

Usage

```
my_ns.bufferTime:Number
```

Description

Read-only property; the number of seconds assigned to the buffer by NetStream.setBufferTime(). The default value is .1(one-tenth of a second). To determine the number of seconds currently in the buffer, use NetStream.bufferLength.

Example

The following example dynamically creates a text field that displays information about the number of seconds that are currently in the buffer. The text field also displays the buffer length that the video is set to, and percentage of buffer that is filled.

```
this.createTextField("buffer_txt", this.getNextHighestDepth(), 10, 10, 300,
  22);
buffer_txt.html = true;

var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
stream_ns.setBufferTime(3);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

var buffer_interval:Number = setInterval(checkBufferTime, 100, stream_ns);
function checkBufferTime(my_ns:NetStream):Void {
  var bufferPct:Number = Math.min(Math.round(my_ns.bufferLength/
  my_ns.bufferTime*100), 100);
  var output_str:String = "<textformat tabStops='[100,200]'>";
  output_str += "Length: "+my_ns.bufferLength+"\t"+"Time:
  "+my_ns.bufferTime+"\t"+"Buffer:"+bufferPct+"%";
  output_str += "</textformat>";
  buffer_txt.htmlText = output_str;
}
```

See also

`NetStream.time`

NetStream.bytesLoaded

Availability

Flash Player 7.

Usage

my_ns.bytesLoaded:*Number*

Description

Read-only property; the number of bytes of data that have been loaded into the player. You can use this method in conjunction with NetStream.bytesTotal to estimate how close the buffer is to being full—for example, to display feedback to a user who is waiting for data to be loaded into the buffer.

Example

The following example creates a progress bar using the Drawing API and the `bytesLoaded` and `bytesTotal` properties that displays the loading progress of video1.flv into the video object instance called `my_video`. A text field called `loaded_txt` is dynamically created to display information about the loading progress as well.

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

this.createTextField("loaded_txt", this.getNextHighestDepth(), 10, 10, 160,
  22);
this.createEmptyMovieClip("progressBar_mc", this.getNextHighestDepth());
progressBar_mc.createEmptyMovieClip("bar_mc",
  progressBar_mc.getNextHighestDepth());
with (progressBar_mc.bar_mc) {
  beginFill(0xFF0000);
  moveTo(0, 0);
  lineTo(100, 0);
  lineTo(100, 10);
  lineTo(0, 10);
  lineTo(0, 0);
  endFill();
  _xscale = 0;
}
progressBar_mc.createEmptyMovieClip("stroke_mc",
  progressBar_mc.getNextHighestDepth());
with (progressBar_mc.stroke_mc) {
  lineStyle(0, 0x000000);
  moveTo(0, 0);
  lineTo(100, 0);
  lineTo(100, 10);
  lineTo(0, 10);
  lineTo(0, 0);
}
```

```
var loaded_interval:Number = setInterval(checkBytesLoaded, 500, stream_ns);
function checkBytesLoaded(my_ns:NetStream) {
  var pctLoaded:Number = Math.round(my_ns.bytesLoaded/my_ns.bytesTotal*100);
  loaded_txt.text = Math.round(my_ns.bytesLoaded/1000)+" of
  "+Math.round(my_ns.bytesTotal/1000)+" KB loaded ("+pctLoaded+"%)";
  progressBar_mc.bar_mc._xscale = pctLoaded;
  if (pctLoaded>=100) {
    clearInterval(loaded_interval);
  }
}
```

See also

`NetStream.bufferLength`

NetStream.bytesTotal

Availability

Flash Player 7.

Usage

```
my_ns.bytesTotal:Number
```

Description

Read-only property; the total size in bytes of the file being loaded into the player.

Example

The following example creates a progress bar using the Drawing API and the `bytesLoaded` and `bytesTotal` properties that displays the loading progress of video1.flv into the video object instance called `my_video`. A text field called `loaded_txt` is dynamically created to display information about the loading progress as well.

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

this.createTextField("loaded_txt", this.getNextHighestDepth(), 10, 10, 160,
  22);
this.createEmptyMovieClip("progressBar_mc", this.getNextHighestDepth());
progressBar_mc.createEmptyMovieClip("bar_mc",
  progressBar_mc.getNextHighestDepth());
with (progressBar_mc.bar_mc) {
  beginFill(0xFF0000);
  moveTo(0, 0);
  lineTo(100, 0);
  lineTo(100, 10);
  lineTo(0, 10);
  lineTo(0, 0);
  endFill();
  _xscale = 0;
}
progressBar_mc.createEmptyMovieClip("stroke_mc",
  progressBar_mc.getNextHighestDepth());
with (progressBar_mc.stroke_mc) {
  lineStyle(0, 0x000000);
  moveTo(0, 0);
  lineTo(100, 0);
  lineTo(100, 10);
  lineTo(0, 10);
  lineTo(0, 0);
}

var loaded_interval:Number = setInterval(checkBytesLoaded, 500, stream_ns);
function checkBytesLoaded(my_ns:NetStream) {
  var pctLoaded:Number = Math.round(my_ns.bytesLoaded/my_ns.bytesTotal*100);
```

```
    loaded_txt.text = Math.round(my_ns.bytesLoaded/1000)+" of
    "+Math.round(my_ns.bytesTotal/1000)+" KB loaded ("+pctLoaded+"%)";
    progressBar_mc.bar_mc._xscale = pctLoaded;
    if (pctLoaded>=100) {
      clearInterval(loaded_interval);
    }
  }
```

See also

NetStream.bytesLoaded, NetStream.bufferTime

NetStream.close()

Availability

Flash Player 7.

Note: This method is also supported in Flash Player 6 when used with Flash Communication Server. For more information, see the Flash Communication Server documentation.

Usage

```
my_ns.close() : Void
```

Parameters

None.

Returns

Nothing.

Description

Method; stops playing all data on the stream, sets the NetStream.time property to 0, and makes the stream available for another use. This command also deletes the local copy of an FLV file that was downloaded using HTTP.

Example

The following `onDisconnect()` function closes a connection and deletes the temporary copy of video1.flv that was stored on the local disk when you click the button called `close_btn`:

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

close_btn.onRelease = function(){
  stream_ns.close();
};
```

See also

`NetStream.pause()`, `NetStream.play()`

NetStream.currentFps

Availability

Flash Player 7.

Note: This property is also supported in Flash Player 6 when used with Flash Communication Server. For more information, see the Flash Communication Server documentation.

Usage

```
my_ns.currentFps:Number
```

Description

Read-only property; the number of frames per second being displayed. If you are exporting FLV files to be played back on a number of systems, you can check this value during testing to help you determine how much compression to apply when exporting the file.

Example

The following example creates a text field that displays the current number of frames per second that video1.flv displays.

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

this.createTextField("fps_txt", this.getNextHighestDepth(), 10, 10, 50, 22);
fps_txt.autoSize = true;
var fps_interval:Number = setInterval(displayFPS, 500, stream_ns);
function displayFPS(my_ns:NetStream) {
  fps_txt.text = "currentFps (frames per second):
  "+Math.floor(my_ns.currentFps);
}
```

NetStream.onStatus

Availability

Flash Player 7.

Note: This handler is also supported in Flash Player 6 when used with Flash Communication Server. For more information, see the Flash Communication Server documentation.

Usage

```
my_ns.onStatus = function(infoObject:Object) : Void{
  // Your code here
}
```

Parameters

infoObject A parameter defined according to the status or error message. For more information about this parameter, see "Description," below.

Returns

Nothing.

Description

Event handler; invoked every time a status change or error is posted for the NetStream object. If you want to respond to this event handler, you must create a function to process the information object.

The information object has a `code` property containing a string that describes the result of the `onStatus` handler, and a `level` property containing a string that is either `status` or `error`.

In addition to this `onStatus` handler, Flash also provides a "super" function called System.onStatus. If `onStatus` is invoked for a particular object and there is no function assigned to respond to it, Flash processes a function assigned to `System.onStatus` if it exists.

The following events notify you when certain NetStream activities occur.

Code property	Level property	Meaning
`NetStream.Buffer.Empty`	status	Data is not being received quickly enough to fill the buffer. Data flow will be interrupted until the buffer refills, at which time a `NetStream.Buffer.Full` message will be sent and the stream will begin playing again.
`NetStream.Buffer.Full`	status	The buffer is full and the stream will begin playing.
`NetStream.Play.Start`	status	Playback has started.
`NetStream.Play.Stop`	status	Playback has stopped.
`NetStream.Play.StreamNotFound`	error	The FLV passed to the `play()` method can't be found.

If you consistently see errors regarding buffer, you should try changing the buffer using the NetStream.setBufferTime() method.

Example

The following example displays data about the stream in the Output panel:

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");

stream_ns.onStatus = function(infoObject:Object) {
  trace("NetStream.onStatus called: ("+getTimer()+" ms)");
  for (var prop in infoObject) {
    trace("\t"+prop+":\t"+infoObject[prop]);
  }
  trace("");
};
```

See also

`System.onStatus`, `NetStream.setBufferTime()`

NetStream.pause()

Availability

Flash Player 7.

Note: This method is also supported in Flash Player 6 when used with Flash Communication Server. For more information, see the Flash Communication Server documentation.

Usage

`my_ns.pause( [ pauseResume:Boolean ] ) : Void`

Parameters

`pauseResume` A Boolean value specifying whether to pause play (`true`) or resume play (`false`). If you omit this parameter, `NetStream.pause()` acts as a toggle: the first time it is called on a specified stream, it pauses play, and the next time it is called, it resumes play. This parameter is optional.

Returns

Nothing.

Description

Method; pauses or resumes playback of a stream.

The first time you call this method (without sending a parameter), it pauses play; the next time, it resumes play. You might want to attach this method to a button that the user presses to pause or resume playback.

Example

The following examples illustrate some uses of this method:

```
my_ns.pause(); // pauses play first time issued
my_ns.pause(); // resumes play
my_ns.pause(false); // no effect, play continues
my_ns.pause(); // pauses play
```

See also

`NetStream.close()`, `NetStream.play()`

NetStream.play()

Availability

Flash Player 7.

Note: This method is also supported in Flash Player 6 when used with Flash Communication Server. For more information, see the Flash Communication Server documentation.

Usage

```
my_ns.play("fileName":String);
```

Parameters

`fileName` The name of an FLV file to play, in quotation marks. Both http:// and file:// formats are supported; the file:// location is always relative to the location of the SWF file.

Returns

Nothing.

Description

Method; begins playback of an external video (FLV) file. To view video data, you must call a Video.attachVideo() method; audio being streamed with the video, or an FLV file that contains only audio, is played automatically.

If you want to control the audio associated with an FLV file, you can use `MovieClip.attachAudio()` to route the audio to a movie clip; you can then create a Sound object to control some aspects of the audio. For more information, see MovieClip.attachAudio().

If the FLV file can't be found, the NetStream.onStatus event handler is invoked. If you want to stop a stream that is currently playing, use NetStream.close().

You can play local FLV files that are stored in the same directory as the SWF file or in a subdirectory; you can't navigate to a higher-level directory. For example, if the SWF file is located in a directory named /training, and you want to play a video stored in the /training/videos directory, you would use the following syntax:

```
my_ns.play("videos/videoName.flv");
```

To play a video stored in the /training directory, you would use the following syntax:

```
my_ns.play("videoName.flv");
```

Example

The following example illustrates some ways to use the `NetStream.play()` command:

```
// Play a file that is on the user's computer
// The joe_user directory is a subdirectory of the directory
//   in which the SWF is stored
my_ns.play("file://joe_user/flash/videos/lectureJune26.flv");

// Play a file on a server
my_ns.play("http://someServer.someDomain.com/flash/video/orientation.flv");
```

See also

`MovieClip.attachAudio()`, `NetStream.close()`, `NetStream.pause()`, `Video.attachVideo()`

NetStream.seek()

Availability

Flash Player 7.

Note: This method is also supported in Flash Player 6 when used with Flash Communication Server. For more information, see the Flash Communication Server documentation.

Usage

`my_ns.seek(numberOfSeconds:Number) : Void`

Parameters

`numberOfSeconds` The approximate time value, in seconds, to move to in an FLV file. The playhead moves to the keyframe of the video that's closest to `numberOfSeconds`.

- To return to the beginning of the stream, pass 0 for `numberOfSeconds`.
- To seek forward from the beginning of the stream, pass the number of seconds you want to advance. For example, to position the playhead at 15 seconds from the beginning, use `my_ns.seek(15)`.
- To seek relative to the current position, pass `my_ns.time + n` or `my_ns.time - n` to seek `n` seconds forward or backward, respectively, from the current position. For example, to rewind 20 seconds from the current position, use `my_ns.seek(my_ns.time - 20)`.

The precise location to which a video seeks will differ depending on the frames per second setting at which it was exported. Therefore, if the same video is exported at 6 fps and 30 fps, it will seek to two different locations if you use, for example, `my_ns.seek(15)` for both video objects.

Returns

Nothing.

Description

Method; seeks the keyframe closest to the specified number of seconds from the beginning of the stream. The stream resumes playing when it reaches the specified location in the stream.

Example

The following example illustrates some ways to use the `NetStream.seek()` command:

```
// Return to the beginning of the stream
my_ns.seek(0);

// Move to a location 30 seconds from the beginning of the stream
my_ns.seek(30);

// Move backwards three minutes from current location
my_ns.seek(my_ns.time - 180);
```

See also

`NetStream.play()`, `NetStream.time`

NetStream.setBufferTime()

Availability

Flash Player 7.

Note: This method is also supported in Flash Player 6 when used with Flash Communication Server. For more information, see the Flash Communication Server documentation.

Usage

```
my_ns.setBufferTime(numberOfSeconds:Number) : Void
```

Parameters

`numberOfSeconds` The number of seconds of data to be buffered before Flash begins displaying data. The default value is 0.1 (one-tenth of a second).

Description

Method; specifies how long to buffer messages before starting to display the stream. For example, if you want to make sure that the first 15 seconds of the stream play without interruption, set `numberOfSeconds` to 15; Flash begins playing the stream only after 15 seconds of data are buffered.

Example

See the example for NetStream.bufferLength.

See also

`NetStream.bufferTime`

NetStream.time

Availability

Flash Player 7.

Note: This property is also supported in Flash Player 6 when used with Flash Communication Server. For more information, see the Flash Communication Server documentation.

Usage

`my_ns.time:Number`

Description

Read-only property; the position of the playhead, in seconds.

Example

The following example displays the current position of the playhead in a dynamically created text field called `time_txt`. Select New Video from the Library options menu to create a video object instance, and give it an instance name `my_video`. Add the following ActionScript to your FLA or AS file:

```
var connection_nc:NetConnection = new NetConnection();
connection_nc.connect(null);
var stream_ns:NetStream = new NetStream(connection_nc);
my_video.attachVideo(stream_ns);
stream_ns.play("video1.flv");
//
stream_ns.onStatus = function(infoObject:Object) {
  statusCode_txt.text = infoObject.code;
};

this.createTextField("time_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
time_txt.text = "LOADING";

var time_interval:Number = setInterval(checkTime, 500, stream_ns);
function checkTime(my_ns:NetStream) {
  var ns_seconds:Number = my_ns.time;
  var minutes:Number = Math.floor(ns_seconds/60);
  var seconds = Math.floor(ns_seconds%60);
  if (seconds<10) {
    seconds = "0"+seconds;
  }
  time_txt.text = minutes+":"+seconds;
}
```

See also

`NetStream.bufferLength`, `NetStream.bytesLoaded`

new

Availability

Flash Player 5.

Usage

```
new constructor()
```

Parameters

`constructor` A function followed by any optional parameters in parentheses. The function is usually the name of the object type (for example, Array, Number, or Object) to be constructed.

Returns

Nothing.

Description

Operator; creates a new, initially anonymous, object and calls the function identified by the `constructor` parameter. The `new` operator passes to the function any optional parameters in parentheses as well as the newly created object, which is referenced using the keyword `this`. The constructor function can use `this` to set the variables of the object.

Example

The following example creates the `Book()` function and then uses the `new` operator to create the objects `book1` and `book2`:

```
function Book(name:String, price:Number) {
  this.name = name;
  this.price = price;
}
var book1 = new Book("Confederacy of Dunces", 19.95);
var book2 = new Book("The Floating Opera", 10.95);
```

Example

The following example uses the `new` operator to create an Array object with 18 elements:

```
var golfCourse_array:Array = new Array(18);
```

See also

`[] (array access)`, `{} (object initializer)`

newline

Availability

Flash Player 4.

Usage

`newline`

Parameters

None.

Returns

Nothing.

Description

Constant; inserts a carriage return character (`\n`) that generates a blank line in text output generated by your code. Use `newline` to make space for information that is retrieved by a function or statement in your code.

Example

The following example shows how `newline` displays output from the `trace()` statement on multiple lines.

```
var myName:String = "Lisa", myAge:Number = 30;
trace(myName+myAge);
trace("-----");
trace(myName+newline+myAge);
/* output:
  Lisa30
  -----
  Lisa
  30
*/
```

nextFrame()

Availability

Flash 2.

Usage

```
nextFrame() : Void
```

Parameters

None.

Returns

Nothing.

Description

Function; sends the playhead to the next frame.

Example

In the following example, when the user presses the Right or Down arrow key, the playhead goes to the next frame and stops. If the user presses the Left or Up arrow key, the playhead goes to the previous frame and stops. The listener is initialized to wait for the arrow key to be pressed, and the `init` variable is used to prevent the listener from being redefined if the playhead returns to Frame 1.

```
stop();
if (init == undefined) {
  someListener = new Object();
  someListener.onKeyDown = function() {
    if (Key.isDown(Key.LEFT) || Key.isDown(Key.UP)) {
      _level0.prevFrame();
    } else if (Key.isDown(Key.RIGHT) || Key.isDown(Key.DOWN)) {
      _level0.nextFrame();
    }
  };
  Key.addListener(someListener);
  init = 1;
}
```

nextScene()

Availability

Flash 2.

Usage

```
nextScene() : Void
```

Parameters

None.

Returns

Nothing.

Description

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

Example

In the following example, when a user clicks the button that is created at runtime, the playhead is sent to Frame 1 of the next scene. Create two scenes, and enter the following ActionScript on Frame 1 of Scene 1.

```
stop();
if (init == undefined) {
  this.createEmptyMovieClip("nextscene_mc", this.getNextHighestDepth());
  nextscene_mc.createTextField("nextscene_txt", this.getNextHighestDepth(),
  200, 0, 100, 22);
  nextscene_mc.nextscene_txt.autoSize = true;
  nextscene_mc.nextscene_txt.border = true;
  nextscene_mc.nextscene_txt.text = "Next Scene";
  this.createEmptyMovieClip("prevscene_mc", this.getNextHighestDepth());
  prevscene_mc.createTextField("prevscene_txt", this.getNextHighestDepth(),
  00, 0, 100, 22);
  prevscene_mc.prevscene_txt.autoSize = true;
  prevscene_mc.prevscene_txt.border = true;
  prevscene_mc.prevscene_txt.text = "Prev Scene";
  nextscene_mc.onRelease = function() {
    nextScene();
  };
  prevscene_mc.onRelease = function() {
    prevScene();
  };
  init = true;
}
```

Make sure you place a `stop()` action on Frame 1 of Scene 2.

See also

`prevScene()`

null

Availability

Flash Player 5.

Usage

`null`

Parameters

None.

Returns

Nothing.

Description

Constant; a special value that can be assigned to variables or returned by a function if no data was provided. You can use `null` to represent values that are missing or that do not have a defined data type.

Example

In a numeric context, `null` evaluates to 0. Equality tests can be performed with `null`. In this statement, a binary tree node has no left child, so the field for its left child could be set to `null`.

```
if (tree.left == null) {
  tree.left = new TreeNode();
}
```

Number()

Availability

Flash Player 4; behavior changed in Flash Player 7.

Usage

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

Parameters

`expression`	An expression to convert to a number.

Returns

A number or `NaN` (not a number).

Description

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

- If *`expression`* is a number, the return value is *`expression`*.
- If *`expression`* is a Boolean value, the return value is 1 if *`expression`* is `true`, 0 if *`expression`* is `false`.
- If *`expression`* is a string, the function attempts to parse *`expression`* as a decimal number with an optional trailing exponent (that is, 1.57505e-3).
- If *`expression`* is `NaN`, the return value is `NaN`.
- If *`expression`* is `undefined`, the return value is as follows:
  - In files published for Flash Player 6 or earlier, the result is 0.
  - In files published for Flash Player 7 or later, the result is `NaN`.

This function is used to convert Flash 4 files containing deprecated operators that are imported into the Flash 5 or later authoring environment. For more information, see `& (bitwise AND)`.

Example

In the following example, a text field is created on the Stage at runtime:

```
this.createTextField("counter_txt", this.getNextHighestDepth(), 0, 0, 100,
  22);
counter_txt.autoSize = true;
counter_txt.text = 0;
function incrementInterval():Void {
  var counter:Number = counter_txt.text;
  // Without the Number() function, Flash would concatenate the value instead
  // of adding values. You could also use "counter_txt.text++;"
  counter_txt.text = Number(counter)+1;
}
var intervalID:Number = setInterval(incrementInterval, 1000);
```

See also

`NaN`, Number class

Number class

Availability

Flash Player 5 (became a native object in Flash Player 6, which improved performance significantly).

Description

The Number class is a simple wrapper object for the Number data type. You can manipulate primitive numeric values by using the methods and properties associated with the Number class. This class is identical to the JavaScript Number class.

To call the Number class methods, you must first use the constructor to create an instance of the Number class. The properties of the Number class, however, are static, which means you do not need an object to use them, so you do not need to use the constructor.

The following example calls the `toString()` method of the Number class, which returns the string `1234`:

```
var myNumber:Number = new Number(1234);
myNumber.toString();
```

The following example assigns the value of the `MIN_VALUE` property to a variable declared without the use of the constructor:

```
var smallest:Number = Number.MIN_VALUE;
```

Method summary for the Number class

Method	Description
`Number.toString()`	Returns the string representation of a Number object.
`Number.valueOf()`	Returns the primitive value of a Number object.

Property summary for the Number class

The properties in the following table are constants:

Property	Description
`Number.MAX_VALUE`	The largest representable number (double-precision IEEE-754). This number is approximately 1.79E+308.
`Number.MIN_VALUE`	The smallest representable number (double-precision IEEE-754). This number is approximately 5e-324.
`Number.NaN`	The value for Not a Number (`NaN`).
`Number.NEGATIVE_INFINITY`	The value for negative infinity.
`Number.POSITIVE_INFINITY`	The value for positive infinity. This value is the same as the global variable Infinity.

Constructor for the Number class

Availability

Flash Player 5.

Usage

```
new Number(value:Number) : Number
```

Parameters

`value` The numeric value of the Number object being created or a value to be converted to a number. The default value is 0 if `value` is not provided.

Returns

A reference to a Number object.

Description

Constructor; creates a new Number object. You must use the Number constructor when using Number.toString() and Number.valueOf(). You do not use a constructor when using the properties of a Number object. The `new Number` constructor is primarily used as a placeholder. A Number object is not the same as the Number() function that converts a parameter to a primitive value.

Example

The following code constructs new Number objects:

```
var n1:Number = new Number(3.4);
var n2:Number = new Number(-10);
```

See also

`Number()`

Number.MAX_VALUE

Availability

Flash Player 5.

Usage

```
Number.MAX_VALUE:Number
```

Description

Property; the largest representable number (double-precision IEEE-754). This number is approximately 1.79e+308.

Example

The following ActionScript displays the largest and smallest representable numbers to the Output panel.

```
trace("Number.MIN_VALUE = "+Number.MIN_VALUE);
trace("Number.MAX_VALUE = "+Number.MAX_VALUE);
```

This code displays the following values:

```
Number.MIN_VALUE = 4.94065645841247e-324
Number.MAX_VALUE = 1.79769313486232e+308
```

Number.MIN_VALUE

Availability

Flash Player 5.

Usage

```
Number.MIN_VALUE
```

Description

Property; the smallest representable number (double-precision IEEE-754). This number is approximately 5e-324.

Example

The following ActionScript displays the largest and smallest representable numbers to the Output panel.

```
trace("Number.MIN_VALUE = "+Number.MIN_VALUE);
trace("Number.MAX_VALUE = "+Number.MAX_VALUE);
```

This code displays the following values:

```
Number.MIN_VALUE = 4.94065645841247e-324
Number.MAX_VALUE = 1.79769313486232e+308
```

Number.NaN

Availability

Flash Player 5.

Usage

`Number.NaN`

Description

Property; the IEEE-754 value representing Not A Number (`NaN`).

See also

`isNaN()`, `NaN`

Number.NEGATIVE_INFINITY

Availability

Flash Player 5.

Usage

`Number.NEGATIVE_INFINITY`

Description

Property; specifies the IEEE-754 value representing negative infinity. The value of this property is the same as that of the constant `-Infinity`.

Negative infinity is a special numeric value that is returned when a mathematical operation or function returns a negative value larger than can be represented.

Example

This example compares the result of dividing the following values.

```
var posResult:Number = 1/0;
if (posResult == Number.POSITIVE_INFINITY) {
  trace("posResult = "+posResult); // output: posResult = Infinity
}
var negResult:Number = -1/0;
if (negResult == Number.NEGATIVE_INFINITY) {
  trace("negResult = "+negResult); // output: negResult = -Infinity
```

Number.POSITIVE_INFINITY

Availability

Flash Player 5.

Usage

```
Number.POSITIVE_INFINITY
```

Description

Property; specifies the IEEE-754 value representing positive infinity. The value of this property is the same as that of the constant Infinity.

Positive infinity is a special numeric value that is returned when a mathematical operation or function returns a value larger than can be represented.

Example

This example compares the result of dividing the following values.

```
var posResult:Number = 1/0;
if (posResult == Number.POSITIVE_INFINITY) {
  trace("posResult = "+posResult); // output: posResult = Infinity
}
var negResult:Number = -1/0;
if (negResult == Number.NEGATIVE_INFINITY) {
  trace("negResult = "+negResult); // output: negResult = -Infinity
```

Number.toString()

Availability

Flash Player 5; behavior changed in Flash Player 7.

Usage

myNumber.toString(*radix:Number*) : *String*

Parameters

radix Specifies the numeric base (from 2 to 36) to use for the number-to-string conversion. If you do not specify the *radix* parameter, the default value is 10.

Returns

A string.

Description

Method; returns the string representation of the specified Number object (*myNumber*).

Example

The following example uses 2 and 8 for the *radix* parameter and returns a string that contains the corresponding representation of the number 9:

```
var myNumber:Number = new Number(9);
trace(myNumber.toString(2)); // output: 1001
trace(myNumber.toString(8)); // output: 11
```

The following example results in a hexadecimal value.

```
var r:Number = new Number(250);
var g:Number = new Number(128);
var b:Number = new Number(114);
var rgb:String = "0x"+ r.toString(16)+g.toString(16)+b.toString(16);
trace(rgb);
// output: rgb:0xFA8072 (Hexadecimal equivalent of the color 'salmon')
```

Number.valueOf()

Availability

Flash Player 5.

Usage

myNumber.valueOf() : *Number*

Parameters

None.

Returns

Number.

Description

Method; returns the primitive value type of the specified Number object.

Example

The following example results in the primative value of the `numSocks` object.

```
var numSocks = new Number(2);
trace(numSocks.valueOf()); // output: 2
```

Object()

Availability

Flash Player 5.

Usage

```
Object( [ value ] ) : Object
```

Parameters

value A number, string, or Boolean value.

Returns

An object.

Description

Conversion function; creates a new empty object or converts the specified number, string, or Boolean value to an object. This command is equivalent to creating an object using the Object constructor (see "Constructor for the Object class" on page 660).

Example

In the following example, a new empty object is created, and then the object is populated with values:

```
var company:Object = new Object();
company.name = "Macromedia, Inc.";
company.address = "600 Townsend Street";
company.city = "San Francisco";
company.state = "CA";
company.postal = "94103";
for (var i in company) {
  trace("company."+i+" = "+company[i]);
}
```

Object class

Availability

Flash Player 5 (became a native object in Flash Player 6, which improved performance significantly).

Description

The Object class is at the root of the ActionScript class hierarchy. This class contains a small subset of the features provided by the JavaScript Object class.

Method summary for the Object class

Method	Description
`Object.addProperty()`	Creates a getter/setter property on an object.
`Object.registerClass()`	Associates a movie clip symbol with an ActionScript object class.
`Object.toString()`	Converts the specified object to a string and returns it.
`Object.unwatch()`	Removes the watchpoint that `Object.watch()` created.
`Object.valueOf()`	Returns the primitive value of an object.
`Object.watch()`	Registers an event handler to be invoked when a specified property of an ActionScript object changes.

Property summary for the Object class

Property	Description
`Object.__proto__`	A reference to the `prototype` property of the object's constructor function.
`Object.__resolve`	A reference to a user-defined function that is called automatically if ActionScript code refers to an undefined method or property.
`Object.constructor`	A reference to the object's constructor function.

Constructor for the Object class

Availability

Flash Player 5.

Usage

```
new Object([value]) : Object
```

Parameters

`value` A number, Boolean value, or string to be converted to an object. If you do not specify *`value`*, the constructor creates a new object with no defined properties. This parameter is optional.

Returns

A reference to an Object object.

Description

Constructor; creates an Object object and stores a reference to the object's constructor method in the object's `constructor` property.

Example

The following example creates a generic object named myObject:

```
var myObject:Object = new Object();
```

Object.addProperty()

Availability

Flash Player 6. In ActionScript 2.0 classes, you can use get or set instead of this method.

Usage

```
myObject.addProperty(prop:String, getFunc:Function, setFunc:Function) :
  Boolean
```

Parameters

prop A string; the name of the object property to create.

getFunc The function that is invoked to retrieve the value of the property; this parameter is a Function object.

setFunc The function that is invoked to set the value of the property; this parameter is a Function object. If you pass the value `null` for this parameter, the property is read-only.

Returns

A Boolean value: `true` if the property is successfully created; `false` otherwise.

Description

Method; creates a getter/setter property. When Flash reads a getter/setter property, it invokes the `get` function, and the function's return value becomes the value of *prop*. When Flash writes a getter/setter property, it invokes the `set` function and passes it the new value as a parameter. If a property with the given name already exists, the new property overwrites it.

A "get" function is a function with no parameters. Its return value can be of any type. Its type can change between invocations. The return value is treated as the current value of the property.

A "set" function is a function that takes one parameter, which is the new value of the property. For example, if property `x` is assigned by the statement `x = 1`, the set function is passed the parameter `1` of type number. The return value of the set function is ignored.

You can add getter/setter properties to prototype objects. If you add a getter/setter property to a prototype object, all object instances that inherit the prototype object inherit the getter/setter property. This makes it possible to add a getter/setter property in one location, the prototype object, and have it propagate to all instances of a class (similar to adding methods to prototype objects). If a get/set function is invoked for a getter/setter property in an inherited prototype object, the reference passed to the get/set function is the originally referenced object—not the prototype object.

If invoked incorrectly, `Object.addProperty()` can fail with an error. The following table describes errors that can occur:

Error condition	What happens
prop is not a valid property name; for example, an empty string.	Returns `false` and the property is not added.

Error condition	What happens
getFunc is not a valid function object.	Returns false and the property is not added.
setFunc is not a valid function object.	Returns false and the property is not added.

Example

In the following example, an object has two internal methods, setQuantity() and getQuantity(). A property, bookcount, can be used to invoke these methods when it is either set or retrieved. A third internal method, getTitle(), returns a read-only value that is associated with the property bookname. When a script retrieves the value of myBook.bookcount, the ActionScript interpreter automatically invokes myBook.getQuantity(). When a script modifies the value of myBook.bookcount, the interpreter invokes myObject.setQuantity(). The bookname property does not specify a set function, so attempts to modify bookname are ignored.

```
function Book() {
  this.setQuantity = function(numBooks:Number):Void {
    this.books = numBooks;
  };
  this.getQuantity = function():Number {
    return this.books;
  };
  this.getTitle = function():String {
    return "Catcher in the Rye";
  };
  this.addProperty("bookcount", this.getQuantity, this.setQuantity);
  this.addProperty("bookname", this.getTitle, null);
}
var myBook = new Book();
myBook.bookcount = 5;
trace("You ordered "+myBook.bookcount+" copies of "+myBook.bookname);
// output: You ordered 5 copies of Catcher in the Rye
```

The previous example works, but the properties bookcount and bookname are added to every instance of the Book object, which requires having two properties for every instance of the object. If there are many properties, such as bookcount and bookname, in a class, they could consume a great deal of memory. Instead, you can add the properties to Book.prototype so that the bookcount and bookname properties exist only in one place. The effect, however, is the same as that of the code in the example that added bookcount and bookname directly to every instance. If an attempt is made to access either property in a Book instance, the property's absence will cause the prototype chain to be ascended until the versions defined in Book.prototype are encountered. The following example shows how to add the properties to Book.prototype:

```
function Book() {}
Book.prototype.setQuantity = function(numBooks:Number):Void {
  this.books = numBooks;
};
Book.prototype.getQuantity = function():Number {
  return this.books;
};
Book.prototype.getTitle = function():String {
  return "Catcher in the Rye";
};
```

```
Book.prototype.addProperty("bookcount", Book.prototype.getQuantity,
  Book.prototype.setQuantity);
Book.prototype.addProperty("bookname", Book.prototype.getTitle, null);
var myBook = new Book();
myBook.bookcount = 5;
trace("You ordered "+myBook.bookcount+" copies of "+myBook.bookname);
```

The following example shows how to use the implicit getter and setter functions available in ActionScript 2.0. For more information, see "Implicit getter/setter methods" in *Using ActionScript in Flash*.Rather than defining the `Book` function and editing `Book.prototype`, you define the `Book` class in an external file named Book.as. For more information, see "Creating and using classes" in *Using ActionScript in Flash*. The following code must be in a separate external file named Book.as that contains only this class definition and resides within the Flash application's classpath:

```
class Book {
  var books:Number;
  function set bookcount(numBooks:Number):Void {
    this.books = numBooks;
  }
  function get bookcount():Number {
    return this.books;
  }
  function get bookname():String {
    return "Catcher in the Rye";
  }
}
```

The following code can then be placed in a FLA file and will function the same way as it does in the previous examples:

```
var myBook:Book = new Book();
myBook.bookcount = 5;
trace("You ordered "+myBook.bookcount+" copies of "+myBook.bookname);
```

Object.constructor

Availability

Flash Player 5

Usage

myObject.constructor

Description

Property; reference to the constructor function for a given object instance. The constructor property is automatically assigned to all objects when they are created using the constructor for the Object class.

Example

The following example is a reference to the constructor function for the myObject object.

```
var my_str:String = new String("sven");
trace(my_str.constructor == String); //output: true
```

If you use the instanceof operator, you can also determine if an object belongs to a specified class:

```
var my_str:String = new String("sven");
trace(my_str instanceof String); //output: true
```

However, in the following example the Object.constructor property converts primitive data types (such as the string literal seen here) into wrapper objects. The instanceof operator does not perform any conversion, as seen in the following example:

```
var my_str:String = "sven";
trace(my_str.constructor == String); //output: true
trace(my_str instanceof String); //output: false
```

See Also

instanceof

Object.__proto__

Availability

Flash Player 5.

Usage

myObject.__proto__

Description

Property; refers to the `prototype` property of the constructor function that created *myObject*. The `__proto__` property is automatically assigned to all objects when they are created. The ActionScript interpreter uses the `__proto__` property to access the `prototype` property of the object's constructor function to find out what properties and methods the object inherits from its class.

Object.registerClass()

Availability

Flash Player 6. If you are using ActionScript 2.0 classes, you can use the ActionScript 2.0 Class field in the Linkage Properties or Symbol Properties dialog box to associate an object with a class instead of using this method. For more information, see "Assigning a class to a movie clip symbol". in *Using ActionScript in Flash*.

Usage

```
Object.registerClass(symbolID:String, theClass:Function) : Boolean
```

Parameters

`symbolID` String; the linkage identifier of the movie clip symbol or the string identifier for the ActionScript class.

`theClass` A reference to the constructor function of the ActionScript class or `null` to unregister the symbol.

Returns

A Boolean value: if the class registration succeeds, a value of `true` is returned; `false` otherwise.

Description

Method; associates a movie clip symbol with an ActionScript object class. If a symbol doesn't exist, Flash creates an association between a string identifier and an object class.

When an instance of the specified movie clip symbol is placed on the Timeline, it is registered to the class specified by the `theClass` parameter rather than to the class MovieClip.

When an instance of the specified movie clip symbol is created by using MovieClip.attachMovie() or MovieClip.duplicateMovieClip(), it is registered to the class specified by `theClass` rather than to the MovieClip class. If `theClass` is `null`, this method removes any ActionScript class definition associated with the specified movie clip symbol or class identifier. For movie clip symbols, any existing instances of the movie clip remain unchanged, but new instances of the symbol are associated with the default class MovieClip.

If a symbol is already registered to a class, this method replaces it with the new registration.

When a movie clip instance is placed by the Timeline or created using `attachMovie()` or `duplicateMovieClip()`, ActionScript invokes the constructor for the appropriate class with the keyword `this` pointing to the object. The constructor function is invoked with no parameters.

If you use this method to register a movie clip with an ActionScript class other than MovieClip, the movie clip symbol doesn't inherit the methods, properties, and events of the built-in MovieClip class unless you include the MovieClip class in the prototype chain of the new class. The following code creates a new ActionScript class called `theClass` that inherits the properties of the MovieClip class:

```
theClass.prototype = new MovieClip();
```

See also

`MovieClip.attachMovie()`, `MovieClip.duplicateMovieClip()`

Object.__resolve

Availability

Flash Player 6.

Usage

```
myObject.__resolve = function (name:String) {
  // your statements here
};
```

Parameters

`name` A string representing the name of the undefined property.

Returns

Nothing

Description

Property; a reference to a user-defined function that is invoked if ActionScript code refers to an undefined property or method. If ActionScript code refers to an undefined property or method of an object, Flash Player determines whether the object's `__resolve` property is defined. If `__resolve` is defined, the function to which it refers is executed and passed the name of the undefined property or method. This lets you programmatically supply values for undefined properties and statements for undefined methods and make it seem as if the properties or methods are actually defined.

This property is useful for enabling highly transparent client/server communication, and is the recommended way of invoking server-side methods.

Example

The following examples progressively build upon the first example and illustrate five different usages of the `__resolve` property. To aid understanding, key statements that differ from the previous usage are in bold typeface.

Usage 1: the following example uses `__resolve` to build an object where every undefined property returns the value `"Hello, world!"`.

```
// instantiate a new object
var myObject:Object = new Object();

// define the __resolve function
myObject.__resolve = function (name) {
  return "Hello, world!";
};
trace (myObject.property1); // output: Hello, world!
trace (myObject.property2); // output: Hello, world!
```

Usage 2: the following example uses `__resolve` as a *functor*, which is a function that generates functions. Using `__resolve` redirects undefined method calls to a generic function named `myFunction`.

```
// instantiate a new object
```

```
var myObject:Object = new Object();

// define a function for __resolve to call
myObject.myFunction = function (name) {
  trace("Method " + name + " was called");
};

// define the __resolve function
myObject.__resolve = function (name) {
  return function () { this.myFunction(name); };
};

// test __resolve using undefined method names
myObject.someMethod(); // output: Method someMethod was called
myObject.someOtherMethod(); //output: Method someOtherMethod was called
```

Usage 3: The following example builds on the previous example by adding the ability to cache resolved methods. By caching methods, `__resolve` is called only once for each method of interest. This allows *lazy construction* of object methods. Lazy construction is an optimization technique that defers the creation, or *construction*, of methods until the time at which a method is first used.

```
// instantiate a new object
var myObject:Object = new Object();
// define a function for __resolve to call
myObject.myFunction = function(name) {
  trace("Method "+name+" was called");
};
// define the __resolve function
myObject.__resolve = function(name) {
  trace("Resolve called for "+name);  // to check when __resolve is called
  // Not only call the function, but also save a reference to it
  var f:Function = function () {
    this.myFunction(name);
  };
  // create a new object method and assign it the reference
  this[name] = f;
  // return the reference
  return f;
};
// test __resolve using undefined method names
// __resolve will only be called once for each method name
myObject.someMethod();  // calls __resolve
myObject.someMethod();  // does not call __resolve because it is now defined
myObject.someOtherMethod();  // calls __resolve
myObject.someOtherMethod();  // does not call __resolve, no longer undefined
```

Usage 4: The following example builds on the previous example by reserving a method name, `onStatus()`, for local use so that it is not resolved in the same way as other undefined properties. Added code is in bold typeface.

```
// instantiate a new object
var myObject:Object = new Object();
// define a function for __resolve to call
myObject.myFunction = function(name) {
  trace("Method "+name+" was called");
};
// define the __resolve function
myObject.__resolve = function(name) {
  // reserve the name "onStatus" for local use
  if (name == "onStatus") {
    return undefined;
  }
  trace("Resolve called for "+name);  // to check when __resolve is called
  // Not only call the function, but also save a reference to it
  var f:Function = function () {
    this.myFunction(name);
  };
  // create a new object method and assign it the reference
  this[name] = f;
  // return the reference
  return f;
};
// test __resolve using the method name "onStatus"
trace(myObject.onStatus("hello"));
// output: undefined
```

Usage 5: The following example builds on the previous example by creating a functor that accepts parameters. This example makes extensive use of the arguments object, and uses several methods of the Array class.

```
// instantiate a new object
var myObject:Object = new Object();

// define a generic function for __resolve to call
myObject.myFunction = function (name) {
  arguments.shift();
  trace("Method " + name + " was called with arguments: " +
  arguments.join(','));
};

// define the __resolve function
myObject.__resolve = function (name) {
  // reserve the name "onStatus" for local use
  if (name == "onStatus") {
    return undefined;
  }
  var f:Function = function () {
    arguments.unshift(name);
    this.myFunction.apply(this, arguments);
  };
  // create a new object method and assign it the reference
  this[name] = f;
  // return the reference to the function
  return f;
};

// test __resolve using undefined method names with parameters
myObject.someMethod("hello");
// output: Method someMethod was called with arguments: hello
```

```
myObject.someOtherMethod("hello","world");
// output: Method someOtherMethod was called with arguments: hello,world
```

Object.toString()

Availability

Flash Player 5.

Usage

```
myObject.toString() : String
```

Parameters

None.

Returns

A string.

Description

Method; converts the specified object to a string and returns it.

Example

This example shows the return value for toString() on a generic object:

```
var myObject:Object = new Object();
trace(myObject.toString()); // output: [object Object]
```

This method can be overridden to return a more meaningful value. The following examples show that this method has been overridden for the built-in classes Date, Array, and Number:

```
// Date.toString() returns the current date and time
var myDate:Date = new Date();
trace(myDate.toString()); // output: [current date and time]

// Array.toString() returns the array contents as a comma-delimited string
var myArray:Array = new Array("one", "two");
trace(myArray.toString()); // output: one,two

// Number.toString() returns the number value as a string
// Because trace() won't tell us whether the value is a string or number
// we will also use typeof() to test whether toString() works.
var myNumber:Number = 5;
trace(typeof (myNumber));  // output: number
trace(myNumber.toString()); // output: 5
trace(typeof (myNumber.toString())); // output: string
```

The following example shows how to override `toString()` in a custom class. First create a text file named *Vehicle.as* that contains only the Vehicle class definition and place it into your Classes folder inside your Configuration folder.

```
// contents of Vehicle.as
class Vehicle {
  var numDoors:Number;
  var color:String;
  function Vehicle(param_numDoors:Number, param_color:String) {
    this.numDoors = param_numDoors;
    this.color = param_color;
    }
    function toString():String {
        var doors:String = "door";
        if (this.numDoors > 1) {
            doors += "s";
        }
        return ("A vehicle that is " + this.color + " and has " + this.numDoors + 
    " " + doors);
    }
}

// code to place into a FLA file
var myVehicle:Vehicle = new Vehicle(2, "red");
trace(myVehicle.toString());
// output: A vehicle that is red and has 2 doors

// for comparison purposes, this is a call to valueOf()
// there is no primitive value of myVehicle, so the object is returned
// giving the same output as toString().
trace(myVehicle.valueOf());
// output: A vehicle that is red and has 2 doors
```

Object.unwatch()

Availability

Flash Player 6.

Usage

`myObject.unwatch (prop:String) : Boolean`

Parameters

prop	A string; the name of the object property that should no longer be watched.

Returns

A Boolean value: `true` if the watchpoint is successfully removed, `false` otherwise.

Description

Method; removes a watchpoint that Object.watch() created. This method returns a value of `true` if the watchpoint is successfully removed, `false` otherwise.

Example

See the example for Object.watch().

See also

`Object.addProperty()`, `Object.watch()`

Object.valueOf()

Availability

Flash Player 5.

Usage

```
myObject.valueOf() : Object
```

Parameters

None.

Returns

The primitive value of the specified object or the object itself.

Description

Method; returns the primitive value of the specified object. If the object does not have a primitive value, the object is returned.

Example

The following example shows the return value of valueOf() for a generic object (which does not have a primitive value) and compares it to the return value of toString():

```
// Create a generic object
var myObject:Object = new Object();
trace(myObject.valueOf()); // output: [object Object]
trace(myObject.toString()); // output: [object Object]
```

The following examples show the return values for the built-in classes Date and Array, and compares them to the return values of `Object.toString()`:

```
// Create a new Date object set to February 1, 2004, 8:15 AM
// The toString() method returns the current time in human-readable form
// The valueOf() method returns the primitive value in milliseconds
var myDate:Date = new Date(2004,01,01,8,15);
trace(myDate.toString()); // output: Sun Feb 1 08:15:00 GMT-0800 2004
trace(myDate.valueOf()); // output: 1075652100000

// Create a new Array object containing two simple elements
// In this case both toString() and valueOf() return the same value: one,two
var myArray:Array = new Array("one", "two");
trace(myArray.toString()); // output: one,two
trace(myArray.valueOf()); // output: one,two
```

See the example for Object.toString() for an example of the return value of `Object.valueOf()` for a custom class that overrides `toString()`.

Object.watch()

Availability

Flash Player 6.

Usage

```
myObject.watch( prop:String, callback:Function [, userData:Object] ) : Boolean
```

Parameters

`prop` A string; the name of the object property to watch.

`callback` The function to invoke when the watched property changes. This parameter is a function object, not a function name as a string. The form of *`callback`* is `callback(prop, oldVal, newVal, userData)`.

`userData` An arbitrary piece of ActionScript data that is passed to the *`callback`* method. If the *`userData`* parameter is omitted, `undefined` is passed to the callback method. This parameter is optional.

Returns

A Boolean value: `true` if the watchpoint is created successfully, `false` otherwise.

Description

Method; registers an event handler to be invoked when a specified property of an ActionScript object changes. When the property changes, the event handler is invoked with `myObject` as the containing object.

Your can use the `return` statement in your *`callback`* method definition to affect the value of the property you are watching. The value returned by your *`callback`* method is assigned to the watched object property. The value you choose to return depends on whether you wish to monitor, modify or prevent changes to the property:

- If you are merely monitoring the property, return the `newVal` parameter.
- If you are modifying the value of the property, return your own value.
- If you want to prevent changes to the property, return the `oldVal` parameter.

If the *`callback`* method you define does not have a `return` statement, then the watched object property is assigned a value of `undefined.`

A watchpoint can filter (or nullify) the value assignment, by returning a modified `newval` (or `oldval`). If you delete a property for which a watchpoint has been set, that watchpoint does not disappear. If you later recreate the property, the watchpoint is still in effect. To remove a watchpoint, use the `Object.unwatch` method.

Only a single watchpoint can be registered on a property. Subsequent calls to `Object.watch()` on the same property replace the original watchpoint.

The `Object.watch()` method behaves similarly to the `Object.watch()` function in JavaScript 1.2 and later. The primary difference is the *`userData`* parameter, which is a Flash addition to `Object.watch()` that Netscape Navigator does not support. You can pass the *`userData`* parameter to the event handler and use it in the event handler.

The `Object.watch()` method cannot watch getter/setter properties. Getter/setter properties operate through *lazy evaluation*— the value of the property is not determined until the property is actually queried. Lazy evaluation is often efficient because the property is not constantly updated; it is, rather, evaluated when needed. However, `Object.watch()` needs to evaluate a property to determine whether to invoke the *`callback`* function. To work with a getter/setter property, `Object.watch()` needs to evaluate the property constantly, which is inefficient.

Generally, predefined ActionScript properties, such as `_x`, `_y`, `_width`, and `_height`, are getter/setter properties and cannot be watched with `Object.watch()`.

Example

The following example uses `watch()` to check whether the `speed` property exceeds the speed limit:

```
// Create a new object
var myObject:Object = new Object();

// Add a property that tracks speed
myObject.speed = 0;

// Write the callback function to be executed if the speed property changes
var speedWatcher:Function = function(prop, oldVal, newVal, speedLimit) {
  // Check whether speed is above the limit
  if (newVal > speedLimit) {
    trace ("You are speeding.");
  }
  else {
    trace ("You are not speeding.");
  }

  // Return the value of newVal.
  return newVal;
}
// Use watch() to register the event handler, passing as parameters:
//   - the name of the property to watch: "speed"
//   - a reference to the callback function speedWatcher
//   - the speedLimit of 55 as the userData parameter
myObject.watch("speed", speedWatcher, 55);

// set the speed property to 54, then to 57
myObject.speed = 54; // output: You are not speeding
myObject.speed = 57; // output: You are speeding

// unwatch the object
myObject.unwatch("speed");
myObject.speed = 54; // there should be no output
```

See also

`Object.addProperty()`, `Object.unwatch()`

on()

Availability

Flash 2. Not all events are supported in Flash 2.

Usage

```
on(mouseEvent) {
  // your statements here
}
```

Parameters

`statement(s)` The instructions to execute when the `mouseEvent` occurs.

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

- `press` The mouse button is pressed while the pointer is over the button.
- `release` The mouse button is released while the pointer is over the button.
- `releaseOutside` While the pointer is over the button, the mouse button is pressed and then rolls outside the button area just before it is released. Both the `press` and the `dragOut` events always precede a `releaseOutside` event.
- `rollOut` The pointer rolls outside the button area.
- `rollOver` The mouse pointer rolls over the button.
- `dragOut` While the pointer is over the button, the mouse button is pressed and then rolls outside the button area.
- `dragOver` While the pointer is over the button, the mouse button has been pressed, then rolled outside the button, and then rolled back over the button.
- `keyPress ("key")` The specified keyboard key is pressed. For the `key` portion of the parameter, specify a key code or key constant. You can use this parameter to intercept a key press, that is, to override any built-in behavior for the specified key. The button can be anywhere in your application, on or off the Stage. One limitation of this technique is that you can't apply the `on()` handler at runtime; you must do it at authoring time. For a list of key codes associated with the keys on a standard keyboard, see Appendix C, "Keyboard Keys and Key Code Values" in *Using ActionScript in Flash;* for a list of key constants, see "Property summary for the Key class" on page 341.

Description

Event handler; specifies the mouse event or keypress that triggers an action.

Example

In the following script, the `startDrag()` function executes when the mouse is pressed, and the conditional script is executed when the mouse is released and the object is dropped:

```
on (press) {
  startDrag(this);
}
```

```
on (release) {
  trace("X:"+this._x);
  trace("Y:"+this._y);
  stopDrag();
}
```

See also

`onClipEvent()`

onClipEvent()

Availability

Flash Player 5.

Usage

```
onClipEvent(movieEvent){
  // your statements here
}
```

Parameters

A `movieEvent` is a trigger called an *event*. When the event occurs, the statements following it within curly braces ({}) are executed. Any of the following values can be specified for the `movieEvent` parameter:

- `load` The action is initiated as soon as the movie clip is instantiated and appears in the Timeline.
- `unload` The action is initiated in the first frame after the movie clip is removed from the Timeline. The actions associated with the `Unload` movie clip event are processed before any actions are attached to the affected frame.
- `enterFrame` The action is triggered continually at the frame rate of the movie clip. The actions associated with the `enterFrame` clip event are processed before any frame actions that are attached to the affected frames.
- `mouseMove` The action is initiated every time the mouse is moved. Use the `_xmouse` and `_ymouse` properties to determine the current mouse position.
- `mouseDown` The action is initiated when the left mouse button is pressed.
- `mouseUp` The action is initiated when the left mouse button is released.
- `keyDown` The action is initiated when a key is pressed. Use `Key.getCode()` to retrieve information about the last key pressed.
- `keyUp` The action is initiated when a key is released. Use the `Key.getCode()` method to retrieve information about the last key pressed.
- `data` The action is initiated when data is received in a `loadVariables()` or `loadMovie()` action. When specified with a `loadVariables()` action, the `data` event occurs only once, when the last variable is loaded. When specified with a `loadMovie()` action, the `data` event occurs repeatedly, as each section of data is retrieved.

Description

Event handler; triggers actions defined for a specific instance of a movie clip.

Example

The following example uses `onClipEvent()` with the `keyDown` movie event and is designed to be attached to a movie clip or button. The `keyDown` movie event is usually used with one or more methods and properties of the Key object. The following script uses `Key.getCode()` to find out which key the user has pressed; if the pressed key matches the `Key.RIGHT` property, the playhead is sent to the next frame; if the pressed key matches the `Key.LEFT` property, the playhead is sent to the previous frame.

```
onClipEvent (keyDown) {
  if (Key.getCode() == Key.RIGHT) {
    this._parent.nextFrame();
  } else if (Key.getCode() == Key.LEFT) {
    this._parent.prevFrame();
  }
}
```

The following example uses `onClipEvent()` with the `load` and `mouseMove` movie events. The `_xmouse` and `_ymouse` properties track the position of the mouse each time the mouse moves, which appears in the text field that's created at runtime.

```
onClipEvent (load) {
  this.createTextField("coords_txt", this.getNextHighestDepth(), 0, 0, 100,
  22);
  coords_txt.autoSize = true;
  coords_txt.selectable = false;
}
onClipEvent (mouseMove) {
  coords_txt.text = "X:"+_root._xmouse+",Y:"+_root._ymouse;
}
```

See also

Key class, MovieClip._xmouse, MovieClip._ymouse, on(), `updateAfterEvent()`

onUpdate

Availability

Flash Player 6.

Usage

```
function onUpdate() {
   ...statements...;
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; `onUpdate` is defined for a Live Preview used with a component. When an instance of a component on the Stage has a Live Preview, the authoring tool invokes the Live Preview's `onUpdate` function whenever the component parameters of the component instance change. The `onUpdate` function is invoked by the authoring tool with no parameters, and its return value is ignored. The `onUpdate` function should be declared on the main Timeline of the Live Preview.

Defining an `onUpdate` function in a Live Preview is optional.

For more information on Live Preview, see *Using Components*.

Example

The `onUpdate` function gives the Live Preview an opportunity to update its visual appearance to match the new values of the component parameters. When the user changes a parameter value in the components Property inspector or Parameters tab in the Components inspector, `onUpdate` is invoked. The `onUpdate` function does something to update itself. For instance, if the component includes a `color` parameter, the `onUpdate` function might alter the color of a movie clip inside the Live Preview to reflect the new parameter value. In addition, it might store the new color in an internal variable.

The following example uses the `onUpdate` function to pass parameter values through an empty movie clip in the Live Preview. To see this code work, place a labeled button component on the Stage with a variable labelColor that specifies the color of the text label. The following code is in the first frame of the main Timeline of the component:

```
/* Define the textColor parameter variable to specify the color of the button
  label text.*/
buttonLabel.textColor = labelColor;
```

In the Live Preview, place an empty movie clip named `xch` in the Live Preview. Then place the following code in the first frame of the Live Preview. Add `"xch"` to the `labelColor` variable path to pass the variable through the `my_mc` movie clip:

```
//Write an onUpdate function, adding "my_mc." to the parameter variable names:
function onUpdate (){
  buttonLabel.textColor = my_mc.labelColor;
}
```

_parent

Availability

Flash Player 5.

Usage

```
_parent.property
_parent._parent.property
```

Description

Identifier; specifies or returns a reference to the movie clip or object that contains the current movie clip or object. The current object is the object containing the ActionScript code that references `_parent`. Use `_parent` to specify a relative path to movie clips or objects that are above the current movie clip or object.

Example

In the following example, there is a movie clip on the Stage with the instance name `square_mc`. Within that movie clip is another movie clip with an instance name `circle_mc`. The following ActionScript lets you modify the `circle_mc` parent instance (which is `square_mc`) when the circle is clicked. When you are working with relative addressing (using `_parent` instead of `_root`), it might be easier to use the Insert Target Path button in the Actions panel at first.

```
this.square_mc.circle_mc.onRelease = function() {
  this._parent._alpha -= 5;
};
```

See also

`_root`, `targetPath()`

parseFloat()

Availability

Flash Player 5.

Usage

`parseFloat(string:String) : Number`

Parameters

`string` The string to read and convert to a floating-point number.

Returns

A number or `NaN` (not a number).

Description

Function; converts a string to a floating-point number. The function reads, or *parses*, and returns the numbers in a string until it reaches a character that is not a part of the initial number. If the string does not begin with a number that can be parsed, `parseFloat()` returns `NaN`. White space preceding valid integers is ignored, as are trailing nonnumeric characters.

Example

The following examples use the `parseFloat()` function to evaluate various types of numbers:

```
trace(parseFloat("-2"));       // output: -2
trace(parseFloat("2.5"));      // output: 2.5
trace(parseFloat(" 2.5"));     // output: 2.5
trace(parseFloat("3.5e6"));    // output: 3500000
trace(parseFloat("foobar"));   // output: NaN
trace(parseFloat("3.75math")); // output: 3.75
trace(parseFloat("0garbage")); // output: 0
```

See also

`NaN`, `parseInt()`

parseInt()

Availability

Flash Player 5.

Usage

```
parseInt(expression:String [, radix:Number]) : Number
```

Parameters

`expression` A string to convert to an integer.

`radix` An integer representing the radix (base) of the number to parse. Legal values are from 2 to 36. This parameter is optional.

Returns

A number or `NaN` (not a number).

Description

Function; converts a string to an integer. If the specified string in the parameters cannot be converted to a number, the function returns `NaN`. Strings beginning with 0x are interpreted as hexadecimal numbers. Integers beginning with 0 or specifying a radix of 8 are interpreted as octal numbers. White space preceding valid integers is ignored, as are trailing nonnumeric characters.

Example

The examples in this section use the `parseInt()` function to evaluate various types of numbers.

The following example returns 3:

```
parseInt("3.5")
```

The following example returns `NaN`:

```
parseInt("bar")
```

The following example returns 4:

```
parseInt("4foo")
```

The following example shows a hexadecimal conversion that returns 1016:

```
parseInt("0x3F8")
```

The following example shows a hexadecimal conversion using the optional *radix* parameter that returns 1000:

```
parseInt("3E8", 16)
```

The following example shows a binary conversion and returns 10, which is the decimal representation of the binary 1010:

```
parseInt("1010", 2)
```

The following examples show octal number parsing and return 511, which is the decimal representation of the octal 777:

```
parseInt("0777")
parseInt("777", 8)
```

See also

`NaN`, `parseFloat()`

play()

Availability

Flash 2.

Usage

```
play() : Void
```

Parameters

None.

Returns

Nothing.

Description

Function; moves the playhead forward in the Timeline.

Example

In the following example, there are two movie clip instances on the Stage with the instance names `stop_mc` and `play_mc`. The ActionScript stops the SWF file's playback when the `stop_mc` movie clip instance is clicked. Playback resumes when the `play_mc` instance is clicked.

```
this.stop_mc.onRelease = function() {
  stop();
};
this.play_mc.onRelease = function() {
  play();
};
trace("frame 1");
```

See also

`gotoAndPlay()`, `MovieClip.gotoAndPlay()`

prevFrame()

Availability

Flash 2.

Usage

```
prevFrame() : Void
```

Parameters

None.

Returns

Nothing.

Description

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

Example

When the user clicks a button called `myBtn_btn` and the following ActionScript is placed on a frame in the Timeline for that button, the playhead is sent to the previous frame:

```
stop();
this.myBtn_btn.onRelease = function(){
  prevFrame();
};
```

See also

`MovieClip.prevFrame()`, `nextFrame()`

prevScene()

Availability

Flash 2.

Usage

```
prevScene() : Void
```

Parameters

None.

Returns

Nothing.

Description

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

See also

`nextScene()`

print()

Availability

Flash Player 4 (4.0.20.0)

Note: If you are authoring for Flash Player 7 or later, you can create a PrintJob object, which gives you (and the user) more control over the printing process. For more information, see the PrintJob class entry.

Usage

```
print(target:Object, "Bounding box":String) : Void
```

Parameters

`target` The instance name of a movie clip to print. By default, all the frames in the target instance can be printed. If you want to print specific frames in the movie clip, assign a `#p` frame label to those frames.

`Bounding box` A modifier that sets the print area of the movie clip. Enclose this parameter in quotation marks (" or '), and specify one of the following values:

- `bmovie` Designates the bounding box of a specific frame in a movie clip as the print area for all printable frames in the movie clip. Assign a `#b` frame label to the frame whose bounding box you want to use as the print area.
- `bmax` Designates a composite of all the bounding boxes of all the printable frames as the print area. Specify `bmax` when the printable frames in your movie clip vary in size.
- `bframe` Indicates that the bounding box of each printable frame should be used as the print area for that frame, which changes the print area for each frame and scales the objects to fit the print area. Use `bframe` if you have objects of different sizes in each frame and want each object to fill the printed page.

Returns

None.

Description

Function; prints the `target` movie clip according to the boundaries specified in the parameter (`bmovie`, `bmax`, or `bframe`). If you want to print specific frames in the target movie clip, attach a `#p` frame label to those frames. Although `print()` results in higher quality prints than `printAsBitmap()`, it cannot be used to print movie clips that use alpha transparencies or special color effects.

If you use `bmovie` for the `Bounding box` parameter but do not assign a `#b` label to a frame, the print area is determined by the Stage size of the loaded movie clip. (The loaded movie clip does not inherit the main movie clip's Stage size.)

All the printable elements in a movie clip must be fully loaded before printing can begin.

The Flash Player printing feature supports PostScript and non-PostScript printers. Non-PostScript printers convert vectors to bitmaps.

Example

The following example prints all the printable frames in `holder_mc` with a print area defined by the bounding box of each frame:

```
this.createEmptyMovieClip("holder_mc", 999);
holder_mc.loadMovie("http://www.macromedia.com/devnet/mx/blueprint/articles/
  nielsen/spotlight_jnielsen.jpg");

this.myBtn_btn.onRelease = function() {
  print(this._parent.holder_mc, "bframe");
};
```

In the previous ActionScript, you could replace `bframe` with `bmovie` so that the print area is defined by the bounding box of a frame with the `#b` frame label attached.

See also

`printAsBitmap()`, `printAsBitmapNum()`, `PrintJob class`, `printNum()`

printAsBitmap()

Availability

Flash Player 4 (4.0.20.0)

Note: If you are authoring for Flash Player 7 or later, you can create a PrintJob object, which gives you (and the user) more control over the printing process. For more information, see the PrintJob class entry.

Usage

`printAsBitmap(`*`target:Object`*`, "`*`Bounding box`*`":`*`String`*`) : `*`Void`*

Parameters

`target` The instance name of the movie clip to print. By default, all the frames in the movie clip are printed. If you want to print specific frames in the movie clip, attach a `#p` frame label to those frames.

`Bounding box` A modifier that sets the print area of the movie clip. Enclose this parameter in quotation marks (" or '), and specify one of the following values:

- `bmovie` Designates the bounding box of a specific frame in a movie clip as the print area for all printable frames in the movie clip. Assign a `#b` frame label to the frame whose bounding box you want to use as the print area.
- `bmax` Designates a composite of all the bounding boxes of all the printable frames as the print area. Specify the `bmax` parameter when the printable frames in your movie clip vary in size.
- `bframe` Indicates that the bounding box of each printable frame should be used as the print area for that frame. This changes the print area for each frame and scales the objects to fit the print area. Use `bframe` if you have objects of different sizes in each frame and want each object to fill the printed page.

Returns

Nothing.

Description

Function; prints the *`target`* movie clip as a bitmap according to the boundaries specified in the parameter (`bmovie`, `bmax`, or `bframe`). Use `printAsBitmap()` to print movie clips that contain frames with objects that use transparency or color effects. The `printAsBitmap()` action prints at the highest available resolution of the printer in order to maintain as much definition and quality as possible.

If your movie clip does not contain alpha transparencies or color effects, Macromedia recommends that you use print() for better quality results.

If you use `bmovie` for the *`Bounding box`* parameter but do not assign a `#b` label to a frame, the print area is determined by the Stage size of the loaded movie clip. (The loaded movie clip does not inherit the main movie clip's Stage size.)

All the printable elements in a movie clip must be fully loaded before printing can begin.

The Flash Player printing feature supports PostScript and non-PostScript printers. Non-PostScript printers convert vectors to bitmaps.

Example

The following example prints all the printable frames in `holder_mc` with a print area defined by the bounding box of the frame:

```
this.createEmptyMovieClip("holder_mc", 999);
holder_mc.loadMovie("http://www.macromedia.com/devnet/mx/blueprint/articles/
  back_button/sp_mchambers.jpg");

this.myBtn_btn.onRelease = function() {
  printAsBitmap(this._parent.holder_mc, "bframe");
};
```

See also

`print()`, `printAsBitmapNum()`, `PrintJob class`, `printNum()`

printAsBitmapNum()

Availability

Flash Player 5.

Note: If you are authoring for Flash Player 7 or later, you can create a PrintJob object, which gives you (and the user) more control over the printing process. For more information, see the PrintJob class entry.

Usage

`printAsBitmapNum(level:Number, "Bounding box":String) : Void`

Parameters

`level` The level in Flash Player to print. By default, all the frames in the level print. If you want to print specific frames in the level, assign a `#p` frame label to those frames.

`Bounding box` A modifier that sets the print area of the movie clip. Enclose this parameter in quotation marks (" or '), and specify one of the following values:

- `bmovie` Designates the bounding box of a specific frame in a movie clip as the print area for all the printable frames in the movie clip. Assign a `#b` frame label to the frame whose bounding box you want to use as the print area.
- `bmax` Designates a composite of all the bounding boxes of all the printable frames as the print area. Specify the `bmax` parameter when the printable frames in your movie clip vary in size.
- `bframe` Indicates that the bounding box of each printable frame should be used as the print area for that frame. This changes the print area for each frame and scales the objects to fit the print area. Use `bframe` if you have objects of different sizes in each frame and you want each object to fill the printed page.

Returns

None.

Description

Function; prints a level in Flash Player as a bitmap according to the boundaries specified in the parameter (`bmovie`, `bmax`, or `bframe`). Use `printAsBitmapNum()` to print movie clips that contain frames with objects that use transparency or color effects. The `printAsBitmapNum()` action prints at the highest available resolution of the printer in order to maintain the highest possible definition and quality. To calculate the printable file size of a frame designated to be printed as a bitmap, multiply pixel width by pixel height by printer resolution.

If your movie clip does not contain alpha transparencies or color effects, using `printNum()` will give you better quality results.

If you use `bmovie` for the `Bounding box` parameter but do not assign a `#b` label to a frame, the print area is determined by the Stage size of the loaded movie clip. (The loaded movie clip does not inherit the main movie's Stage size.)

All the printable elements in a movie clip must be fully loaded before printing can start.

The Flash Player printing feature supports PostScript and non-PostScript printers. Non-PostScript printers convert vectors to bitmaps.

Example

The following example prints the contents of the Stage when the user clicks the button `myBtn_btn`. The area to print is defined by the bounding box of the frame.

```
myBtn_btn.onRelease = function(){
  printAsBitmapNum(0, "bframe")
};
```

See also

print(), printAsBitmap(), PrintJob class, printNum()

PrintJob class

Availability

Flash Player 7.

Description

The PrintJob class lets you create content and print it to one or more pages. This class, in addition to offering improvements to print functionality provided by the `print()` method, lets you render dynamic content offscreen, prompt users with a single Print dialog box, and print an unscaled document with proportions that map to the proportions of the content. This capability is especially useful for rendering and printing dynamic content, such as database content and dynamic text.

Additionally, with properties populated by `PrintJob.start()`, your document can read your user's printer settings, such as page height, width, and orientation, and you can configure your document to dynamically format Flash content that is appropriate for the printer settings. These user layout properties are read-only and cannot be changed by Flash Player.

Method summary for the PrintJob class

You must use the methods for PrintJob class in the order listed in the following table:

Method	Description
`PrintJob.start()`	Displays the operating system's print dialog boxes and starts spooling.
`PrintJob.addPage()`	Adds one page to the print spooler.
`PrintJob.send()`	Sends spooled pages to the printer.

Constructor for the PrintJob class

Availability

Flash Player 7.

Usage

```
my_pj = new PrintJob() : PrintJob
```

Parameters

None.

Returns

A reference to a PrintJob object.

Description

Constructor; creates a PrintJob object that you can use to print one or more pages.

To implement a print job, use the following methods in sequence. You must place all commands relating to a specific print job in the same frame, from the constructor through `PrintJob.send()` and delete. Replace the `[params]` to the `my_pj.addPage()` method calls with your custom parameters.

```
// create PrintJob object
var my_pj:PrintJob = new PrintJob();

// display print dialog box, but only initiate the print job
// if start returns successfully.
if (my_pj.start()) {

  // use a variable to track successful calls to addPage
  var pagesToPrint:Number = 0;

  // add specified area to print job
  // repeat once for each page to be printed
  if (my_pj.addPage([params])) {
    pagesToPrint++;
  }
  if (my_pj.addPage([params])) {
    pagesToPrint++;
  }
  if (my_pj.addPage([params])) {
    pagesToPrint++;
  }

  // send pages from the spooler to the printer, but only if one or more
  // calls to addPage() was successful. You should always check for successful
  // calls to start() and addPage() before calling send().
  if (pagesToPrint > 0) {
    my_pj.send();  // print page(s)
  }
}

// clean up
delete my_pj;  // delete object
```

You cannot create a second PrintJob object while the first one is still active. You cannot create a second PrintJob object (by calling `new PrintJob()`) while the first PrintJob object is still active, the second PrintJob object will not be created.

Example

See PrintJob.addPage().

See also

`PrintJob.addPage()`, `PrintJob.send()`, `PrintJob.start()`

PrintJob.addPage()

Availability

Flash Player 7.

Usage

```
my_pj.addPage(target:Object [, printArea:Object] [, options:Object ] [,
  frameNumber:Number]) : Boolean
```

Parameters

target A number or string; the level or instance name of the movie clip to print. Pass a number to specify a level (for example, 0 is the `_root` movie), or a string (in quotation marks [""]) to specify the instance name of a movie clip.

printArea An optional object that specifies the area to print, in the following format:

`{xMin:topLeft, xMax:topRight, yMin:bottomLeft, yMax:bottomRight}`

The coordinates you specify for *printArea* represent screen pixels relative to the registration point of the `_root` movie clip (if *target* = 0) or of the level or movie clip specified by *target*. You must provide all four coordinates. The width (`xMax-xMin`) and height (`yMax-yMin`) must each be greater than 0.

Points are print units of measurement, and pixels are screen units of measurement; points are a fixed physical size (1/72 inch), but the size of a pixel depends on the resolution of the particular screen. You can use the following equivalencies to convert inches or centimeters to twips or points (a twip is 1/20 of a point):

- 1 point = 1/72 inch = 20 twips
- 1 inch = 72 points = 1440 twips
- 1 cm = 567 twips

You can't reliably convert between pixels and points; the conversion rate depends on the screen and its resolution. If the screen is set to display 72 pixels per inch, for example, one point is equal to one pixel.

Note: If you have previously used `print()`, `printAsBitmap()`, `printAsBitmapNum()`, or `printNum()` to print from Flash, you might have used a `#b` frame label to specify the area to print. When using the `addPage()` method, you must use the *printArea* parameter to specify the print area; `#b` frame labels are ignored.

If you omit the *printArea* parameter, or if it is passed incorrectly, the full Stage area of *target* is printed. If you don't want to specify a value for *printArea* but want to specify a value for *options* or *frameNumber*, pass `null` for *printArea*.

options An optional parameter that specifies whether to print as vector or bitmap, in the following format:

`{printAsBitmap:Boolean}`

The default value is `false`, which represents a request for vector printing. To print *target* as a bitmap, pass `true` for `printAsBitmap`. Remember the following suggestions when determining which value to use:

- If the content that you're printing includes a bitmap image, use `{printAsBitmap:true}` to include any transparency and color effects.
- If the content does not include bitmap images, omit this parameter or use `{printAsBitmap:false}` to print the content in higher quality vector format.

If *`options`* is omitted or is passed incorrectly, vector printing is used. If you don't want to specify a value for *`options`* but want to specify a value for *`frameNumber`*, pass `null` for *`options`*.

`frameNumber` An optional number that lets you specify which frame to print; passing a `frameNumber` does not cause the ActionScript on that frame to be invoked. If you omit this parameter, the current frame in *`target`* is printed.

Note: If you previously used `print()`, `printAsBitmap()`, `printAsBitmapNum()`, or `printNum()` to print from Flash, you might have used a `#p` frame label on multiple frames to specify which pages to print. To use `PrintJob.addPage()` to print multiple frames, you must issue a `PrintJob.addPage()` command for each frame; `#p` frame labels are ignored. For one way to do this programmatically, see the Example section.

Returns

A Boolean value: `true` if the page is successfully sent to the print spooler; `false` otherwise.

Description

Method; sends the specified level or movie clip as a single page to the print spooler. Before using this method, you must use PrintJob.start(); after calling `PrintJob.addPage()` one or more times for a print job, you must use PrintJob.send() to send the spooled pages to the printer.

If this method returns `false` (for example, if you haven't called `PrintJob.start()` or the user canceled the print job), any subsequent calls to `PrintJob.addPage()` will fail. However, if previous calls to `PrintJob.addPage()` were successful, the concluding `PrintJob.send()` command sends the successfully spooled pages to the printer.

If you passed a value for *`printArea`*, the `xMin` and `yMin` coordinates map to the upper left corner (0,0 coordinates) of the printable area on the page. The user's printable area is described by the read-only `pageHeight` and `pageWidth` properties set by `PrintJob.start()`. Because the printout aligns with the upper left corner of the printable area on the page, the printout is clipped to the right and/or bottom if the area defined in *`printArea`* is bigger than the printable area on the page. If you haven't passed a value for *`printArea`* and the Stage is larger than the printable area, the same type of clipping takes place.

For more information, see "Specifying a print area (when not using the PrintJob object)" in *Using Flash*.

If you want to scale a movie clip before you print it, set its `MovieClip._xscale` and `MovieClip._yscale` properties before calling this method, and set them back to their original values afterward. The scale of a movie clip has no relation to *`printArea`*. That is, if you specify that you print an area that is 50 x 50 pixels in size, 2500 pixels are printed. If you have scaled the movie clip, the same 2500 pixels are printed, but the movie clip is printed at the scaled size.

The Flash Player printing feature supports PostScript and non-PostScript printers. Non-PostScript printers convert vectors to bitmaps.

Example

The following example shows several ways to issue the `addPage()` command:

```
my_btn.onRelease = function()
{
  var pageCount:Number = 0;

  var my_pj:PrintJob = new PrintJob();

  if (my_pj.start())
  {
    // Print entire current frame of the _root movie in vector format
    if (my_pj.addPage(0)){
      pageCount++;

      // Starting at 0,0, print an area 400 pixels wide and 500 pixels high
      // of the current frame of the _root movie in vector format
      if (my_pj.addPage(0, {xMin:0,xMax:400,yMin:0,yMax:500})){
        pageCount++;

        // Starting at 0,0, print an area 400 pixels wide and 500 pixels high
        // of frame 1 of the _root movie in bitmap format
        if (my_pj.addPage(0, {xMin:0,xMax:400,yMin:0,yMax:500},
            {printAsBitmap:true}, 1)){
          pageCount++;

          // Starting 50 pixels to the right of 0,0 and 70 pixels down,
          // print an area 500 pixels wide and 600 pixels high
          // of frame 4 of level 5 in vector format
          if (my_pj.addPage(5, {xMin:50,xMax:550,yMin:70,yMax:670},null, 4)){
            pageCount++;

            // Starting at 0,0, print an area 400 pixels wide
            // and 400 pixels high of frame 3 of the "dance_mc" movie clip
            // in bitmap format
            if (my_pj.addPage("dance_mc",
                {xMin:0,xMax:400,yMin:0,yMax:400},{printAsBitmap:true}, 3)){
              pageCount++;

              // Starting at 0,0, print an area 400 pixels wide
              // and 600 pixels high of frame 3 of the "dance_mc" movie clip
              // in vector format at 50% of its actual size
              var x:Number = dance_mc._xscale;
              var y:Number = dance_mc._yscale;
              dance_mc._xscale = 50;
              dance_mc._yscale = 50;

              if (my_pj.addPage("dance_mc",
                  {xMin:0,xMax:400,yMin:0,yMax:600},null, 3)){
                pageCount++;
              }
              dance_mc._xscale = x;
              dance_mc._yscale = y;
            }
```

```
                    }
                }
            }
        }
    }

    // If addPage() was successful at least once, print the spooled pages.
    if (pageCount > 0){
      my_pj.send();
    }
    delete my_pj;
}
```

See also

PrintJob.send(), PrintJob.start()

PrintJob.send()

Availability

Flash Player 7.

Usage

```
my_pj.send() : Void
```

Parameters

None.

Returns

Nothing.

Description

Method; used following PrintJob.start() and PrintJob.addPage() to send spooled pages to the printer. Because calls to `PrintJob.send()` will not be successful if related calls to `PrintJob.start()` and `PrintJob.addpage()` failed, you should check that calls to `PrintJob.addpage()` and `PrintJob.start()` were successful before calling `PrintJob.send()`:

```
var my_pj:PrintJob = new PrintJob();
if (my_pj.start()) {
  if (my_pj.addPage(this)) {
    my_pj.send();
  }
}
delete my_pj;
```

Example

See PrintJob.addPage() and PrintJob.start().

See also

PrintJob.addPage(), PrintJob.start()

PrintJob.start()

Availability

Flash Player 7.

Usage

```
my_pj.start() : Boolean
```

Parameters

None.

Returns

A Boolean value: `true` if the user clicks OK when the print dialog boxes appear; `false` if the user clicks Cancel or if an error occurs.

Description

Method; displays the operating system's print dialog boxes and starts spooling. The print dialog boxes let the user change print settings. When the `PrintJob.start()` method returns successfully, the following read-only properties are populated, representing the user's print settings:

Property	Type	Units	Notes
`PrintJob.paperHeight`	Number	Points	Overall paper height.
`PrintJob.paperWidth`	Number	Points	Overall paper width.
`PrintJob.pageHeight`	Number	Points	Height of actual printable area on the page; any user-set margins are ignored.
`PrintJob.pageWidth`	Number	Points	Width of actual printable area on the page; any user-set margins are ignored.
`PrintJob.orientation`	String	N/A	"Portrait" or "landscape."

For more information, see "Specifying a print area (when not using the PrintJob object)" in *Using Flash*.

After the user clicks OK in the Print dialog box, the player begins spooling a print job to the operating system. You should issue any ActionScript commands that affect the printout, and you can use PrintJob.addPage() commands to send pages to the spooler. You can use the read-only height, width, and orientation properties this method populates to format the printout.

Because the user sees information such as "Printing page 1" immediately after clicking OK, you should call the `PrintJob.addPage()` and `PrintJob.send()` commands as soon as possible.

If this method returns `false` (for example, if the user clicks Cancel instead of OK in the operating system's Print dialog box), any subsequent calls to `PrintJob.addPage()` and `PrintJob.send()` will fail. However, if you test for this return value and don't send `PrintJob.addPage()` commands as a result, you should still delete the PrintJob object to make sure the print spooler is cleared, as shown in the following example:

```
var my_pj:PrintJob = new PrintJob();
var myResult:Boolean = my_pj.start();
  if(myResult) {
    // addPage() and send() statements here
  }
delete my_pj;
```

Example

The following example shows how you might use the value of the orientation property to adjust the printout:

```
// create PrintJob object
var my_pj:PrintJob = new PrintJob();

// display print dialog box
if (my_pj.start()) {
  // boolean to track whether addPage succeeded, change this to a counter
  // if more than one call to addPage is possible
  var pageAdded:Boolean = false;

  // check the user's printer orientation setting
  // and add appropriate print area to print job
  if (my_pj.orientation == "portrait") {
    // Here, the printArea measurements are appropriate for an 8.5" x 11"
    // portrait page.
    pageAdded = my_pj.addPage(this,{xMin:0,xMax:600,yMin:0,yMax:800});
  }
  else {
    // my_pj.orientation is "landscape".
    // Now, the printArea measurements are appropriate for an 11" x 8.5"
    // landscape page.
    pageAdded = my_pj.addPage(this,{xMin:0,xMax:750,yMin:0,yMax:600});
  }

  // send pages from the spooler to the printer
  if (pageAdded) {
    my_pj.send();
  }
}

// clean up
delete my_pj;
```

See also

PrintJob.addPage(), PrintJob.send()

printNum()

Availability

Flash Player 5.

Note: If you are authoring for Flash Player 7 or later, you can create a PrintJob object, which gives you (and the user) more control over the printing process. For more information, see the PrintJob class entry.

Usage

`printNum (level:Number, "Bounding box":String) : Void`

Parameters

`level` The level in Flash Player to print. By default, all the frames in the level print. If you want to print specific frames in the level, assign a `#p` frame label to those frames.

`Bounding box` A modifier that sets the print area of the movie clip. Enclose this parameter in quotation marks (" or '), and specify one of the following values:

- `bmovie` Designates the bounding box of a specific frame in a movie clip as the print area for all printable frames in the movie clip. Assign a `#b` frame label to the frame whose bounding box you want to use as the print area.
- `bmax` Designates a composite of all the bounding boxes of all the printable frames as the print area. Specify the `bmax` parameter when the printable frames in your movie clip vary in size.
- `bframe` Indicates that the bounding box of each printable frame should be used as the print area for that frame. This changes the print area for each frame and scales the objects to fit the print area. Use `bframe` if you have objects of different sizes in each frame and want each object to fill the printed page.

Returns

Nothing.

Description

Function; prints the level in Flash Player according to the boundaries specified in the `Bounding box` parameter (`"bmovie"`, `"bmax"`, `"bframe"`). If you want to print specific frames in the target movie clip, attach a `#p` frame label to those frames. Although using `printNum()` results in higher quality prints than using `printAsBitmapNum()`, you cannot use `printNum()` to print movies with alpha transparencies or special color effects.

If you use `bmovie` for the `Bounding box` parameter but do not assign a `#b` label to a frame, the print area is determined by the Stage size of the loaded movie clip. (The loaded movie clip does not inherit the main movie's Stage size.)

All the printable elements in a movie clip must be fully loaded before printing can begin.

The Flash Player printing feature supports PostScript and non-PostScript printers. Non-PostScript printers convert vectors to bitmaps.

See also

`print()`, `printAsBitmap()`, `printAsBitmapNum()`, PrintJob class

private

Availability

Flash Player 6.

Usage

```
class someClassName{
  private var name;
  private function name() {
    // your statements here
  }
}
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Parameters

`name` The name of the variable or function that you want to specify as private.

Description

Keyword; specifies that a variable or function is available only to the class that declares or defines it or to subclasses of that class. By default, a variable or function is available to any caller. Use this keyword if you want to restrict access to a variable or function. For more information, see "Controlling member access" in *Using ActionScript in Flash*.

You can use this keyword only in class definitions, not in interface definitions.

Example

The following example demonstrates how you can hide certain properties within a class using the `private` keyword. Create a new AS file called Login.as.

```
class Login {
  private var loginUserName:String;
  private var loginPassword:String;
  public function Login(param_username:String, param_password:String) {
    this.loginUserName = param_username;
    this.loginPassword = param_password;
  }
  public function get username():String {
    return this.loginUserName;
  }
  public function set username(param_username:String):Void {
    this.loginUserName = param_username;
  }
  public function set password(param_password:String):Void {
    this.loginPassword = param_password;
  }
}
```

In the same directory as Login.as, create a new FLA or AS document. Enter the following ActionScript in Frame 1 of the Timeline.

```
import Login:
var gus:Login = new Login("Gus", "Smith");
trace(gus.username); // output: Gus
trace(gus.password); // output: undefined
trace(gus.loginPassword); // error
```

Because `loginPassword` is a private variable, you cannot access it from outside the Login.as class file. Attempts to access the private variable generate an error message.

See also

`public`, `static`

public

Flash Player 6.

Usage

```
class someClassName{
  public var name;
  public function name() {
    // your statements here
  }
}
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Parameters

name The name of the variable or function that you want to specify as public.

Description

Keyword; specifies that a variable or function is available to any caller. Because variables and functions are public by default, this keyword is used primarily for stylistic reasons. For example, you might want to use it for reasons of consistency in a block of code that also contains private or static variables.

Example

The following example shows how you can use public variables in a class file. Create a new class file called User.as and enter the following code:

```
class User {
  public var age:Number;
  public var name:String;
}
```

Then create a new FLA or AS file in the same directory, and enter the following ActionScript in Frame 1 of the Timeline:

```
import User;
var jimmy:User = new User();
jimmy.age = 27;
jimmy.name = "jimmy";
```

If you change one of the public variables in the User class to a private variable, an error is generated when trying to access the property.

For more information, see "Controlling member access" in *Using ActionScript in Flash*.

See also

`private`, `static`

_quality

Availability

Flash Player 5.

Usage

```
_quality:String
```

Description

Property (global); sets or retrieves the rendering quality used for a movie clip. Device fonts are always aliased and therefore are unaffected by the `_quality` property.

The `_quality` property can be set to the following values:

- `"LOW"` Low rendering quality. Graphics are not anti-aliased, and bitmaps are not smoothed.
- `"MEDIUM"` Medium rendering quality. Graphics are anti-aliased using a 2 x 2 pixel grid, but bitmaps are not smoothed. Suitable for movie clips that do not contain text.
- `"HIGH"` High rendering quality. Graphics are anti-aliased using a 4 x 4 pixel grid, and bitmaps are smoothed if the movie clip is static. This is the default rendering quality setting used by Flash.
- `"BEST"` Very high rendering quality. Graphics are anti-aliased using a 4 x 4 pixel grid and bitmaps are always smoothed.

Example

The following example sets the rendering quality to `LOW`:

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

removeMovieClip()

Availability

Flash Player 4.

Usage

```
removeMovieClip(target)
```

Parameters

`target` The target path of a movie clip instance created with `duplicateMovieClip()` or the instance name of a movie clip created with MovieClip.attachMovie(), MovieClip.duplicateMovieClip(), or MovieClip.createEmptyMovieClip().

Returns

None.

Description

Function; deletes the specified movie clip.

Example

The following example creates a new movie clip called `myClip_mc` and duplicates the movie clip. The second movie clip is called `newClip_mc`. Images are loaded into both movie clips. When a button, `button_mc`, is clicked, the duplicated movie clip is removed from the Stage.

```
this.createEmptyMovieClip("myClip_mc", this.getNextHighestDepth());
myClip_mc.loadMovie("http://www.macromedia.com/devnet/mx/blueprint/articles/
  server_side/jeremy_gray.jpg");
duplicateMovieClip(this.myClip_mc, "newClip_mc", this.getNextHighestDepth());
newClip_mc.loadMovie("http://www.macromedia.com/devnet/mx/blueprint/articles/
  performance/spotlight_speterson.jpg");
newClip_mc._x = 200;
this.button_mc.onRelease = function() {
  removeMovieClip(this._parent.newClip_mc);
};
```

See also

`duplicateMovieClip()`, `MovieClip.duplicateMovieClip()`, `MovieClip.attachMovie()`, `MovieClip.removeMovieClip()`

return

Availability

Flash Player 5.

Usage

```
return[expression]
```

Parameters

`expression` A string, number, Boolean, array, or object to evaluate and return as a value of the function. This parameter is optional.

Returns

The evaluated `expression` parameter, if provided.

Description

Statement; specifies the value returned by a function. The `return` statement evaluates `expression` and returns the result as a value of the function in which it executes. The `return` statement causes execution to return immediately to the calling function. If the `return` statement is used alone, it returns `undefined`.

You can't return multiple values. If you try to do so, only the last value is returned. In the following example, `c` is returned:

```
return a, b, c ;
```

If you need to return multiple values, you might want to use an array or object instead.

Example

The following example uses the `return` statement inside the body of the `sum()` function to return the added value of the three parameters. The next line of code calls `sum()` and assigns the returned value to the variable `newValue`.

```
function sum(a:Number, b:Number, c:Number):Number {
  return (a+b+c);
}
var newValue:Number = sum(4, 32, 78);
trace(newValue);  // output: 114
```

See also

`function`

_root

Availability

Flash Player 5.

Usage

```
_root.movieClip
_root.action
_root.property
```

Parameters

movieClip The instance name of a movie clip.

action An action or method.

property A property of the MovieClip object.

Description

Identifier; specifies or returns a reference to the root movie clip Timeline. If a movie clip has multiple levels, the root movie clip Timeline is on the level containing the currently executing script. For example, if a script in level 1 evaluates `_root`, `_level1` is returned.

Specifying `_root` is the same as using the deprecated slash notation (/) to specify an absolute path within the current level.

Caution: If a movie clip that contains `_root` is loaded into another movie clip, `_root` refers to the Timeline of the loading movie clip, not the Timeline that contains `_root`. If you want to ensure that `_root` refers to the Timeline of the loaded movie clip even if it is loaded into another movie clip, use `MovieClip._lockroot`.

Example

The following example stops the Timeline of the level containing the currently executing script:

```
_root.stop();
```

The following example traces variables and instances in the scope of `_root`:

```
for (prop in _root) {
  trace("_root."+prop+" = "+_root[prop]);
}
```

See also

`MovieClip._lockroot`, `_parent`, `targetPath()`

Selection class

Availability

Flash Player 5.

Description

The Selection class lets you set and control the text field in which the insertion point is located (that is, the field that has focus). Selection-span indexes are zero-based (for example, the first position is 0, the second position is 1, and so on).

There is no constructor function for the Selection class, because there can be only one currently focused field at a time.

Method summary for the Selection class

Method	Description
`Selection.addListener()`	Registers an object to receive notification when `onSetFocus` is invoked.
`Selection.getBeginIndex()`	Returns the index at the beginning of the selection span. Returns -1 if there is no index or currently selected field.
`Selection.getCaretIndex()`	Returns the current caret (insertion point) position in the currently focused selection span. Returns -1 if there is no caret position or currently focused selection span.
`Selection.getEndIndex()`	Returns the index at the end of the selection span. Returns -1 if there is no index or currently selected field.
`Selection.getFocus()`	Returns the name of the variable for the currently focused text field. Returns `null` if there is no currently focused text field.
`Selection.removeListener()`	Removes an object that was registered with `addListener()`.
`Selection.setFocus()`	Focuses the text field associated with the specified variable.
`Selection.setSelection()`	Sets the beginning and ending indexes of the selection span.

Listener summary for the Selection class

Listener	Description
`Selection.onSetFocus`	Notified when the input focus changes.

Selection.addListener()

Availability

Flash Player 6.

Usage

`Selection.addListener(`*`newListener:Object`*`) : `*`Void`*

Parameters

`newListener`	An object with an `onSetFocus` method.

Returns

None.

Description

Method; registers an object to receive keyboard focus change notifications. When the focus changes (for example, whenever Selection.setFocus() is invoked), all listening objects registered with `addListener()` have their `onSetFocus` method invoked. Multiple objects may listen for focus change notifications. If the listener *`newListener`* is already registered, no change occurs.

Example

In the following example, you create two input text fields at runtime, setting the borders for each text field to `true`. This code creates a new (generic) ActionScript object named `focusListener`. This object defines for itself an `onSetFocus` property, to which it assigns a function. The function takes two parameters: a reference to the text field that lost focus, and one to the text field that gained focus. The function sets the `border` property of the text field that lost focus to `false`, and sets the border property of the text field that gained focus to `true`:

```
this.createTextField("one_txt", 99, 10, 10, 200, 20);
this.createTextField("two_txt", 100, 10, 50, 200, 20);
one_txt.border = true;
one_txt.type = "input";
two_txt.border = true;
two_txt.type = "input";

var focusListener:Object = new Object();
focusListener.onSetFocus = function(oldFocus_txt, newFocus_txt) {
  oldFocus_txt.border = false;
  newFocus_txt.border = true;
};
Selection.addListener(focusListener);
```

When you test the SWF file, try using Tab to move between the two text fields. Make sure that you select Control > Disable Keyboard Shortcuts so you can change focus between the two fields using Tab.

Selection.getBeginIndex()

Availability

Flash Player 5.

Usage

```
Selection.getBeginIndex() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the index at the beginning of the selection span. If no index exists or no text field currently has focus, the method returns -1. Selection span indexes are zero-based (for example, the first position is 0, the second position is 1, and so on).

Example

The following example creates a text field at runtime, and sets its properties. A context menu item is added that can be used to change the currently selected text to uppercase characters.

```
this.createTextField("output_txt", this.getNextHighestDepth(), 0, 0, 300,
  200);
output_txt.multiline = true;
output_txt.wordWrap = true;
output_txt.border = true;
output_txt.type = "input";
output_txt.text = "Enter your text here";
var my_cm:ContextMenu = new ContextMenu();
my_cm.customItems.push(new ContextMenuItem("Uppercase...", doUppercase));
function doUppercase():Void {
  var startIndex:Number = Selection.getBeginIndex();
  var endIndex:Number = Selection.getEndIndex();
  var stringToUppercase:String = output_txt.text.substring(startIndex,
  endIndex);
  output_txt.replaceText(startIndex, endIndex,
  stringToUppercase.toUpperCase());
}
output_txt.menu = my_cm;
```

An example can also be found in the Strings.fla file in the HelpExamples Folder. Typical paths to this folder are:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/

See Also

`Selection.getEndIndex()`

Selection.getCaretIndex()

Availability

Flash Player 5.

Usage

```
Selection.getCaretIndex() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the index of the blinking insertion point (caret) position. If there is no blinking insertion point displayed, the method returns -1. Selection span indexes are zero-based (for example, the first position is 0, the second position is 1, and so on).

Example

The following example creates and sets the properties of a text field at runtime. The `getCaretIndex()` method is used to return the index of the caret and display its value in another text field.

```
this.createTextField("pos_txt", this.getNextHighestDepth(), 50, 20, 100, 22);
this.createTextField("content_txt", this.getNextHighestDepth(), 50, 50, 400,
  300);
content_txt.border = true;
content_txt.type = "input";
content_txt.wordWrap = true;
content_txt.multiline = true;
content_txt.onChanged = getCaretPos;

var keyListener:Object = new Object();
keyListener.onKeyUp = getCaretPos;
Key.addListener(keyListener);

var mouseListener:Object = new Object();
mouseListener.onMouseUp = getCaretPos;
Mouse.addListener(mouseListener);

function getCaretPos() {
  pos_txt.text = Selection.getCaretIndex();
}
```

An example can also be found in the Strings.fla file in the HelpExamples Folder. Typical paths to this folder are:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/

Selection.getEndIndex()

Availability

Flash Player 5.

Usage

```
Selection.getEndIndex() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the ending index of the currently focused selection span. If no index exists, or if there is no currently focused selection span, the method returns -1. Selection span indexes are zero-based (for example, the first position is 0, the second position is 1, and so on).

Example

This example is excerpted from the Strings.fla file in the HelpExamples folder.

```
/* define the function which converts the selected text in an instance,
   and convert the string to upper or lower case. */
function convertCase(target, menuItem) {
  var beginIndex:Number = Selection.getBeginIndex();
  var endIndex:Number = Selection.getEndIndex();
  var tempString:String;
  // make sure that text is actually selected.
  if (beginIndex>-1 && endIndex>-1) {
    // set the temporary string to the text before the selected text.
    tempString = target.text.slice(0, beginIndex);
    switch (menuItem.caption) {
    case 'Uppercase...' :
      // if the user selects the "Uppercase..." context menu item, convert the
  selected text to upper case.
      tempString += target.text.substring(beginIndex,
  endIndex).toUpperCase();
      break;
    case 'Lowercase...' :
      tempString += target.text.substring(beginIndex,
  endIndex).toLowerCase();
      break;
    }
    // append the text after the selected text to the temporary string.
    tempString += target.text.slice(endIndex);
    // set the text in the target text field to the contents of the temporary
  string.
    target.text = tempString;
  }
}
```

See the Strings.fla file for the entire script. Typical paths to the HelpExamples folder are:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/

See Also

`Selection.getBeginIndex()`

Selection.getFocus()

Availability

Flash Player 5. Instance names for buttons and text fields work in Flash Player 6 and later.

Usage

```
Selection.getFocus() : String
```

Parameters

None.

Returns

A string or `null`.

Description

Method; returns a string specifying the target path of the object that has focus.

- If a TextField object has focus, and the object has an instance name, this method returns the target path of the TextField object. Otherwise, it returns the TextField's variable name.
- If a Button object or button movie clip has focus, this method returns the target path of the Button object or button movie clip.
- If neither a TextField object, Button object, Component instance, nor button movie clip has focus, this method returns `null`.

Example

The following example displays the currently focused selection's target path in a TextArea component instance. Add several component instances or button, text field and movie clip instances to the Stage. Then add the following ActionScript to your AS or FLA file.

```
var focus_ta:mx.controls.TextArea;
my_mc.onRelease = function() {};
my_btn.onRelease = function() {};

var keyListener:Object = new Object();
keyListener.onKeyDown = function() {
  if (Key.isDown(Key.SPACE)) {
    focus_ta.text = Selection.getFocus()+newline+focus_ta.text;
  }
};
Key.addListener(keyListener);
```

Test the SWF file, and use Tab to move between the instances on the Stage. Make sure you have Control > Disable Keyboard Shortcuts selected in the test environment.

See also

`Selection.onSetFocus`, `Selection.setFocus()`

Selection.onSetFocus

Availability

Flash Player 6.

Usage

```
someListener.onSetFocus = function( [ oldFocus:Object [, newFocus:Object ] ]){
  // statements;
}
```

Parameters

oldfocus	The object losing focus. This parameter is optional.
newfocus	The object receiving focus. This parameter is optional.

Description

Listener; notified when the input focus changes. To use this listener, you must create a listener object. You can then define a function for this listener and use Selection.addListener() to register the listener with the Selection object, as in the following code:

```
var someListener:Object = new Object();
someListener.onSetFocus = function () {
  // statements
}
Selection.addListener(someListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

Example

The following example demonstrates how to determine when input focus changes in a SWF file between several dynamically created text fields. Enter the following ActionScript into a FLA or AS file and then test the document:

```
this.createTextField("one_txt", 1, 0, 0, 100, 22);
this.createTextField("two_txt", 2, 0, 25, 100, 22);
this.createTextField("three_txt", 3, 0, 50, 100, 22);
this.createTextField("four_txt", 4, 0, 75, 100, 22);

for (var i in this) {
  if (this[i] instanceof TextField) {
    this[i].border = true;
    this[i].type = "input";
  }
}

this.createTextField("status_txt", this.getNextHighestDepth(), 200, 10, 300,
  100);
status_txt.html = true;
status_txt.multiline = true;

var someListener:Object = new Object();
someListener.onSetFocus = function(oldFocus, newFocus) {
  status_txt.htmlText = "<b>setFocus triggered</b>";
  status_txt.htmlText += "<textformat tabStops='[20,80]'>";
  status_txt.htmlText += "&nbsp;\toldFocus:\t"+oldFocus;
  status_txt.htmlText += "&nbsp;\tnewFocus:\t"+newFocus;
  status_txt.htmlText += "&nbsp;\tgetFocus:\t"+Selection.getFocus();
  status_txt.htmlText += "</textformat>";
};
Selection.addListener(someListener);
```

See also

`Selection.addListener()`, `Selection.setFocus()`

Selection.removeListener()

Availability

Flash Player 6.

Usage

```
Selection.removeListener(listener:Object)
```

Parameters

`listener`	The object that will no longer receive focus notifications.

Returns

If *`listener`* was successfully removed, the method returns a `true` value. If *`listener`* was not successfully removed—for example, if *`listener`* was not on the Selection object's listener list—the method returns a value of `false`.

Description

Method; removes an object previously registered with Selection.addListener().

Example

The following ActionScript dynamically creates several text field instances. When you select a text field, information displays in the Output panel. When you click the `remove_btn` instance, the listener is removed and information no longer displays in the Output panel.

```
this.createTextField("one_txt", 1, 0, 0, 100, 22);
this.createTextField("two_txt", 2, 0, 25, 100, 22);
this.createTextField("three_txt", 3, 0, 50, 100, 22);
this.createTextField("four_txt", 4, 0, 75, 100, 22);

for (var i in this) {
  if (this[i] instanceof TextField) {
    this[i].border = true;
    this[i].type = "input";
  }
}

var selectionListener:Object = new Object();
selectionListener.onSetFocus = function(oldFocus, newFocus) {
  trace("Focus shifted from "+oldFocus+" to "+newFocus);
};
Selection.addListener(selectionListener);

remove_btn.onRelease = function() {
  trace("removeListener invoked");
  Selection.removeListener(selectionListener);
};
```

See Also

`Selection.addListener()`

Selection.setFocus()

Availability

Flash Player 5. Instance names for buttons and movie clips work only in Flash Player 6 and later.

Usage

```
Selection.setFocus("instanceName":String)
```

Parameters

`instanceName` A string specifying the path to a button, movie clip, or text field instance.

Returns

A Boolean value; `true` if the focus attempt is successful, `false` if it fails.

Description

Method; gives focus to the selectable (editable) text field, button, or movie clip specified by *`instanceName`*. The *`instanceName`* parameter must be a string literal of the path to the instance. You can use dot or slash notation to specify the path. You can also use a relative or absolute path. If you are using ActionScript 2.0, you must use dot notation.

If `null` is passed, the current focus is removed.

Example

In the following example, the text field focuses on the `username_txt` text field when it is running in a browser window. If the user does not fill in one of the required text fields (`username_txt` and `password_txt`), the cursor automatically focuses in the text field that's missing data. For example, if the user does not type anything into the `username_txt` text field and clicks the submit button, an error message appears and the cursor focuses in the `username_txt` text field.

```
this.createTextField("status_txt", this.getNextHighestDepth(), 100, 70, 100,
  22);
this.createTextField("username_txt", this.getNextHighestDepth(), 100, 100,
  100, 22);
this.createTextField("password_txt", this.getNextHighestDepth(), 100, 130,
  100, 22);
this.createEmptyMovieClip("submit_mc", this.getNextHighestDepth());
submit_mc.createTextField("submit_txt", this.getNextHighestDepth(), 100, 160,
  100, 22);
submit_mc.submit_txt.autoSize = "center";
submit_mc.submit_txt.text = "Submit";
submit_mc.submit_txt.border = true;
submit_mc.onRelease = checkForm;
username_txt.border = true;
password_txt.border = true;
username_txt.type = "input";
password_txt.type = "input";
password_txt.password = true;
Selection.setFocus("username_txt");
//
function checkForm():Boolean {
  if (username_txt.text.length == 0) {
    status_txt.text = "fill in username";
    Selection.setFocus("username_txt");
    return false;
  }
  if (password_txt.text.length == 0) {
    status_txt.text = "fill in password";
    Selection.setFocus("password_txt");
    return false;
  }
  status_txt.text = "success!";
  Selection.setFocus(null);
  return true;
}
```

See also

`Selection.getFocus()`, `Selection.onSetFocus`

Selection.setSelection()

Availability

Flash Player 5.

Usage

```
Selection.setSelection(start:Number, end:Number) : Void
```

Parameters

start The beginning index of the selection span.

end The ending index of the selection span.

Returns

Nothing.

Description

Method; sets the selection span of the currently focused text field. The new selection span will begin at the index specified in the `start` parameter, and end at the index specified in the `end` parameter. Selection span indexes are zero-based (for example, the first position is 0, the second position is 1, and so on). This method has no effect if there is no currently focused text field.

Example

In the following ActionScript, you create a text field at runtime and add a string to it. Then you focus the text field and select a span of characters in the focused text field.

```
this.createTextField("myText_txt", 99, 10, 10, 200, 30);
myText_txt.text = "this is my text";
Selection.setFocus("myText_txt");
Selection.setSelection(0, 3);
```

set

Availability

Flash Player 6.

Usage

```
function set property(varName) {
  // your statements here
}
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Parameters

property Word that refers to the property that `set` will access; this value must be the same as the value used in the corresponding `get` command.

varName The local variable that sets the value you're assigning.

Returns

Nothing.

Description

Keyword; permits implicit setting of properties associated with objects based on classes you have defined in external class files. Using implicit set methods lets you modify the value of an object's property without accessing the property directly. Implicit get/set methods are syntactic shorthand for the `Object.addProperty()` method in ActionScript 1. For more information, see "Implicit getter/setter methods" in *Using ActionScript in Flash*.

Example

The following example creates a Login class that demonstrates how the `set` keyword can be used to set private variables:

```
class Login {
  private var loginUserName:String;
  private var loginPassword:String;
  public function Login(param_username:String, param_password:String) {
    this.loginUserName = param_username;
    this.loginPassword = param_password;
  }
  public function get username():String {
    return this.loginUserName;
  }
  public function set username(param_username:String):Void {
    this.loginUserName = param_username;
  }
  public function set password(param_password:String):Void {
    this.loginPassword = param_password;
  }
}
```

In a FLA or AS file that is in the same directory as Login.as, enter the following ActionScriptin Frame 1 of the Timeline:

```
var gus:Login = new Login("Gus", "Smith");
trace(gus.username); // output: Gus
gus.username = "Rupert";
trace(gus.username); // output: Rupert
```

In the following example, the `get` function executes when the value is traced. The `set` function triggers only when you pass it a value, as shown in the line:

```
gus.username = “Rupert”;
```

See also

`get`, `Object.addProperty()`

set variable

Availability

Flash Player 4.

Usage

```
set("variableString", expression)
```

Parameters

`variableString` A string that names a variable to hold the value of the `expression` parameter.

`expression` A value assigned to the variable.

Returns

Nothing.

Description

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

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

Strict data typing is not supported inside a `set` statement. If you use this statement to set a variable to a value whose data type is different from the data type associated with the variable in a class file, no compiler error is generated.

A subtle but important distinction to bear in mind is that the parameter `variableString` is a string, not a variable name. If you pass an existing variable name as the first parameter to `set()` without enclosing the name in quotation marks (`""`), the variable is evaluated before the value of `expression` is assigned to it. For example, if you create a string variable named `myVariable` and assign it the value "Tuesday," and then forget to use quotation marks, you will inadvertently create a new variable named `Tuesday` that contains the value you intended to assign to `myVariable`:

```
var myVariable:String = "Tuesday";
set (myVariable, "Saturday");
trace(myVariable); // outputs Tuesday
trace(Tuesday); // outputs Saturday
```

You can avoid this situation by using quotation marks (""):

```
set ("myVariable", "Saturday");
trace(myVariable); //outputs Saturday
```

Example

In the following example, you assign a value to a variable. You are assigning the value of `"Jakob"` to the `name` variable.

```
set("name", "Jakob");
trace(name);
```

The following code loops three times and creates three new variables, called `caption0`, `caption1`, and `caption2`:

```
for (var i = 0; i<3; i++) {
  set("caption"+i, "this is caption "+i);
}
trace(caption0);
trace(caption1);
trace(caption2);
```

See also

`var`, `call()`

setInterval()

Availability

Flash Player 6.

Usage

```
setInterval(functionName:Function, interval:Number [, param1:Object, param2,
  ..., paramN]) : Number
```

Parameters

`functionName` A function name or a reference to an anonymous function.

`interval` The time in milliseconds between calls to the *`functionName`* parameter.

`param1, param2, ..., paramN` Optional parameters passed to the *`function`* or *`methodName`* parameter.

Returns

An identifying integer that you can pass to `clearInterval()` to cancel the interval.

Description

Function; calls a function or a method or an object at periodic intervals while a SWF file plays. You can use an interval function to update variables from a database or to update a time display.

If *`interval`* is less than the SWF file's frame rate (for example, 10 frames per second [fps] is equal to 100 milliseconds), the interval function is called as close to *`interval`* as possible. You must use the `updateAfterEvent()` function to make sure that the screen refreshes often enough. If *`interval`* is greater than the SWF file's frame rate, the interval function is only called each time the playhead enters a frame; this minimizes the impact each time the screen is refreshed.

Example

Usage 1: The following example calls an anonymous function every 1000 milliseconds (1 second).

```
setInterval( function(){ trace("interval called"); }, 1000 );
```

Usage 2: The following example defines two event handlers and calls each of them. Both calls to `setInterval()` send the string `"interval called"` to the Output panel every 1000 milliseconds.The first call to `setInterval()` calls the `callback1()` function, which contains a `trace()` statement. The second call to setInterval() passes the `"interval called"` string to the function `callback2()` as a parameter.

```
function callback1() {
  trace("interval called");
}

function callback2(arg) {
  trace(arg);
}

setInterval( callback1, 1000 );
setInterval( callback2, 1000, "interval called" );
```

Usage 3: This example uses a method of an object. You must use this syntax when you want to call a method that is defined for an object.

```
obj = new Object();
obj.interval = function() {
  trace("interval function called");
}

setInterval( obj, "interval", 1000 );

obj2 = new Object();
obj2.interval = function(s) {
  trace(s);
}
setInterval( obj2, "interval", 1000, "interval function called" );
```

You must use the second form of the setInterval() syntax to call a method of an object, as shown in the following example:

```
setInterval( obj2, "interval", 1000, "interval function called" );
```

When working with this function, you need to be careful about the memory you use in a SWF file. For example, when you remove a movie clip from the SWF file, it will not remove any `setInterval()` function running within it. Always remove the `setInterval()` function by using `clearInterval()` when you have finished using it, as shown in the following example:

```
// create an event listener object for our MovieClipLoader instance
var listenerObjectbject = new Object();
listenerObject.onLoadInit = function(target_mc:MovieClip) {
  trace("start interval");
  /* after the target movie clip loaded, create a callback which executes
  about every 1000 ms (1 second) and calls the intervalFunc function. */
  target_mc.myInterval = setInterval(intervalFunc, 1000, target_mc);
};
function intervalFunc(target_mc) {
  // display a trivial message which displays the instance name and arbitrary
  text.
  trace(target_mc+" has been loaded for "+getTimer()/1000+" seconds.");
  /* when the target movie clip is clicked (and released) you clear the
  interval and remove the movie clip. If you don't clear the interval before
  deleting the movie clip, the function still calls itself every second even
  though the movie clip instance is no longer present. */
  target_mc.onRelease = function() {
    trace("clear interval");
    // clearInterval(this.myInterval);
    // delete the target movie clip
    removeMovieClip(this);
  };
}
var jpeg_mcl:MovieClipLoader = new MovieClipLoader();
jpeg_mcl.addListener(listenerObject);
jpeg_mcl.loadClip("http://www.macromedia.com/software/central/images/
  klynch_breezo.jpg", this.createEmptyMovieClip("jpeg_mc",
  this.getNextHighestDepth()));
```

If you work with `setInterval()` within classes, you need to be sure to use the `this` keyword when you call the function. The `setInterval()` function does not have access to class members if you do not use the keyword. This is illustrated in the following example. For a FLA file with a button called `deleteUser_btn`, add the following ActionScript to Frame 1:

```
var me:User = new User("Gary");
this.deleteUser_btn.onRelease = function() {
  trace("Goodbye, "+me.username);
  delete me;
};
```

Then create a FLA in the same directory called User.as. Enter the following ActionScript:

```
class User {
  var intervalID:Number;
  var username:String;
  function User(param_username:String) {
    trace("Welcome, "+param_username);
    this.username = param_username;
    this.intervalID = setInterval(this, "traceUsername", 1000, this.username);
  }
  function traceUsername(str:String) {
    trace(this.username+" is "+getTimer()/1000+" seconds old, happy 
  birthday.");
  }
}
```

See also

`clearInterval()`, `updateAfterEvent()`

setProperty()

Availability

Flash Player 4.

Usage

`setProperty(`*`target:Object`*`, `*`property:Object`*`, `*`value/expression:Object`*`) : `*`Void`*

Parameters

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

`property` The property to be set.

`value` The new literal value of the property.

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

Returns

Nothing.

Description

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

Example

The following ActionScript creates a new movie clip and loads an image into it. The `_x` and `_y` coordinates are set for the clip using `setProperty()`. When you click the button called `right_btn`, the `_x` coordinate of a movie clip named `params_mc` is incremented by 20 pixels.

```
this.createEmptyMovieClip("params_mc", 999);
params_mc.loadMovie("http://www.macromedia.com/devnet/mx/blueprint/articles/
  nielsen/spotlight_jnielsen.jpg");
setProperty(this.params_mc, _y, 20);
setProperty(this.params_mc, _x, 20);
this.right_btn.onRelease = function() {
  setProperty(params_mc, _x, getProperty(params_mc, _x)+20);
};
```

See also

`getProperty()`

SharedObject class

Availability

Flash Player 6.

Description

Shared objects are powerful: They offer real-time data sharing between objects that are persistent on the user's computer. You can consider local shared objects as *cookies*.

You can use local shared objects to maintain local persistence. For example, you can call `SharedObject.getLocal()` to create a shared object, such as a calculator with memory, in the player. Because the shared object is locally persistent, Flash saves its data attributes on the user's computer when the SWF file ends. The next time the SWF file runs, the calculator will display the values it had when the SWF file ended. Alternatively, if you set the shared object's properties to `null` before the SWF file ends, the calculator opens without any prior values the next time the SWF file runs.

To create a local shared object, use the following syntax:

```
// Create a local shared object
var so:SharedObject = SharedObject.getLocal("foo");
```

Local disk space considerations

Local shared objects are always persistent on the client, depending on available memory and disk space.

By default, Flash can save locally persistent remote shared objects as large as 100K. When you try to save a larger object, Flash Player shows the Local Storage dialog box, which lets the user allow or deny local storage for the domain that is requesting access. (Ensure that your Stage size is at least 215 x 138 pixels; this is the minimum size Flash requires to display the dialog box.)

If the user clicks Allow, the object is saved and `SharedObject.onStatus` is invoked with a `code` property of `SharedObject.Flush.Success`; if the user clicks Deny, the object is not saved and `SharedObject.onStatus` is invoked with a `code` property of `SharedObject.Flush.Failed`.

The user can also specify permanent local storage settings for a particular domain by right-clicking (Windows) or Control-clicking (Macintosh) while a SWF file is running, selecting Settings, and then opening the Local Storage panel.

You can't use ActionScript to specify local storage settings for a user, but you can display the Local Storage panel for the user by using `System.showSettings(1)`.

The following list summarizes how the user's disk space choices interact with shared objects:

- If the user selects Never, objects are not saved locally and all `SharedObject.flush()` commands issued for the object return `false`.
- If the user selects Unlimited (moves the slider all the way to the right), objects are saved locally, as available disk space allows.

- If the user selects None (moves the slider all the way to the left), all `SharedObject.flush()` commands issued for the object return `"pending"`, and the player asks the user if additional disk space can be allotted to make room for the object.
- If the user selects 10K, 100K, 1 MB, or 10 MB, objects are saved locally and `SharedObject.flush()` returns `true` if the object fits within the specified amount of space. If more space is needed, `SharedObject.flush()` returns `"pending"`, and the player asks the user if additional disk space can be allotted to make room for the object.

Additionally, if the user selects a value that is less than the amount of disk space currently being used for locally persistent data, the player warns the user that any locally saved shared objects will be deleted.

Note: There is no size limit in Flash Player that runs from the authoring environment.

Method summary for the SharedObject class

Method	Description
`SharedObject.clear()`	Purges all the data from the shared object and deletes the shared object from the disk.
`SharedObject.flush()`	Immediately writes a locally persistent shared object to a local file.
`SharedObject.getLocal()`	Returns a reference to a locally persistent shared object that is available only to the current client.
`SharedObject.getSize()`	Gets the current size of the shared object, in bytes.

Property summary for the SharedObject class

Property	Description
`SharedObject.data`	The collection of attributes assigned to the `data` property of the object; these attributes can be shared and/or stored.

Event handler summary for the SharedObject class

Event handler	Description
`SharedObject.onStatus`	Invoked every time an error, warning, or informational note is posted for a shared object.

Constructor for the SharedObject class

For information on creating local shared objects, see `SharedObject.getLocal()`.

SharedObject.clear()

Availability

Flash Player 7.

Usage

my_so.clear() *: Void*

Parameters

None.

Returns

Nothing.

Description

Method; purges all the data from the shared object and deletes the shared object from the disk. The reference to *my_so* is still active, and *my_so* is now empty.

Example

The following example sets data in the shared object, and then empties all of the data from the shared object.

```
var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.data.name = "Hector";
trace("before my_so.clear():");
for (var prop in my_so.data) {
  trace("\t"+prop);
}
trace("");
my_so.clear();
trace("after my_so.clear():");
for (var prop in my_so.data) {
  trace("\t"+prop);
}
```

This ActionScript displays the following message in the Output panel:

```
before my_so.clear():
  name

after my_so.clear():
```

SharedObject.data

Availability

Flash Player 6.

Usage

myLocalSharedObject.data:*Object*

Description

Property; the collection of attributes assigned to the `data` property of the object; these attributes can be shared and/or stored. Each attribute can be an object of any basic ActionScript or JavaScript type—Array, Number, Boolean, and so on. For example, the following lines assign values to various aspects of a shared object:

```
var items_array:Array = new Array(101, 346, 483);
var currentUserIsAdmin:Boolean = true;
var currentUserName:String = "Ramona";

var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.data.itemNumbers = items_array;
my_so.data.adminPrivileges = currentUserIsAdmin;
my_so.data.userName = currentUserName;

for (var prop in my_so.data) {
  trace(prop+": "+my_so.data[prop]);
}
```

All attributes of a shared object's `data` property are saved if the object is persistent, and the shared object contains the following information:

```
userName: Ramona
adminPrivileges: true
itemNumbers: 101,346,483
```

Note: Do not assign values directly to the `data` property of a shared object, as in `so.data = someValue`; Flash ignores these assignments.

To delete attributes for local shared objects, use code such as `delete so.data.attributeName`; setting an attribute to `null` or `undefined` for a local shared object does not delete the attribute.

To create *private* values for a shared object—values that are available only to the client instance while the object is in use and are not stored with the object when it is closed—create properties that are not named `data` to store them, as shown in the following example:

```
var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.favoriteColor = "blue";
my_so.favoriteNightClub = "The Bluenote Tavern";
my_so.favoriteSong = "My World is Blue";

for (var prop in my_so) {
  trace(prop+": "+my_so[prop]);
}
```

The shared object contains the following data:

```
favoriteSong: My World is Blue
favoriteNightClub: The Bluenote Tavern
favoriteColor: blue
data: [object Object]
```

Example

The following example saves text from a TextInput component instance to a shared object named `my_so` (for the complete example, see SharedObject.getLocal()):

```
// create listener object and function for <enter> event
var textListener:Object = new Object();
textListener.enter = function(eventObj:Object) {
  my_so.data.myTextSaved = eventObj.target.text;
  my_so.flush();
};
```

SharedObject.flush()

Availability

Flash Player 6.

Usage

`myLocalSharedObject.flush([minimumDiskSpace:Number]) : Boolean`

Parameters

`minimumDiskSpace` An optional integer specifying the number of bytes that must be allotted for this object. The default value is 0.

Returns

A Boolean value: `true` or `false`, or a string value of `"pending"`, as described in the following list:

- If the user has permitted local information storage for objects from this domain, and the amount of space allotted is sufficient to store the object, this method returns `true`. (If you have passed a value for `minimumDiskSpace`, the amount of space allotted must be at least equal to that value for `true` to be returned).
- If the user has permitted local information storage for objects from this domain, but the amount of space allotted is not sufficient to store the object, this method returns `"pending"`.
- If the user has permanently denied local information storage for objects from this domain, or if Flash cannot save the object for any reason, this method returns `false`.

Description

Method; immediately writes a locally persistent shared object to a local file. If you don't use this method, Flash writes the shared object to a file when the shared object session ends—that is, when the SWF file is closed, when the shared object is garbage-collected because it no longer has any references to it or when you call `SharedObject.clear()`.

If this method returns `"pending"`, the Flash Player shows a dialog box asking the user to increase the amount of disk space available to objects from this domain. To allow space for the shared object to "grow" when it is saved in the future, which avoids return values of `"pending"`, pass a value for `minimumDiskSpace`. When Flash tries to write the file, it looks for the number of bytes passed to `minimumDiskSpace`, instead of looking for enough space to save the shared object at its current size.

For example, if you expect a shared object to grow to a maximum size of 500 bytes, even though it might start out much smaller, pass 500 for `minimumDiskSpace`. If Flash asks the user to allot disk space for the shared object, it will ask for 500 bytes. After the user allots the requested amount of space, Flash won't have to ask for more space on future attempts to flush the object (as long as its size doesn't exceed 500 bytes).

After the user responds to the dialog box, this method is called again and returns either `true` or `false`; `SharedObject.onStatus` is also invoked with a `code` property of `SharedObject.Flush.Success` or `SharedObject.Flush.Failed`.

For more information, see "Local disk space considerations" on page 735.

Example

The following function gets a shared object, `my_so`, and fills writable properties with user-provided settings. Finally, `flush()` is called to save the settings and allot a minimum of 1000 bytes of disk space.

```
this.syncSettingsCore = function(soName:String, override:Boolean,
  settings:Object) {
  var my_so:SharedObject = SharedObject.getLocal(soName, "http://
  www.mydomain.com/app/sys");
  // settings list index
  var i;
  // For each specified value in settings:
  // If override is true, set the persistent setting to the provided value.
  // If override is false, fetch the persistent setting, unless there
  // isn't one, in which case, set it to the provided value.
  for (i in settings) {
    if (override || (my_so.data[i] == null)) {
      my_so.data[i] = settings[i];
    } else {
      settings[i] = my_so.data[i];
    }
  }
  my_so.flush(1000);
};
```

SharedObject.getLocal()

Availability

Flash Player 6.

Usage

`SharedObject.getLocal(objectName:String [, localPath:String]) : SharedObject`

Note: The correct syntax is `SharedObject.getLocal`. To assign the object to a variable, use syntax like `myLocal_so = SharedObject.getLocal`.

Parameters

`objectName` A string that represents the name of the object. The name can include forward slashes (`/`); for example, `work/addresses` is a legal name. Spaces are not allowed in a shared object name, nor are the following characters:

`~ % & \ ; :  " ' , < > ? #`

`localPath` An optional string parameter that specifies the full or partial path to the SWF file that created the shared object, and that determines where the shared object will be stored locally. The default value is the full path. See the Description section for a discussion of this parameter's importance.

Returns

A reference to a shared object that is persistent locally and is available only to the current client. If Flash can't create or find the shared object (for example, if *`localPath`* was specified but no such directory exists), this method returns `null`.

Description

Method; returns a reference to a locally persistent shared object that is available only to the current client.

Note: If the user has selected to never allow local storage for this domain, the object is not saved locally, even if a value for *`localPath`* is specified. For more information, see "Local disk space considerations" on page 735.

To avoid name collisions, Flash looks at the location of the SWF file that is creating the shared object. For example, if a SWF file at www.myCompany.com/apps/stockwatcher.swf creates a shared object named `portfolio`, that shared object will not conflict with another object named `portfolio` that was created by a SWF file at www.yourCompany.com/photoshoot.swf because the SWF files originate from different directories.

Although the `localPath` parameter is optional, developers should give some thought to its use, especially if other SWF files will need to access the shared object. If the data in the shared object are specific to one SWF file that will not be moved to another location, then use of the default value makes sense. If other SWF files need access to the shared object or if the SWF file that creates the shared object will later be moved, then the value of this parameter can have a profound effect on whether any SWF files will be able to access the shared object. For example, if you create a shared object with `localPath` set to the default value of the full path to the SWF file, then no other SWF file will be able to access that shared object. If you later move the original SWF file to another location, then not even that SWF file will be able to access the data already stored in the shared object.

You can reduce the likelihood that you will inadvertently restrict access to a shared object by using the `localpath` parameter. The most permissive option is to set the `localPath` parameter to "/", which makes the shared object available to all SWF files in the domain, but will increase the likelihood of name collisions with other shared objects in the domain. More restrictive options are available to the extent that you can append the `localPath` parameter with folder names that are contained in the full path to the SWF file. For example, your `localPath` parameter options for the `portfolio` shared object created by the SWF file at www.myCompany.com/apps/stockwatcher.swf are: "/"; "/apps"; or "/apps/stockwatcher.swf". You will need to determine which option provides enough flexibility for your application.

Example

The following example creates a shared object that stores text typed into a TextInput component instance. The resulting SWF file will load the saved text from the shared object when it starts playing. Every time the user presses Enter, the text in the text field is written to the shared object. To use this example, drag a TextInput component onto the Stage, and name the instance `myText_ta`. Copy the following code into the main Timeline (click on an empty area of the Stage or press Escape to remove focus from the component):

```
// create the shared object and set localpath to server root
var my_so:SharedObject = SharedObject.getLocal("savedText", "/");
// load saved text from shared object into myText_ti TextInput component
myText_ti.text = my_so.data.myTextSaved;
// assign an empty string to myText_ti if the shared object is undefined
// to prevent the text input box from displaying "undefined" when
// this script is first run.
if (myText_ti.text == undefined) {
  myText_ti.text = "";
}
// create listener object and function for <enter> event
var textListener:Object = new Object();
textListener.enter = function(eventObj:Object) {
  my_so.data.myTextSaved = eventObj.target.text;
  my_so.flush();
};
// register listener with TextInput component instance
myText_ti.addEventListener("enter", textListener);
```

The following example saves the last frame a user entered to a local shared object `kookie`:

```
// Get the kookie
var my_so:SharedObject = SharedObject.getLocal("kookie");
```

```
// Get the user of the kookie and go to the frame number saved for this user.
if (my_so.data.user != undefined) {
  this.user = my_so.data.user;
  this.gotoAndStop(my_so.data.frame);
}
```

The following code block is placed on each SWF file frame:

```
// On each frame, call the rememberme function to save the frame number.
function rememberme() {
  my_so.data.frame=this._currentframe;
  my_so.data.user="John";
}
```

SharedObject.getSize()

Availability

Flash Player 6.

Usage

myLocalSharedObject.getSize() *: Number*

Parameters

None.

Returns

A numeric value specifying the size of the shared object, in bytes.

Description

Method; gets the current size of the shared object, in bytes.

Flash calculates the size of a shared object by stepping through each of its data properties; the more data properties the object has, the longer it takes to estimate its size. For this reason, estimating object size can have significant processing time. Therefore, you might want to avoid using this method unless you have a specific need for it.

Example

The following example gets the size of the shared object `my_so`:

```
var items_array:Array = new Array(101, 346, 483);
var currentUserIsAdmin:Boolean = true;
var currentUserName:String = "Ramona";

var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.data.itemNumbers = items_array;
my_so.data.adminPrivileges = currentUserIsAdmin;
my_so.data.userName = currentUserName;

var soSize:Number = my_so.getSize();
trace(soSize);
```

SharedObject.onStatus

Availability

Flash Player 6.

Usage

```
myLocalSharedObject.onStatus = function(infoObject:Object) {
  // your statements here
}
```

Parameters

`infoObject` A parameter defined according to the status message.

Returns

Nothing.

Description

Event handler; invoked every time an error, warning, or informational note is posted for a shared object. If you want to respond to this event handler, you must create a function to process the information object generated by the shared object.

The information object has a `code` property containing a string that describes the result of the `onStatus` handler, and a `level` property containing a string that is either `"Status"` or `"Error"`.

In addition to this `onStatus` handler, Flash also provides a *super* function called System.onStatus. If `onStatus` is invoked for a particular object and no function is assigned to respond to it, Flash processes a function assigned to `System.onStatus`, if it exists.

The following events notify you when certain SharedObject activities occur:

Code property	Level property	Meaning
`SharedObject.Flush.Failed`	Error	A SharedObject.flush() command that returned `"pending"` has failed (the user did not allot additional disk space for the shared object when Flash Player showed the Local Storage Settings dialog box).
`SharedObject.Flush.Success`	Status	A SharedObject.flush() command that returned `"pending"` has been successfully completed (the user allotted additional disk space for the shared object).

Example

The following example displays different messages based on whether the user chooses to allow or deny the SharedObject object instance to write to the disk.

```
var message_str:String;
this.createTextField("message_txt", this.getNextHighestDepth(), 0, 0, 300,
  22);
message_txt.html = true;
```

```
this.createTextField("status_txt", this.getNextHighestDepth(), 10, 30, 300,
  100);
status_txt.multiline = true;
status_txt.html = true;

var items_array:Array = new Array(101, 346, 483);
var currentUserIsAdmin:Boolean = true;
var currentUserName:String = "Ramona";
var my_so:SharedObject = SharedObject.getLocal("superfoo");
my_so.data.itemNumbers = items_array;
my_so.data.adminPrivileges = currentUserIsAdmin;
my_so.data.userName = currentUserName;

my_so.onStatus = function(infoObject:Object) {
  status_txt.htmlText = "<textformat tabStops='[50]'>";
  for (var i in infoObject) {
    status_txt.htmlText += "<b>"+i+"</b>"+"\t"+infoObject[i];
  }
  status_txt.htmlText += "</textformat>";
};
var flushResult = my_so.flush(1000001);
switch (flushResult) {
case 'pending' :
  message_str = "flush is pending, waiting on user interaction.";
  break;
case true :
  message_str = "flush was successful. Requested storage space approved.";
  break;
case false :
  message_str = "flush failed. User denied request for additional storage.";
  break;
}
message_txt.htmlText = "<a
  href=\"asfunction:System.showSettings,1\"><u>"+message_str+"</u></a>";
```

See also

`SharedObject.getLocal()`, `System.onStatus`

Sound class

Availability

Flash Player 5.

Description

The Sound class lets you control sound in a movie. You can add sounds to a movie clip from the library while the movie is playing and control those sounds. If you do not specify a target when you create a new Sound object, you can use the methods to control sound for the whole movie.

You must use the constructor `new Sound` to create a Sound object before calling the methods of the Sound class.

Method summary for the Sound class

Method	Description
`Sound.attachSound()`	Attaches the sound specified in the parameter.
`Sound.getBytesLoaded()`	Returns the number of bytes loaded for the specified sound.
`Sound.getBytesTotal()`	Returns the size of the sound in bytes.
`Sound.getPan()`	Returns the value of the previous `setPan()` call.
`Sound.getTransform()`	Returns the value of the previous `setTransform()` call.
`Sound.getVolume()`	Returns the value of the previous `setVolume()` call.
`Sound.loadSound()`	Loads an MP3 file into Flash Player.
`Sound.setPan()`	Sets the left/right balance of the sound.
`Sound.setTransform()`	Sets the amount of each channel, left and right, to be played in each speaker.
`Sound.setVolume()`	Sets the volume level for a sound.
`Sound.start()`	Starts playing a sound from the beginning or, optionally, from an offset point set in the parameter.
`Sound.stop()`	Stops the specified sound or all sounds currently playing.

Property summary for the Sound class

Property	Description
`Sound.duration`	Read-only; the length of a sound, in milliseconds.
`Sound.id3`	Read-only; provides access to the metadata that is part of an MP3 file.
`Sound.position`	Read-only; the number of milliseconds a sound has been playing.

Event handler summary for the Sound class

Event handler	Description
`Sound.onID3`	Invoked each time new ID3 data is available.
`Sound.onLoad`	Invoked when a sound loads.
`Sound.onSoundComplete`	Invoked when a sound stops playing.

Constructor for the Sound class

Availability

Flash Player 5.

Usage

```
new Sound([target:Object]) : Sound
```

Parameters

`target` The movie clip instance on which the Sound object operates. This parameter is optional.

Returns

A reference to a Sound object.

Description

Constructor; creates a new Sound object for a specified movie clip. If you do not specify a target instance, the Sound object controls all of the sounds in the movie.

Example

The following example creates a new Sound object called `global_sound`. The second line calls `setVolume()` and adjusts the volume on all sounds in the movie to 50%.

```
var global_sound:Sound = new Sound();
global_sound.setVolume(50);
```

The following example creates a new Sound object, passes it the target movie clip *`my_mc`*, and calls the *`start`* method, which starts any sound in *`my_mc`*.

```
var movie_sound:Sound = new Sound(my_mc);
movie_sound.start();
```

Sound.attachSound()

Availability

Flash Player 5.

Usage

my_sound.attachSound("*idName*":*String*) : *Void*

Parameters

idName The identifier of an exported sound in the library. The identifier is located in the Linkage Properties dialog box.

Returns

Nothing.

Description

Method; attaches the sound specified in the *idName* parameter to the specified Sound object. The sound must be in the library of the current SWF file and specified for export in the Linkage Properties dialog box. You must call `Sound.start()` to start playing the sound.

To make sure that the sound can be controlled from any scene in the SWF file, place the sound on the main Timeline of the SWF file.

Example

The following example attaches the sound `logoff_id` to `my_sound`. A sound in the library has the linkage identifier `logoff_id`.

```
var my_sound:Sound = new Sound();
my_sound.attachSound("logoff_id");
my_sound.start();
```

Sound.duration

Availability

Flash Player 6.

Usage

```
my_sound.duration:Number
```

Description

Read-only property; the duration of a sound, in milliseconds.

Example

The following example loads a sound and displays the duration of the sound file in the Output panel. Add the following ActionScript to your FLA or AS file.

```
var my_sound:Sound = new Sound();
my_sound.onLoad = function(success:Boolean) {
  var totalSeconds:Number = this.duration/1000;
  trace(this.duration+" ms ("+Math.round(totalSeconds)+" seconds)");
  var minutes:Number = Math.floor(totalSeconds/60);
  var seconds = Math.floor(totalSeconds)%60;
  if (seconds<10) {
    seconds = "0"+seconds;
  }
  trace(minutes+":"+seconds);
};
my_sound.loadSound("song1.mp3", true);
```

The following example loads several songs into a SWF file. A progress bar, created using the Drawing API, displays the loading progress. When the music starts and completes loading, information displays in the Output panel. Add the following ActionScript to your FLA or AS file.

```
var pb_height:Number = 10;
var pb_width:Number = 100;
var pb:MovieClip = this.createEmptyMovieClip("progressBar_mc",
  this.getNextHighestDepth());
pb.createEmptyMovieClip("bar_mc", pb.getNextHighestDepth());
pb.createEmptyMovieClip("vBar_mc", pb.getNextHighestDepth());
pb.createEmptyMovieClip("stroke_mc", pb.getNextHighestDepth());
pb.createTextField("pos_txt", pb.getNextHighestDepth(), 0, pb_height,
  pb_width, 22);

pb._x = 100;
pb._y = 100;

with (pb.bar_mc) {
  beginFill(0x00FF00);
  moveTo(0, 0);
  lineTo(pb_width, 0);
  lineTo(pb_width, pb_height);
  lineTo(0, pb_height);
  lineTo(0, 0);
  endFill();
```

```
    _xscale = 0;
  }
  with (pb.vBar_mc) {
    lineStyle(1, 0x000000);
    moveTo(0, 0);
    lineTo(0, pb_height);
  }
  with (pb.stroke_mc) {
    lineStyle(3, 0x000000);
    moveTo(0, 0);
    lineTo(pb_width, 0);
    lineTo(pb_width, pb_height);
    lineTo(0, pb_height);
    lineTo(0, 0);
  }

  var my_interval:Number;
  var my_sound:Sound = new Sound();
  my_sound.onLoad = function(success:Boolean) {
    if (success) {
      trace("sound loaded");
    }
  };
  my_sound.onSoundComplete = function() {
    clearInterval(my_interval);
    trace("Cleared interval");
  }
  my_sound.loadSound("song3.mp3", true);
  my_interval = setInterval(updateProgressBar, 100, my_sound);

  function updateProgressBar(the_sound:Sound):Void {
    var pos:Number = Math.round(the_sound.position/the_sound.duration*100);
    pb.bar_mc._xscale = pos;
    pb.vBar_mc._x = pb.bar_mc._width;
    pb.pos_txt.text = pos+"%";
  }
```

See Also

Sound.position

Sound.getBytesLoaded()

Availability

Flash Player 6.

Usage

```
my_sound.getBytesLoaded() : Number
```

Parameters

None.

Returns

An integer indicating the number of bytes loaded.

Description

Method; returns the number of bytes loaded (streamed) for the specified Sound object. You can compare the value of `getBytesLoaded()` with the value of `getBytesTotal()` to determine what percentage of a sound has loaded.

Example

The following example dynamically creates two text fields that display the bytes that are loaded and the total number of bytes for a sound file that loads into the SWF file. A text field also displays a message when the file finishes loading. Add the following ActionScript to your FLA or AS file:

```
this.createTextField("message_txt", this.getNextHighestDepth(), 10,10,300,22)
this.createTextField("status_txt", this.getNextHighestDepth(), 10, 50, 300,
  40);
status_txt.autoSize = true;
status_txt.multiline = true;
status_txt.border = false;

var my_sound:Sound = new Sound();
my_sound.onLoad = function(success:Boolean) {
  if (success) {
    this.start();
    message_txt.text = "Finished loading";
  }
};
my_sound.onSoundComplete = function() {
  message_txt.text = "Clearing interval";
  clearInterval(my_interval);
};
my_sound.loadSound("song2.mp3", true);
var my_interval:Number;
my_interval = setInterval(checkProgress, 100, my_sound);
function checkProgress(the_sound:Sound):Void {
  var pct:Number = Math.round(the_sound.getBytesLoaded()/
  the_sound.getBytesTotal()*100);
  var pos:Number = Math.round(the_sound.position/the_sound.duration*100);
```

```
    status_txt.text = the_sound.getBytesLoaded()+" of
    "+the_sound.getBytesTotal()+" bytes ("+pct+"%)"+newline;
    status_txt.text += the_sound.position+" of "+the_sound.duration+"
    milliseconds ("+pos+"%)"+newline;
  }
```

See also

`Sound.getBytesTotal()`

Sound.getBytesTotal()

Availability

Flash Player 6.

Usage

```
my_sound.getBytesTotal() : Number
```

Parameters

None.

Returns

An integer indicating the total size, in bytes, of the specified Sound object.

Description

Method; returns the size, in bytes, of the specified Sound object.

Example

See Sound.getBytesLoaded() for a sample usage of this method.

See also

`Sound.getBytesLoaded()`

Sound.getPan()

Availability

Flash Player 5.

Usage

```
my_sound.getPan() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the pan level set in the last `setPan()` call as an integer from -100 (left) to 100 (right). (0 sets the left and right channels equally.) The pan setting controls the left-right balance of the current and future sounds in a SWF file.

This method is cumulative with `setVolume()` or `setTransform()`.

Example

The following example creates a slider bar using the Drawing API. When the user drags the slider bar, the pan level of the loaded sound changes. The current pan level is displayed in a dynamically created text field. Add the following ActionScript to your FLA or AS file:

```
var bar_width:Number = 200;
this.createEmptyMovieClip("bar_mc", this.getNextHighestDepth());
with (bar_mc) {
  lineStyle(4, 0x000000);
  moveTo(0, 0);
  lineTo(bar_width+4, 0);
  lineStyle(0, 0x000000);
  moveTo((bar_width/2)+2, -8);
  lineTo((bar_width/2)+2, 8);
}
bar_mc._x = 100;
bar_mc._y = 100;

this.createEmptyMovieClip("knob_mc", this.getNextHighestDepth());
with (knob_mc) {
  lineStyle(0, 0x000000);
  beginFill(0xCCCCCC);
  moveTo(0, 0);
  lineTo(4, 0);
  lineTo(4, 10);
  lineTo(0, 10);
  lineTo(0, 0);
  endFill();
}
knob_mc._x = bar_mc._x+(bar_width/2);
knob_mc._y = bar_mc._y-(knob_mc._height/2);
knob_mc.left = knob_mc._x-(bar_width/2);
knob_mc.right = knob_mc._x+(bar_width/2);
knob_mc.top = knob_mc._y;
knob_mc.bottom = knob_mc._y;

knob_mc.onPress = function() {
  this.startDrag(false, this.left, this.top, this.right, this.bottom);
};
knob_mc.onRelease = function() {
  this.stopDrag();
  var multiplier:Number = 100/(this.right-this.left)*2;
  var pan:Number = (this._x-this.left-(bar_width/2))*multiplier;
  my_sound.setPan(pan);
  pan_txt.text = my_sound.getPan();
};

var my_sound:Sound = new Sound();
my_sound.loadSound("song2.mp3", true);
this.createTextField("pan_txt", this.getNextHighestDepth(), knob_mc._x,
  knob_mc._y+knob_mc._height, 20, 22);
pan_txt.selectable = false;
pan_txt.autoSize = "center";
pan_txt.text = my_sound.getPan();
```

See also

`Sound.setPan()`

Sound.getTransform()

Availability

Flash Player 5.

Usage

```
my_sound.getTransform() : Object
```

Parameters

None.

Returns

An object with properties that contain the channel percentage values for the specified sound object.

Description

Method; returns the sound transform information for the specified Sound object set with the last `Sound.setTransform()` call.

Example

The following example attaches four movie clips from a symbol in the library (linkage identifier: `knob_id`) that are used as sliders (or "knobs") to control the sound file that loads into the SWF file. These sliders control the transform object, or balance, of the sound file. For more information, see the entry for Sound.setTransform(). Add the following ActionScript to your FLA or AS file:

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song1.mp3", true);
var transform_obj:Object = my_sound.getTransform();

this.createEmptyMovieClip("transform_mc", this.getNextHighestDepth());
transform_mc.createTextField("transform_txt",
  transform_mc.getNextHighestDepth, 0, 8, 120, 22);
transform_mc.transform_txt.html = true;

var knob_ll:MovieClip = transform_mc.attachMovie("knob_id", "ll_mc",
  transform_mc.getNextHighestDepth(), {_x:0, _y:30});
var knob_lr:MovieClip = transform_mc.attachMovie("knob_id", "lr_mc",
  transform_mc.getNextHighestDepth(), {_x:30, _y:30});
var knob_rl:MovieClip = transform_mc.attachMovie("knob_id", "rl_mc",
  transform_mc.getNextHighestDepth(), {_x:60, _y:30});
var knob_rr:MovieClip = transform_mc.attachMovie("knob_id", "rr_mc",
  transform_mc.getNextHighestDepth(), {_x:90, _y:30});

knob_ll.top = knob_ll._y;
knob_ll.bottom = knob_ll._y+100;
knob_ll.left = knob_ll._x;
knob_ll.right = knob_ll._x;
knob_ll._y = knob_ll._y+(100-transform_obj['ll']);
knob_ll.onPress = pressKnob;
knob_ll.onRelease = releaseKnob;
knob_ll.onReleaseOutside = releaseKnob;

knob_lr.top = knob_lr._y;
knob_lr.bottom = knob_lr._y+100;
knob_lr.left = knob_lr._x;
knob_lr.right = knob_lr._x;
knob_lr._y = knob_lr._y+(100-transform_obj['lr']);
knob_lr.onPress = pressKnob;
knob_lr.onRelease = releaseKnob;
knob_lr.onReleaseOutside = releaseKnob;

knob_rl.top = knob_rl._y;
knob_rl.bottom = knob_rl._y+100;
knob_rl.left = knob_rl._x;
knob_rl.right = knob_rl._x;
knob_rl._y = knob_rl._y+(100-transform_obj['rl']);
knob_rl.onPress = pressKnob;
knob_rl.onRelease = releaseKnob;
knob_rl.onReleaseOutside = releaseKnob;

knob_rr.top = knob_rr._y;
knob_rr.bottom = knob_rr._y+100;
knob_rr.left = knob_rr._x;
knob_rr.right = knob_rr._x;
knob_rr._y = knob_rr._y+(100-transform_obj['rr']);
knob_rr.onPress = pressKnob;
knob_rr.onRelease = releaseKnob;

knob_rr.onReleaseOutside = releaseKnob;

updateTransformTxt();

function pressKnob() {
  this.startDrag(false, this.left, this.top, this.right, this.bottom);
}
function releaseKnob() {
  this.stopDrag();
  updateTransformTxt();
}
function updateTransformTxt() {
  var ll_num:Number = 30+100-knob_ll._y;
  var lr_num:Number = 30+100-knob_lr._y;
  var rl_num:Number = 30+100-knob_rl._y;
  var rr_num:Number = 30+100-knob_rr._y;
  my_sound.setTransform({ll:ll_num, lr:lr_num, rl:rl_num, rr:rr_num});
  transform_mc.transform_txt.htmlText = "<textformat
  tabStops='[0,30,60,90]'>";
  transform_mc.transform_txt.htmlText +=
  ll_num+"\t"+lr_num+"\t"+rl_num+"\t"+rr_num;
  transform_mc.transform_txt.htmlText += "</textformat>";
}
```

See Also

Sound.setTransform()

Sound.getVolume()

Availability

Flash Player 5.

Usage

```
my_sound.getVolume() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the sound volume level as an integer from 0 to 100, where 0 is off and 100 is full volume. The default setting is 100.

Example

The following example creates a slider using the Drawing API and a movie clip that is created at runtime. A dynamically created text field displays the current volume level of the sound playing in the SWF file. Add the following ActionScript to your AS or FLA file:

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song3.mp3", true);

this.createEmptyMovieClip("knob_mc", this.getNextHighestDepth());

knob_mc.left = knob_mc._x;
knob_mc.right = knob_mc.left+100;
knob_mc.top = knob_mc._y;
knob_mc.bottom = knob_mc._y;

knob_mc._x = my_sound.getVolume();

with (knob_mc) {
  lineStyle(0, 0x000000);
  beginFill(0xCCCCCC);
  moveTo(0, 0);
  lineTo(4, 0);
  lineTo(4, 18);
  lineTo(0, 18);
  lineTo(0, 0);
  endFill();
}

knob_mc.createTextField("volume_txt", knob_mc.getNextHighestDepth(),
  knob_mc._width+4, 0, 30, 22);
knob_mc.volume_txt.text = my_sound.getVolume();

knob_mc.onPress = function() {
  this.startDrag(false, this.left, this.top, this.right, this.bottom);
```

```
    this.isDragging = true;
  };
  knob_mc.onMouseMove = function() {
    if (this.isDragging) {
      this.volume_txt.text = this._x;
    }
  }
  knob_mc.onRelease = function() {
    this.stopDrag();
    this.isDragging = false;
    my_sound.setVolume(this._x);

  };
```

See also

`Sound.setVolume()`

Sound.id3

Availability

Flash Player 6; behavior updated in Flash Player 7.

Usage

my_sound.id3*:Object*

Description

Read-only property; provides access to the metadata that is part of an MP3 file.

MP3 sound files can contain ID3 tags, which provide metadata about the file. If an MP3 sound that you load using Sound.attachSound() or Sound.loadSound() contains ID3 tags, you can query these properties. Only ID3 tags that use the UTF-8 character set are supported.

Flash Player 6 (6.0.40.0) and later use the `Sound.id3` property to support ID3 1.0 and ID3 1.1 tags. Flash Player 7 adds support for ID3 2.0 tags, specifically 2.3 and 2.4. The following table lists the standard ID3 2.0 tags and the type of content the tags represent; you query them in the format *my_sound*`.id3.COMM`, *my_sound*`.id3.TIME`, and so on. MP3 files can contain tags other than those in this table; `Sound.id3` provides access to those tags as well.

Property	Description
`COMM`	Comment
`TALB`	Album/movie/show title
`TBPM`	Beats per minute
`TCOM`	Composer
`TCON`	Content type
`TCOP`	Copyright message
`TDAT`	Date
`TDLY`	Playlist delay
`TENC`	Encoded by
`TEXT`	Lyricist/text writer
`TFLT`	File type
`TIME`	Time
`TIT1`	Content group description
`TIT2`	Title/song name/content description
`TIT3`	Subtitle/description refinement
`TKEY`	Initial key
`TLAN`	Languages
`TLEN`	Length

Property	Description
TMED	Media type
TOAL	Original album/movie/show title
TOFN	Original filename
TOLY	Original lyricists/text writers
TOPE	Original artists/performers
TORY	Original release year
TOWN	File owner/licensee
TPE1	Lead performers/soloists
TPE2	Band/orchestra/accompaniment
TPE3	Conductor/performer refinement
TPE4	Interpreted, remixed, or otherwise modified by
TPOS	Part of a set
TPUB	Publisher
TRCK	Track number/position in set
TRDA	Recording dates
TRSN	Internet radio station name
TRSO	Internet radio station owner
TSIZ	Size
TSRC	ISRC (international standard recording code)
TSSE	Software/hardware and settings used for encoding
TYER	Year
WXXX	URL link frame

Flash Player 6 supported several ID31.0 tags. If these tags are in not in the MP3 file, but corresponding ID3 2.0 tags are, the ID3 2.0 tags are copied into the ID3 1.0 properties, as shown in the following table. This process provides backward compatibility with scripts that you may have written already that read ID3 1.0 properties.

ID3 2.0 tag	Corresponding ID3 1.0 property
COMM	`Sound.id3.comment`
TALB	`Sound.id3.album`
TCON	`Sound.id3.genre`
TIT2	`Sound.id3.songname`
TPE1	`Sound.id3.artist`

ID3 2.0 tag	Corresponding ID3 1.0 property
`TRCK`	`Sound.id3.track`
`TYER`	`Sound.id3.year`

Example

The following example traces the ID3 properties of song.mp3 to the Output panel:

```
var my_sound:Sound = new Sound();
my_sound.onID3 = function(){
  for( var prop in my_sound.id3 ){
    trace( prop + " : "+ my_sound.id3[prop] );
  }
}
my_sound.loadSound("song.mp3", false);
```

See also

`Sound.attachSound()`, `Sound.loadSound()`

Sound.loadSound()

Availability

Flash Player 6.

Usage

my_sound.loadSound(*url:String*, *isStreaming:Boolean*) *: Void*

Parameters

url The location on a server of an MP3 sound file.

isStreaming A Boolean value that indicates whether the sound is a streaming sound (`true`) or an event sound (`false`).

Returns

Nothing.

Description

Method; loads an MP3 file into a Sound object. You can use the *isStreaming* parameter to indicate whether the sound is an event or a streaming sound.

Event sounds are completely loaded before they play. They are managed by the ActionScript Sound class and respond to all methods and properties of this class.

Streaming sounds play while they are downloading. Playback begins when sufficient data has been received to start the decompressor. For more information, see “Working with Sound” in *Using Flash*.

All MP3s (event or streaming) loaded with this method are saved in the browser’s file cache on the user’s system.

Example

The following example loads an event sound, which cannot play until it is fully loaded:

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song1.mp3", false);
```

The following example loads a streaming sound:

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song1.mp3", true);
```

See also

`Sound.onLoad`

Sound.onID3

Availability

Flash Player 7.

Usage

```
my_sound.onID3 = function(){
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked each time new ID3 data is available for an MP3 file that you load using Sound.attachSound() or Sound.loadSound(). This handler provides access to ID3 data without polling. If both ID3 1.0 and ID3 2.0 tags are present in a file, this handler is called twice.

Example

The following example displays the ID3 properties of song1.mp3 to an instance of the DataGrid component. Add a DataGrid with the instance name `id3_dg` to your document, and add the following ActionScript to your FLA or AS file:

```
import mx.controls.gridclasses.DataGridColumn;
var id3_dg:mx.controls.DataGrid;
id3_dg.move(0, 0);
id3_dg.setSize(Stage.width, Stage.height);
var property_dgc:DataGridColumn = id3_dg.addColumn(new
  DataGridColumn("property"));
property_dgc.width = 100;
property_dgc.headerText = "ID3 Property";
var value_dgc:DataGridColumn = id3_dg.addColumn(new DataGridColumn("value"));
value_dgc.width = id3_dg._width-property_dgc.width;
value_dgc.headerText = "ID3 Value";

var my_sound:Sound = new Sound();
my_sound.onID3 = function() {
  trace("onID3 called at "+getTimer()+" ms.");
  for (var prop in this.id3) {
    id3_dg.addItem({property:prop, value:this.id3[prop]});
  }
};
my_sound.loadSound("song1.mp3", true);
```

See also

`Sound.attachSound(), Sound.id3, Sound.loadSound()`

Sound.onLoad

Availability

Flash Player 6.

Usage

```
my_sound.onLoad = function(success:Boolean){
  // your statements here
}
```

Parameters

`success` A Boolean value of `true` if `my_sound` has been loaded successfully, false otherwise.

Returns

Nothing.

Description

Event handler; invoked automatically when a sound loads. You must create a function that executes when the this handler is invoked. You can use either an anonymous function or a named function (for an example of each, see `Sound.onSoundComplete`). You should define this handler before you call `mySound.loadSound()`.

Example

The following example creates a new Sound object, and loads a sound. Loading the sound is handled by the `onLoad` handler, which allows you to start the song after it is successfully loaded. Create a new FLA file, and add the following ActionScript to your FLA or AS file. For this example to work, you must have an MP3 called song1.mp3 in the same directory as your FLA or AS file.

```
this.createTextField("status_txt", this.getNextHighestDepth(), 0,0,100,22);

// create a new Sound object
var my_sound:Sound = new Sound();
// if the sound loads, play it; if not, trace failure loading
my_sound.onLoad = function(success:Boolean) {
  if (success) {
    my_sound.start();
    status_txt.text = "Sound loaded";
  } else {
    status_txt.text = "Sound failed";
  }
};
// load the sound
my_sound.loadSound("song1.mp3", true);
```

See also

`Sound.loadSound()`

Sound.onSoundComplete

Availability

Flash Player 6.

Usage

```
my_sound.onSoundComplete = function(){
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked automatically when a sound finishes playing. You can use this handler to trigger events in a SWF file when a sound finishes playing.

You must create a function that executes when this handler is invoked. You can use either an anonymous function or a named function.

Example

Usage 1: The following example uses an anonymous function:

```
var my_sound:Sound = new Sound();
my_sound.attachSound("mySoundID");
my_sound.onSoundComplete = function() {
  trace("mySoundID completed");
};
my_sound.start();
```

Usage 2: The following example uses a named function:

```
function callback1() {
  trace("mySoundID completed");
}
var my_sound:Sound = new Sound();
my_sound.attachSound("mySoundID");
my_sound.onSoundComplete = callback1;
my_sound.start();
```

See also

`Sound.onLoad`

Sound.position

Availability

Flash Player 6.

Usage

```
my_sound.position:Number
```

Description

Read-only property; the number of milliseconds a sound has been playing. If the sound is looped, the position is reset to 0 at the beginning of each loop.

Example

See Sound.duration for a sample usage of this property.

See Also

`Sound.duration`

Sound.setPan()

Availability

Flash Player 5.

Usage

```
my_sound.setPan(pan:Number) : Number
```

Parameters

`pan` An integer specifying the left-right balance for a sound. The range of valid values is -100 to 100, where -100 uses only the left channel, 100 uses only the right channel, and 0 balances the sound evenly between the two channels.

Returns

An integer.

Description

Method; determines how the sound is played in the left and right channels (speakers). For mono sounds, *pan* determines which speaker (left or right) the sound plays through.

Example

See Sound.getPan() for a sample usage of this method.

See also

`Sound.attachSound()`, `Sound.getPan()`, `Sound.setTransform()`, `Sound.setVolume()`, `Sound.start()`

Sound.setTransform()

Availability

Flash Player 5.

Usage

```
my_sound.setTransform(soundTransformObject:Object) : Void
```

Parameters

`soundTransformObject` An object created with the constructor for the generic Object class.

Returns

Nothing.

Description

Method; sets the sound transform (or balance) information, for a Sound object.

The `soundTransformObject` parameter is an object that you create using the constructor method of the generic Object class with parameters specifying how the sound is distributed to the left and right channels (speakers).

Sounds use a considerable amount of disk space and memory. Because stereo sounds use twice as much data as mono sounds, it is generally best to use 22-KHz 6-bit mono sounds. You can use `setTransform()` to play mono sounds as stereo, play stereo sounds as mono, and to add interesting effects to sounds.

The properties for the `soundTransformObject` are as follows:

`ll` A percentage value specifying how much of the left input to play in the left speaker (0-100).

`lr` A percentage value specifying how much of the right input to play in the left speaker (0-100).

`rr` A percentage value specifying how much of the right input to play in the right speaker (0-100).

`rl` A percentage value specifying how much of the left input to play in the right speaker (0-100).

The net result of the parameters is represented by the following formula:

```
leftOutput = left_input * ll + right_input * lr
rightOutput = right_input * rr + left_input * rl
```

The values for `left_input` or `right_input` are determined by the type (stereo or mono) of sound in your SWF file.

Stereo sounds divide the sound input evenly between the left and right speakers and have the following transform settings by default:

```
ll = 100
lr = 0
rr = 100
rl = 0
```

Mono sounds play all sound input in the left speaker and have the following transform settings by default:

```
ll = 100
lr = 100
rr = 0
rl = 0
```

Example

The following example illustrates a setting that can be achieved by using `setTransform()`, but cannot be achieved by using `setVolume()` or `setPan()`, even if they are combined.

The following code creates a new `soundTransformObject` object and sets its properties so that sound from both channels will play only in the left channel.

```
var mySoundTransformObject:Object = new Object();
mySoundTransformObject.ll = 100;
mySoundTransformObject.lr = 100;
mySoundTransformObject.rr = 0;
mySoundTransformObject.rl = 0;
```

To apply the `soundTransformObject` object to a Sound object, you then need to pass the object to the Sound object using `setTransform()` as follows:

```
my_sound.setTransform(mySoundTransformObject);
```

The following example plays a stereo sound as mono; the `soundTransformObjectMono` object has the following parameters:

```
var mySoundTransformObjectMono:Object = new Object();
mySoundTransformObjectMono.ll = 50;
mySoundTransformObjectMono.lr = 50;
mySoundTransformObjectMono.rr = 50;
mySoundTransformObjectMono.rl = 50;
my_sound.setTransform(mySoundTransformObjectMono);
```

This example plays the left channel at half capacity and adds the rest of the left channel to the right channel; the `soundTransformObjectHalf` object has the following parameters:

```
var mySoundTransformObjectHalf:Object = new Object();
mySoundTransformObjectHalf.ll = 50;
mySoundTransformObjectHalf.lr = 0;
mySoundTransformObjectHalf.rr = 100;
mySoundTransformObjectHalf.rl = 50;
my_sound.setTransform(mySoundTransformObjectHalf);

var mySoundTransformObjectHalf:Object = {ll:50, lr:0, rr:100, rl:50};
```

Also see the example for Sound.getTransform().

See also

Object class, Sound.getTransform()

Sound.setVolume()

Availability

Flash Player 5.

Usage

`my_sound.setVolume(volume:Number) : Void`

Parameters

`volume` A number from 0 to 100 representing a volume level. 100 is full volume and 0 is no volume. The default setting is 100.

Returns

Nothing.

Description

Method; sets the volume for the Sound object.

Example

See Sound.getVolume() for a sample usage of this method.

See also

`Sound.setPan()`, `Sound.setTransform()`

Sound.start()

Availability

Flash Player 5.

Usage

my_sound.start([*secondOffset*:*Number*, *loop*:*Number*]) : *Void*

Parameters

secondOffset An optional parameter that lets you start playing the sound at a specific point. For example, if you have a 30-second sound and want the sound to start playing in the middle, specify 15 for the *secondOffset* parameter. The sound is not delayed 15 seconds, but rather starts playing at the 15-second mark.

loop An optional parameter that lets you specify the number of times the sound should play consecutively.

Returns

Nothing.

Description

Method; starts playing the last attached sound from the beginning if no parameter is specified, or starting at the point in the sound specified by the *secondOffset* parameter.

Example

The following example creates a new Sound object, and loads a sound. Loading the sound is handled by the `onLoad` handler, which allows you to start the song after it's successfully loaded. Then the sound starts playing using the `start()` method. Create a new FLA file, and add the following ActionScript to your FLA or AS file. For this example to work, you must have an MP3 called song1.mp3 in the same directory as your FLA or AS file.

```
this.createTextField("status_txt", this.getNextHighestDepth(), 0,0,100,22);

// create a new Sound object
var my_sound:Sound = new Sound();
// if the sound loads, play it; if not, trace failure loading
my_sound.onLoad = function(success:Boolean) {
  if (success) {
    my_sound.start();
    status_txt.text = "Sound loaded";
  } else {
    status_txt.text = "Sound failed";
  }
};
// load the sound
my_sound.loadSound("song1.mp3", true);
```

See also

`Sound.stop()`

Sound.stop()

Availability

Flash Player 5.

Usage

`my_sound.stop(["idName":String]) : Void`

Parameters

`idName` An optional parameter specifying a specific sound to stop playing. The `idName` parameter must be enclosed in quotation marks (" ").

Returns

Nothing.

Description

Method; stops all sounds currently playing if no parameter is specified, or just the sound specified in the `idName` parameter.

Example

The following example uses two buttons, `stop_btn` and `play_btn`, to control the playback of a sound that loads into a SWF file. Add two buttons to your document and add the following ActionScript to your FLA or AS file:

```
var my_sound:Sound = new Sound();
my_sound.loadSound("song1.mp3", true);

stop_btn.onRelease = function() {
  trace("sound stopped");
  my_sound.stop();
};
play_btn.onRelease = function() {
  trace("sound started");
  my_sound.start();
};
```

See also

`Sound.start()`

_soundbuftime

Availability

Flash Player 4.

Usage

`_soundbuftime:Number = integer`

Parameters

`integer` The number of seconds before the SWF file starts to stream.

Description

Property (global); establishes the number of seconds of streaming sound to buffer. The default value is 5 seconds.

Example

The following example streams an MP3 file and buffers the sound before it plays for the user. Two text fields are created at runtime to hold a timer and debugging information. The `_soundbuftime` property is set to buffer the MP3 for 10 seconds. A new Sound object instance is created for the MP3.

```
// create text fields to hold debug information.
this.createTextField("counter_txt", this.getNextHighestDepth(), 0, 0, 100,
  22);
this.createTextField("debug_txt", this.getNextHighestDepth(), 0, 20, 100, 22);
// set the sound buffer to 10 seconds.
_soundbuftime = 10;
// create the new sound object instance
var bg_sound:Sound = new Sound();
// load the MP3 sound file and set streaming to true.
bg_sound.loadSound("yourSound.mp3", true);
// function is triggered when the song finishes loading
bg_sound.onLoad = function() {
  debug_txt.text = "sound loaded";
};
debug_txt.text = "sound init";
function updateCounter() {
  counter_txt.text++;
}
counter_txt.text = 0;
setInterval(updateCounter, 1000);
```

Stage class

Availability

Flash Player 6.

Description

The Stage class is a top-level class whose methods, properties, and handlers you can access without using a constructor. Use the methods and properties of this class to access and manipulate information about the boundaries of a SWF file.

Method summary for the Stage class

Method	Description
`Stage.addListener()`	Adds a listener object that detects when a SWF file is resized.
`Stage.removeListener()`	Removes a listener object from the Stage object.

Property summary for the Stage class

Property	Description
`Stage.align`	Alignment of the SWF file in the player or browser.
`Stage.height`	Height of the Stage, in pixels.
`Stage.scaleMode`	The current scaling of the SWF file.
`Stage.showMenu`	Shows or hides the default items in the Flash Player context menu.
`Stage.width`	Width of the Stage, in pixels.

Event handler summary for the Stage class

Event handler	Description
`Stage.onResize`	Invoked when `Stage.scaleMode` is set to `"noScale"` and the SWF file is resized.

Stage.addListener()

Availability

Flash Player 6.

Usage

```
Stage.addListener(myListener:Object) : Void
```

Parameters

myListener An object that listens for a callback notification from the Stage.onResize event.

Returns

Nothing.

Description

Method; detects when a SWF file is resized (but only if `Stage.scaleMode = "noScale"`). The `addListener()` method doesn't work with the default movie clip scaling setting (`showAll`) or other scaling settings (`exactFit` and `noBorder`).

To use `addListener()`, you must first create a *listener object*. Stage listener objects receive notification from `Stage.onResize`.

Example

This example creates a new listener object called `stageListener`. It then uses `myListener` to call `onResize` and define a function that will be called when `onResize` is triggered. Finally, the code adds the `myListener` object to the callback list of the Stage object. Listener objects allow multiple objects to listen for resize notifications.

```
this.createTextField("stageSize_txt", this.getNextHighestDepth(), 10, 10, 100,
  22);
var stageListener:Object = new Object();
stageListener.onResize = function() {
  stageSize_txt.text = "w:"+Stage.width+", h:"+Stage.height;
};
Stage.scaleMode = "noScale";
Stage.addListener(stageListener);
```

See also

Stage.onResize, Stage.removeListener()

Stage.align

Availability

Flash Player 6.

Usage

```
Stage.align:String
```

Description

Property; indicates the current alignment of the SWF file in the player or browser.

The following table lists the values for the `align` property. Any value not listed here centers the SWF file in Flash player or browser area, which is the default setting.

Value	Vertical	Horizontal
`"T"`	top	center
`"B"`	bottom	center
`"L"`	center	left
`"R"`	center	right
`"TL"`	top	left
`"TR"`	top	right
`"BL"`	bottom	left
`"BR"`	bottom	right

Example

The following example demonstrates different alignments of the SWF file. Add a ComboBox instance to your document with the instance name `stageAlign_cb`. Add the following ActionScript to your FLA or AS file:

```
var stageAlign_cb:mx.controls.ComboBox;
stageAlign_cb.dataProvider = ['T', 'B', 'L', 'R', 'TL', 'TR', 'BL', 'BR'];
var cbListener:Object = new Object();
cbListener.change = function(evt:Object) {
  var align:String = evt.target.selectedItem;
  Stage.align = align;
};
stageAlign_cb.addEventListener("change", cbListener);
Stage.scaleMode = "noScale";
```

Select different alignment settings from the ComboBox.

Stage.height

Availability

Flash Player 6.

Usage

```
Stage.height:Number
```

Description

Property (read-only); indicates the current height, in pixels, of the Stage. When the value of `Stage.scaleMode` is `noScale`, the `height` property represents the height of Flash Player. When the value of `Stage.scaleMode` is not `noScale`, `height` represents the height of the SWF file.

Example

This example creates a new listener object called `stageListener`. It then uses `myListener` to call `onResize` and define a function that will be called when `onResize` is triggered. Finally, the code adds the `myListener` object to the callback list of the Stage object. Listener objects allow multiple objects to listen for resize notifications.

```
this.createTextField("stageSize_txt", this.getNextHighestDepth(), 10, 10, 100,
  22);
var stageListener:Object = new Object();
stageListener.onResize = function() {
  stageSize_txt.text = "w:"+Stage.width+", h:"+Stage.height;
};
Stage.scaleMode = "noScale";
Stage.addListener(stageListener);
```

See also

`Stage.align`, `Stage.scaleMode`, `Stage.width`

Stage.onResize

Availability

Flash Player 6.

Usage

```
myListener.onResize = function(){
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Listener; invoked when `Stage.scaleMode` is set to `noScale` and the SWF file is resized. You can use this event handler to write a function that lays out the objects on the Stage when a SWF file is resized.

Example

The following example displays a message in the Output panel when the Stage is resized:

```
Stage.scaleMode = "noScale"
var myListener:Object = new Object();
myListener.onResize = function () {
  trace("Stage size is now " + Stage.width + " by " + Stage.height);
}
Stage.addListener(myListener);
// later, call Stage.removeListener(myListener)
```

See also

`Stage.addListener()`, `Stage.removeListener()`

Stage.removeListener()

Availability

Flash Player 6.

Usage

`Stage.removeListener(myListener:Object) : Boolean`

Parameters

`myListener` An object added to an object's callback list with `addListener()`.

Returns

A Boolean value.

Description

Method; removes a listener object created with `addListener()`.

Example

The following example displays the Stage dimensions in a dynamically created text field. When you resize the Stage, the values in the text field update. Create a button with an instance name `remove_btn`. Add the following ActionScript to Frame 1 of the Timeline.

```
this.createTextField("stageSize_txt", this.getNextHighestDepth(), 10, 10, 100, 
  22);
stageSize_txt.autoSize = true;
stageSize_txt.border = true;
var stageListener:Object = new Object();
stageListener.onResize = function() {
  stageSize_txt.text = "w:"+Stage.width+", h:"+Stage.height;
};
Stage.addListener(stageListener);

remove_btn.onRelease = function() {
  stageSize_txt.text = "Removing Stage listener...";
  Stage.removeListener(stageListener);
}
```

Select Control > Test Movie to test this example. The values you see in the text field are updated when you resize the testing environment. When you click `remove_btn`, the listener is removed and the values are no longer updated in the text field.

See also

`Stage.addListener()`

Stage.scaleMode

Availability

Flash Player 6.

Usage

`Stage.scaleMode:String`

Description

Property; indicates the current scaling of the SWF file within Flash Player. The `scaleMode` property forces the SWF file into a specific scaling mode. By default, the SWF file uses the HTML parameters set in the Publish Settings dialog box.

The `scaleMode` property can use the values `"exactFit"`, `"showAll"`, `"noBorder"`, and `"noScale"`. Any other value sets the `scaleMode` property to the default `"showAll"`.

Example

The following example demonstrates various scale settings for the SWF file. Add a ComboBox instance to your document with the instance name `scaleMode_cb`. Add the following ActionScript to your FLA or AS file:

```
var scaleMode_cb:mx.controls.ComboBox;
scaleMode_cb.dataProvider = ["showAll", "exactFit", "noBorder", "noScale"];
var cbListener:Object = new Object();
cbListener.change = function(evt:Object) {
  var scaleMode_str:String = evt.target.selectedItem;
  Stage.scaleMode = scaleMode_str;
};
scaleMode_cb.addEventListener("change", cbListener);
```

To view another example, see the stagesize.fla file in the HelpExamples Folder. The following list provides typical paths to the HelpExamples Folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/
-

Stage.showMenu

Availability

Flash Player 6.

Usage

```
Stage.showMenu:Boolean
```

Description

Property; specifies whether to show or hide the default items in the Flash Player context menu. If `showMenu` is set to `true` (the default), all context menu items appear. If `showMenu` is set to `false`, only Settings and About Macromedia Flash Player items appear.

Example

The following example creates a clickable text link that lets the user enable and disable the Flash Player context menu.

```
this.createTextField("showMenu_txt", this.getNextHighestDepth(), 10, 10, 100,
  22);
showMenu_txt.html = true;
showMenu_txt.autoSize = true;
showMenu_txt.htmlText = "<a href=\"asfunction:toggleMenu\"><u>Stage.showMenu =
  "+Stage.showMenu+"</u></a>";
function toggleMenu() {
  Stage.showMenu = !Stage.showMenu;
  showMenu_txt.htmlText = "<a href=\"asfunction:toggleMenu\"><u>Stage.showMenu
  = "+Stage.showMenu+"</u></a>";
}
```

See also

ContextMenu class, ContextMenuItem class

Stage.width

Availability

Flash Player 6.

Usage

`Stage.width:Number`

Description

Property (read-only); indicates the current width, in pixels, of the Stage. When the value of `Stage.scaleMode` is `"noScale"`, the `width` property represents the width of Flash Player. This means that `Stage.width` will vary as you resize the player window. When the value of `Stage.scaleMode` is not `"noScale"`, `width` represents the width of the SWF file as set at author-time in the Document Properties dialog box. This means that the value of `width` will stay constant as you resize the player window.

Example

This example creates a new listener object called `stageListener`. It then uses `myListener` to call `onResize` and define a function that will be called when `onResize` is triggered. Finally, the code adds the `myListener` object to the callback list of the Stage object. Listener objects allow multiple objects to listen for resize notifications.

```
this.createTextField("stageSize_txt", this.getNextHighestDepth(), 10, 10, 100,
  22);
var stageListener:Object = new Object();
stageListener.onResize = function() {
  stageSize_txt.text = "w:"+Stage.width+", h:"+Stage.height;
};
Stage.scaleMode = "noScale";
Stage.addListener(stageListener);
```

See also

`Stage.align`, `Stage.height`, `Stage.scaleMode`

startDrag()

Availability

Flash Player 4.

Usage

```
startDrag(target:Object,[lock:Boolean, left:Number, top:Number, right:Number,
  bottom:Number]) : Void
```

Parameters

target The target path of the movie clip to drag.

lock A Boolean value specifying whether the draggable movie clip is locked to the center of the mouse position (`true`) or locked to the point where the user first clicked the movie clip (`false`). This parameter is optional.

left, *top*, *right*, *bottom* Values relative to the coordinates of the movie clip's parent that specify a constraint rectangle for the movie clip. These parameters are optional.

Returns

Nothing.

Description

Function; makes the *target* movie clip draggable while the movie plays. Only one movie clip can be dragged at a time. After a `startDrag()` operation is executed, the movie clip remains draggable until it is explicitly stopped by `stopDrag()` or until a `startDrag()` action for another movie clip is called.

Example

The following example creates a movie clip, `pic_mc`, at runtime that users can drag to any location by attaching the `startDrag()` and `stopDrag()` actions to the movie clip. An image is loaded into `pic_mc` using the MovieClipLoader class.

```
var pic_mcl:MovieClipLoader = new MovieClipLoader();
pic_mcl.loadClip("http://www.macromedia.com/devnet/mx/blueprint/articles/
  qa_petmarket/spotlight_thale.jpg", this.createEmptyMovieClip("pic_mc",
  this.getNextHighestDepth()));
var listenerObject:Object = new Object();
listenerObject.onLoadInit = function(target_mc) {
  target_mc.onPress = function() {
    startDrag(this);
  };
  target_mc.onRelease = function() {
    stopDrag();
  };
};
pic_mcl.addListener(listenerObject);
```

See also

`MovieClip._droptarget`, `MovieClip.startDrag()`, `stopDrag()`

static

Availability

Flash Player 6.

Usage

```
class someClassName{
  static var name;
  static function name() {
    // your statements here
  }
}
```

Note: To use this keyword, you must specify ActionScript 2.0 and Flash Player 6 or later in the Flash tab of your FLA file's Publish Settings dialog box. This keyword is supported only when used in external script files, not in scripts written in the Actions panel.

Parameters

`name` The name of the variable or function that you want to specify as static.

Description

Keyword; specifies that a variable or function is created only once per class rather than being created in every object based on that class. For more information, see "Instance and class members" in *Using ActionScript in Flash.* You can access a static class member without creating an instance of the class by using the syntax `someClassName.name`. If you do create an instance of the class, you can also access a static member using the instance.

You can use this keyword in class definitions only, not in interface definitions.

Example

The following example demonstrates how you can use the `static` keyword to create a counter that tracks how many instances of the class have been created. Because the `numInstances` variable is static, it will be created only once for the entire class, not for every single instance. Create a new AS file called Users.as and enter the following code:

```
class Users {
  private static var numInstances:Number = 0;
  function Users() {
    numInstances++;
  }
  static function get instances():Number {
    return numInstances;
  }
}
```

Create a FLA or AS document in the same directory, and enter the following ActionScript in Frame 1 of the Timeline:

```
trace(Users.instances);
var user1:Users = new Users();
trace(Users.instances);
var user2:Users = new Users();
trace(Users.instances);
```

See also

`private`, `public`

stop()

Availability

Flash 2.

Usage

```
stop()
```

Parameters

None.

Returns

Nothing.

Description

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

See also

`gotoAndStop()`, `MovieClip.gotoAndStop()`

stopAllSounds()

Availability

Flash Player 3.

Usage

```
stopAllSounds() : Void
```

Parameters

None.

Returns

Nothing.

Description

Function; stops all sounds currently playing in a SWF file without stopping the playhead. Sounds set to stream will resume playing as the playhead moves over the frames in which they are located.

Example

The following code creates a text field, in which the song's ID3 information appears. A new Sound object instance is created, and your MP3 is loaded into the SWF file. ID3 information is extracted from the sound file. When the user clicks `stop_mc`, the sound is paused. When the user clicks `play_mc`, the song resumes from its paused position.

```
this.createTextField("songinfo_txt", this.getNextHighestDepth, 0, 0,
  Stage.width, 22);
var bg_sound:Sound = new Sound();
bg_sound.loadSound("yourSong.mp3", true);
bg_sound.onID3 = function() {
  songinfo_txt.text = "("+this.id3.artist+") "+this.id3.album+" -
  "+this.id3.track+" - "+this.id3.songname;
  for (prop in this.id3) {
    trace(prop+" = "+this.id3[prop]);
  }
  trace("ID3 loaded.");
};
this.play_mc.onRelease = function() {
  /* get the current offset. if you stop all sounds and click the play button,
  the MP3 continues from where it was stopped, instead of restarting from the
  beginning. */
  var numSecondsOffset:Number = (bg_sound.position/1000);
  bg_sound.start(numSecondsOffset);
};
this.stop_mc.onRelease = function() {
  stopAllSounds();
};
```

See also

Sound class

stopDrag()

Availability

Flash Player 4.

Usage

```
stopDrag() : Void
```

Parameters

None.

Returns

Nothing.

Description

Function; stops the current drag operation.

Example

The following code, placed in the main Timeline, stops the drag action on the movie clip instance `my_mc` when the user releases the mouse button:

```
my_mc.onPress = function () {
  startDrag(this);
}

my_mc.onRelease = function() {
  stopDrag();
}
```

See also

`MovieClip._droptarget`, `MovieClip.stopDrag()`, `startDrag()`

String()

Availability

Flash Player 4; behavior changed in Flash Player 7.

Usage

`String(expression)`

Parameters

expression	An expression to convert to a string.

Returns

A string.

Description

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

- If *expression* is a number, the return string is a text representation of the number.
- If *expression* is a string, the return string is *expression*.
- If *expression* is an object, the return value is a string representation of the object generated by calling the string property for the object or by calling `Object.toString()` if no such property exists.

If *expression* is `undefined`, the return values are as follows:

- In files published for Flash Player 6 or earlier, the result is an empty string ("").
- In files published for Flash Player 7 or later, the result is `undefined`.
- If *expression* is a Boolean value, the return string is `"true"` or `"false"`.
- If *expression* is a movie clip, the return value is the target path of the movie clip in slash (/) notation.

Note: Slash notation is not supported by ActionScript 2.0.

Example

In the following example, you use ActionScript to convert specified expressions to a string:

```
var string1:String = String("3");
var string2:String = String("9");
trace(string1+string2);  // output: 39
```

Because both parameters are strings, the values are concatenated rather than added.

See also

`Number.toString()`, `Object.toString()`, `String class`, `" " (string delimiter)`

String class

Availability

Flash Player 5 (became a native object in Flash Player 6, which improved performance significantly).

Description

The String class is a wrapper for the string primitive data type, and provides methods and properties that let you manipulate primitive string value types. You can convert the value of any object into a string using the `String()` function.

All the methods of the String class, except for `concat()`, `fromCharCode()`, `slice()`, and `substr()`, are generic, which means the methods call `toString()` before performing their operations, and you can use these methods with other non-String objects.

Because all string indexes are zero-based, the index of the last character for any string `x` is `x.length - 1`.

You can call any of the methods of the String class using the constructor method `new String` or using a string literal value. If you specify a string literal, the ActionScript interpreter automatically converts it to a temporary String object, calls the method, and then discards the temporary String object. You can also use the `String.length` property with a string literal.

Do not confuse a string literal with a String object. In the following example, the first line of code creates the string literal `first_string`, and the second line of code creates the String object `second_string`:

```
var first_string:String = "foo"
var second_string:String = new String("foo")
```

Use string literals unless you specifically need to use a String object.

Method summary for the String class

Method	Description
`String.charAt()`	Returns the character at a specific location in a string.
`String.charCodeAt()`	Returns the value of the character at the specified index as a 16-bit integer between 0 and 65535.
`String.concat()`	Combines the text of two strings and returns a new string.
`String.fromCharCode()`	Returns a string comprising the characters specified in the parameters.
`String.indexOf()`	Returns the position of the first occurrence of a specified substring.
`String.lastIndexOf()`	Returns the position of the last occurrence of a specified substring.
`String.slice()`	Extracts a section of a string and returns a new string.
`String.split()`	Splits a String object into an array of strings by separating the string into substrings.
`String.substr()`	Returns a specified number of characters in a string, beginning at a specified location.

Method	Description
`String.substring()`	Returns the characters between two indexes in a string.
`String.toLowerCase()`	Converts the string to lowercase and returns the result; does not change the contents of the original object.
`String.toUpperCase()`	Converts the string to uppercase and returns the result; does not change the contents of the original object.

Property summary for the String class

Property	Description
`String.length`	A nonzero-based integer specifying the number of characters in the specified String object.

Constructor for the String class

Availability

Flash Player 5.

Usage

```
new String(value:String) : String
```

Parameters

`value` The initial value of the new String object.

Returns

A reference to a String object.

Description

Constructor; creates a new String object.

Note: Because string literals use less overhead than String objects and are generally easier to use, you should use string literals instead of the constructor for the String class unless you have a good reason to use a String object rather than a string literal.

See also

`String()`, `" " (string delimiter)`

String.charAt()

Availability

Flash Player 5.

Usage

my_str.charAt(*index:Number*) : *String*

Parameters

index A number; an integer specifying the position of a character in the string. The first character is indicated by 0, and the last character is indicated by *my_str*.length-1.

Returns

A character.

Description

Method; returns the character in the position specified by the parameter *index*. If *index* is not a number from 0 to string.length - 1, an empty string is returned.

This method is similar to String.charCodeAt() except that the returned value is a character, not a 16-bit integer character code.

Example

In the following example, this method is called on the first letter of the string "Chris":

```
var my_str:String = "Chris";
var firstChar_str:String = my_str.charAt(0);
trace(firstChar_str); // output: C
```

See also

String.charCodeAt()

String.charCodeAt()

Availability

Flash Player 5.

Usage

my_str.charCodeAt(*index:Number*) *: Number*

Parameters

index A number; an integer that specifies the position of a character in the string. The first character is indicated by 0, and the last character is indicated by *my_str*.length - 1.

Returns

A number; an integer.

Description

Method; returns a 16-bit integer from 0 to 65535 that represents the character specified by *index*. If *index* is not a number from 0 to string.length - 1, NaN is returned.

This method is similar to String.charAt() except that the returned value is a 16-bit integer character code, not a character.

Example

In the following example, this method is called on the first letter of the string "Chris":

```
var my_str:String = "Chris";
var firstChar_num:Number = my_str.charCodeAt(0);
trace(firstChar_num); // output: 67
```

See also

String.charAt()

String.concat()

Availability

Flash Player 5.

Usage

`my_str.concat(value1,...valueN) : String`

Parameters

value1,...valueN	Zero or more values to be concatenated.

Returns

A string.

Description

Method; combines the value of the String object with the parameters and returns the newly formed string; the original value, *my_str*, is unchanged.

Example

The following example creates two strings and combines them using `String.concat()`:

```
var stringA:String = "Hello";
var stringB:String = "World";
var combinedAB:String = stringA.concat(" ", stringB);
trace(combinedAB); // output: Hello World
```

String.fromCharCode()

Availability

Flash Player 5.

Usage

`String.fromCharCode(`*`c1:Number,c2,...cN`*`) : `*`String`*

Parameters

`c1,c2,...cN` A number; decimal integers that represent ASCII values.

Returns

A string.

Description

Method; returns a string comprising the characters represented by the ASCII values in the parameters.

Example

The following example uses `fromCharCode()` to insert an `@` character in the e-mail address:

```
var address_str:String = "dog"+String.fromCharCode(64)+"house.net";
trace(address_str); // output: dog@house.net
```

String.indexOf()

Availability

Flash Player 5.

Usage

my_str.indexOf(*substring:String*, [*startIndex:Number*])

Parameters

substring A string; the substring to be searched for within *my_str*.

startIndex A number; an optional integer specifying the starting point in *my_str* to search for the substring.

Returns

A number; the position of the first occurrence of the specified substring or -1.

Description

Method; searches the string and returns the position of the first occurrence of *substring* found at or after *startIndex* within the calling string. This index is zero-based, meaning that the first character in a string is considered to be at index 0—not index 1. If *substring* is not found, the method returns -1.

Example

The following examples use `indexOf()` to return the index of characters and substrings:

```
var searchString:String = "Lorem ipsum dolor sit amet.";
var index:Number;

index = searchString.indexOf("L");
trace(index); // output: 0

index = searchString.indexOf("l");
trace(index); // output: 14

index = searchString.indexOf("i");
trace(index); // output: 6

index = searchString.indexOf("ipsum");
trace(index); // output: 6

index = searchString.indexOf("i", 7);
trace(index); // output: 19

index = searchString.indexOf("z");
trace(index); // output: -1
```

See also

`String.lastIndexOf()`

String.lastIndexOf()

Availability

Flash Player 5.

Usage

`my_str.lastIndexOf(substring:String, [startIndex:Number]) : Number`

Parameters

substring String; the string for which to search.

startIndex Number; an optional integer specifying the starting point from which to search for *substring*.

Returns

A number; the position of the last occurrence of the specified substring or -1.

Description

Method; searches the string from right to left and returns the index of the last occurrence of *substring* found before *startIndex* within the calling string. This index is zero-based, meaning that the first character in a string is considered to be at index 0—not index 1. If *substring* is not found, the method returns -1.

Example

The following example shows how to use `lastIndexOf()` to return the index of a certain character:

```
var searchString:String = "Lorem ipsum dolor sit amet.";
var index:Number;

index = searchString.lastIndexOf("L");
trace(index); // output: 0

index = searchString.lastIndexOf("l");
trace(index); // output: 14

index = searchString.lastIndexOf("i");
trace(index); // output: 19

index = searchString.lastIndexOf("ipsum");
trace(index); // output: 6

index = searchString.lastIndexOf("i", 18);
trace(index); // output: 6

index = searchString.lastIndexOf("z");
trace(index); // output: -1
```

See also

`String.indexOf()`

String.length

Availability

Flash Player 5.

Usage

```
my_str.length:Number
```

Description

Property; an integer specifying the number of characters in the specified String object.

Because all string indexes are zero-based, the index of the last character for any string `x` is `x.length - 1`.

Example

The following example creates a new String object and uses `String.length` to count the number of characters:

```
var my_str:String = "Hello world!";
trace(my_str.length); // output: 12
```

The following example loops from 0 to `my_str.length`. The code checks the characters within a string, and if the string contains the `@` character, `true` displays in the Output panel. If it does not contain the `@` character, then `false` displays in the Output panel.

```
function checkAtSymbol(my_str:String):Boolean {
  for (var i = 0; i<my_str.length; i++) {
    if (my_str.charAt(i) == "@") {
      return true;
    }
  }
  return false;
}

trace(checkAtSymbol("dog@house.net")); // output: true
trace(checkAtSymbol("Chris")); // output: false
```

An example is also in the Strings.fla file in the HelpExamples folder. The following list gives typical paths to this folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/
-

String.slice()

Availability

Flash Player 5.

Usage

`my_str``.slice(`*`start:Number`*`, [`*`end:Number`*`]) : `*`String`*

Parameters

`start` A number; the zero-based index of the starting point for the slice. If *`start`* is a negative number, the starting point is determined from the end of the string, where -1 is the last character.

`end` A number; an integer that is 1+ the index of the ending point for the slice. The character indexed by the *`end`* parameter is not included in the extracted string. If this parameter is omitted, `String.length` is used. If `end` is a negative number, the ending point is determined by counting back from the end of the string, where -1 is the last character.

Returns

A string; a substring of the specified string.

Description

Method; returns a string that includes the *`start`* character and all characters up to, but not including, the *`end`* character. The original String object is not modified. If the *`end`* parameter is not specified, the end of the substring is the end of the string. If the character indexed by *`start`* is the same as or to the right of the character indexed by *`end`*, the method returns an empty string.

Example

The following example creates a variable, `my_str`, assigns it a String value, and then calls the `slice()` method using a variety of values for both the *`start`* and *`end`* parameters. Each call to `slice()` is wrapped in a `trace()` statement that displays the output in the Output panel.

```
// Index values for the string literal
// positive index:  0 1 2 3 4
// string:          L o r e m
// negative index: -5-4-3-2-1

var my_str:String = "Lorem";

// slice the first character
trace("slice(0,1): "+my_str.slice(0, 1)); // output: slice(0,1): L
trace("slice(-5,1): "+my_str.slice(-5, 1)); // output: slice(-5,1): L

// slice the middle three characters
trace("slice(1,4): "+my_str.slice(1, 4)); // slice(1,4): ore
trace("slice(1,-1): "+my_str.slice(1, -1)); // slice(1,-1): ore

// slices that return empty strings because start is not to the left of end
trace("slice(1,1): "+my_str.slice(1, 1)); // slice(1,1):
trace("slice(3,2): "+my_str.slice(3, 2)); // slice(3,2):
trace("slice(-2,2): "+my_str.slice(-2, 2)); // slice(-2,2):
```

```
// slices that omit the end parameter use String.length, which equals 5
trace("slice(0): "+my_str.slice(0)); // slice(0): Lorem
trace("slice(3): "+my_str.slice(3)); // slice(3): em
```

An example is also in the Strings.fla file in the HelpExamples folder. The following list gives typical paths to this folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/
-

See also

`String.substr()`, `String.substring()`

String.split()

Availability

Flash Player 5.

Usage

my_str.split("*delimiter*":*String*, [*limit*:*Number*]) : *Number*

Parameters

delimiter A string; the character or string at which *my_str* splits.

limit A number; the number of items to place into the array. This parameter is optional.

Returns

An array; an array containing the substrings of *my_str*.

Description

Method; splits a String object into substrings by breaking it wherever the specified *delimiter* parameter occurs and returns the substrings in an array. If you use an empty string ("") as a delimiter, each character in the string is placed as an element in the array.

If the *delimiter* parameter is undefined, the entire string is placed into the first element of the returned array.

Example

The following example returns an array with five elements:

```
var my_str:String = "P,A,T,S,Y";
var my_array:Array = my_str.split(",");
for (var i = 0; i<my_array.length; i++) {
  trace(my_array[i]);
}
/* output:
  P
  A
  T
  S
  Y
*/
```

The following example returns an array with two elements, "P" and "A":

```
var my_str:String = "P,A,T,S,Y";
var my_array:Array = my_str.split(",", 2);
trace(my_array); // output: P,A
```

The following example shows that if you use an empty string ("") for the *delimiter* parameter, each character in the string is placed as an element in the array:

```
var my_str:String = new String("Joe");
var my_array:Array = my_str.split("");
for (var i = 0; i<my_array.length; i++) {
  trace(my_array[i]);
}
```

```
/* output:
  J
  o
  e
*/
```

An example is also in the Strings.fla file in the HelpExamples folder. The following list gives typical paths to this folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/
-

See Also

`Array.join()`

String.substr()

Availability

Flash Player 5.

Usage

my_str.substr(*start:Number*, [*length:Number*]) *: String*

Parameters

start A number; an integer that indicates the position of the first character in *my_str* to be used to create the substring. If *start* is a negative number, the starting position is determined from the end of the string, where the -1 is the last character.

length A number; the number of characters in the substring being created. If *length* is not specified, the substring includes all the characters from the start to the end of the string.

Returns

A string; a substring of the specified string.

Description

Method; returns the characters in a string from the index specified in the *start* parameter through the number of characters specified in the *length* parameter. The substr method does not change the string specified by *my_str*; it returns a new string.

Example

The following example creates a new string, my_str and uses substr() to return the second word in the string; first, using a positive *start* parameter, and then using a negative *start* parameter:

```
var my_str:String = new String("Hello world");
var mySubstring:String = new String();
mySubstring = my_str.substr(6,5);
trace(mySubstring); // output: world

mySubstring = my_str.substr(-5,5);
trace(mySubstring); // output: world
```

An example is also in the Strings.fla file in the HelpExamples folder. The following list gives typical paths to this folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/
-

String.substring()

Availability

Flash Player 5.

Usage

my_str.substring(*start:Number*, [*end:Number*]) *: String*

Parameters

start A number; an integer that indicates the position of the first character of *my_str* used to create the substring. Valid values for *start* are 0 through String.length - 1. If *start* is a negative value, 0 is used.

end A number; an integer that is 1+ the index of the last character in *my_str* to be extracted. Valid values for *end* are 1 through String.length. The character indexed by the *end* parameter is not included in the extracted string. If this parameter is omitted, String.length is used. If this parameter is a negative value, 0 is used.

Returns

String: a substring of the specified string.

Description

Method; returns a string comprising the characters between the points specified by the *start* and *end* parameters. If the *end* parameter is not specified, the end of the substring is the end of the string. If the value of *start* equals the value of *end*, the method returns an empty string. If the value of *start* is greater than the value of *end*, the parameters are automatically swapped before the function executes and the original value is unchanged.

Example

The following example shows how to use substring():

```
var my_str:String = "Hello world";
var mySubstring:String = my_str.substring(6,11);
trace(mySubstring); // output: world
```

The following example shows what happens if a negative *start* parameter is used:

```
var my_str:String = "Hello world";
var mySubstring:String = my_str.substring(-5,5);
trace(mySubstring); // output: Hello
```

An example is also in the Strings.fla file in the Examples folder. The following list gives typical paths to this folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/
-

String.toLowerCase()

Availability

Flash Player 5.

Usage

my_str.toLowerCase() : *String*

Parameters

None.

Returns

A string.

Description

Method; returns a copy of the String object, with all uppercase characters converted to lowercase. The original value is unchanged.

Example

The following example creates a string with all uppercase characters and then creates a copy of that string using `toLowerCase()` to convert all uppercase characters to lowercase characters:

```
var upperCase:String = "LOREM IPSUM DOLOR";
var lowerCase:String = upperCase.toLowerCase();
trace("upperCase: " + upperCase); // output: upperCase: LOREM IPSUM DOLOR
trace("lowerCase: " + lowerCase); // output: lowerCase: lorem ipsum dolor
```

An example is also in the Strings.fla file in the HelpExamples folder. The following list gives typical paths to this folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/
-

See Also

`String.toUpperCase()`

String.toUpperCase()

Availability

Flash Player 5.

Usage

```
my_str.toUpperCase() : String
```

Parameters

None.

Returns

A string.

Description

Method; returns a copy of the String object, with all lowercase characters converted to uppercase. The original value is unchanged.

Example

The following example creates a string with all lowercase characters and then creates a copy of that string using `toUpperCase()`:

```
var lowerCase:String = "lorem ipsum dolor";
var upperCase:String = lowerCase.toUpperCase();
trace("lowerCase: " + lowerCase); // output: lowerCase: lorem ipsum dolor
trace("upperCase: " + upperCase); // output: upperCase: LOREM IPSUM DOLOR
```

An example is also found in the Strings.fla file in the HelpExamples folder. The following list gives typical paths to this folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/
-

See also

`String.toLowerCase()`

" " (string delimiter)

Availability

Flash Player 4.

Usage

`"text"`

Parameters

text	A sequence of zero or more characters.

Returns

Nothing.

Description

String delimiter; when used before and after characters, quotation marks ("") indicate that the characters have a literal value and are considered a *string*, not a variable, numerical value, or other ActionScript element.

Example

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

```
var yourGuess:String = "Prince Edward Island";
submit_btn.onRelease = function() {
  trace(yourGuess);
};

// displays Prince Edward Island in the Output panel
```

See also

`String class`, `String()`

super

Availability

Flash Player 6.

Usage

```
super.method([arg1, ..., argN])
super([arg1, ..., argN])
```

Parameters

`method` The method to invoke in the superclass.

`arg1` Optional parameters that are passed to the superclass version of the method (syntax 1) or to the constructor function of the superclass (syntax 2).

Returns

Both forms invoke a function. The function can return any value.

Description

Operator: the first syntax style can be used within the body of an object method to invoke the superclass version of a method and can optionally pass parameters (`arg1` ... `argN`) to the superclass method. This is useful for creating subclass methods that add behavior to superclass methods but also invoke the superclass methods to perform their original behavior.

The second syntax style can be used within the body of a constructor function to invoke the superclass version of the constructor function and can optionally pass parameters to it. This is useful for creating a subclass that performs additional initialization but also invokes the superclass constructor to perform superclass initialization.

Example

In the following example, you create two classes. You use the `super` keyword in the Sock class to call functions in the parent class (Clothes). Although both the Socks and Clothes classes have a method called `getColor()`, using `super` lets you specifically reference the base class's methods and properties. Create a new AS file called Clothes.as, and enter the following code:

```
class Clothes {
  private var color:String;
  function Clothes(param_color) {
    this.color = param_color;
    trace("[Clothes] I am the constructor");
  }
  function getColor():String {
    trace("[Clothes] I am getColor");
    return this.color;
  }
  function setColor(param_color:String):Void {
    this.color = param_color;
    trace("[Clothes] I am setColor");
  }
}
```

Then create a new class called Socks that extends the Clothes class, as shown in the following example:

```
class Socks extends Clothes {
  private var color:String;
  function Socks(param_color:String) {
    this.color = param_color;
    trace("[Socks] I am the constructor");
  }
  function getColor():String {
    trace("[Socks] I am getColor");
    return super.getColor();
  }
  function setColor(param_color:String):Void {
    this.color = param_color;
    trace("[Socks] I am setColor");
  }
}
```

Then create a new AS or FLA file and enter the following ActionScript in the document:

```
import Socks;
var mySock:Socks = new Socks("maroon");
trace(" -> "+mySock.getColor());
mySock.setColor("Orange");
trace(" -> "+mySock.getColor());
```

The following result is displayed in the Output panel:

```
[Clothes] I am the constructor
[Socks] I am the constructor
[Socks] I am getColor
[Clothes] I am getColor
-> maroon
[Socks] I am setColor
[Socks] I am getColor
[Clothes] I am getColor
-> Orange
```

If you forgot to put the `super` keyword in the Sock class's `getColor()` method, then the `getColor()` method could call itself repeatedly, which would cause the script to fail because of infinite recursion problems. The Output panel would display the following error if you didn't use the `super` keyword:

```
[Socks] I am getColor
[Socks] I am getColor
...
[Socks] I am getColor
256 levels of recursion were exceeded in one action list.
This is probably an infinite loop.
Further execution of actions has been disabled in this movie.
```

switch

Availability

Flash Player 4.

Usage

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

Parameters

`expression` Any expression.

`caseClause` A `case` keyword followed by an expression, a colon, and a group of statements to execute if the expression matches the switch `expression` parameter using strict equality (===).

`defaultClause` A `default` keyword followed by statements to execute if none of the case expressions match the switch `expression` parameter strict equality (===).

Returns

Nothing.

Description

Statement; creates a branching structure for ActionScript statements. As with the `if` statement, the `switch` statement tests a condition and executes statements if the condition returns a value of `true`. All switch statements should include a default case. The default case should include a break statement that prevents a fall-through error if another case is added later. When a case falls through, it doesn't have a break statement.

Example

In the following example, if the `String.fromCharCode(Key.getAscii())` parameter evaluates to `A`, the `trace()` statement that follows `case "A"` executes; if the parameter evaluates to `a`, the `trace()` statement that follows `case "a"` executes; and so on. If no `case` expression matches the `String.fromCharCode(Key.getAscii())` parameter, the `trace()` statement that follows the `default` keyword executes.

```
var listenerObj:Object = new Object();
listenerObj.onKeyDown = function() {
  switch (String.fromCharCode(Key.getAscii())) {
  case "A" :
    trace("you pressed A");
    break;
  case "a" :
    trace("you pressed a");
    break;
  case "E" :
  case "e" :
    trace("you pressed E or e");
    break;
  case "I" :
  case "i" :
    trace("you pressed I or i");
    break;
  default :
    trace("you pressed some other key");
  }
};
Key.addListener(listenerObj);
```

See also

`=== (strict equality)`, `break`, `case`, `default`, `if`

System.capabilities object

Availability

Flash Player 6.

Description

You can use the System.capabilities object to determine the abilities of the system and player hosting a SWF file, which lets you tailor content for different formats. For example, the screen of a cell phone (black and white, 100 square pixels) is different than the 1000-square-pixel color PC screen. To provide appropriate content to as many users as possible, you can use the System.capabilities object to determine the type of device a user has. You can then either specify to the server to send different SWF files based on the device capabilities or tell the SWF file to alter its presentation based on the capabilities of the device.

You can send capabilities information using a `GET` or `POST` HTTP method. The following example shows a server string for a computer that has MP3 support, 1600 x 1200 pixel resolution, is running Windows XP, and Flash Player 7 (7.0.19.0):

```
"A=t&SA=t&SV=t&EV=t&MP3=t&AE=t&VE=t&ACC=f&PR=t&SP=t&SB=f&DEB=t&V=WIN%207%2C0%2
  C19%2C0&M=Macromedia%20Windows&R=1600x1200&DP=72&COL=color&AR=1.0&OS=Window
  s%20XP&L=en&PT=External&AVD=f&LFD=f&WD=f"
```

Property summary for the System.capabilities object

All properties of the System.capabilities object are read-only.

Property	Description	Server string
`System.capabilities.avHardwareDisable`	Specifies whether the user's camera and microphone are enabled or disabled.	`AVD`
`System.capabilities.hasAccessibility`	Indicates whether the player is running on a system that supports communication between Flash Player and accessibility aids.	`ACC`
`System.capabilities.hasAudio`	Indicates whether the player is running on a system that has audio capabilities.	`A`
`System.capabilities.hasAudioEncoder`	Indicates whether the player is running on a system that can encode an audio stream, such as that coming from a microphone.	`AE`
`System.capabilities.hasEmbeddedVideo`	Indicates whether the player is running on a system that supports embedded video.	`EV`
`System.capabilities.hasMP3`	Indicates whether the player is running on a system that has an MP3 decoder.	`MP3`
`System.capabilities.hasPrinting`	Indicates whether the player is running on a system that supports printing.	`PR`

Property	Description	Server string
`System.capabilities.hasScreenBroadcast`	Indicates whether the player supports the development of screen broadcast applications to be run through the Flash Communication Server.	`SB`
`System.capabilities.hasScreenPlayback`	Indicates whether the player supports the playback of screen broadcast applications that are being run through the Flash Communication Server.	`SP`
`System.capabilities.hasStreamingAudio`	Indicates whether the player can play streaming audio.	`SA`
`System.capabilities.hasStreamingVideo`	Indicates whether the player can play streaming video.	`SV`
`System.capabilities.hasVideoEncoder`	Indicates whether the player can encode a video stream, such as that coming from a web camera.	`VE`
`System.capabilities.isDebugger`	Indicates whether the player is an officially released version or a special debugging version.	`DEB`
`System.capabilities.language`	Indicates the language of the system on which the player is running.	`L`
`System.capabilities.localFileReadDisable`	Specifies whether the player will attempt to read anything (including the first SWF file the player launches with) from the user's hard disk.	`LFD`
`System.capabilities.manufacturer`	Indicates the manufacturer of Flash Player.	`M`
`System.capabilities.os`	Indicates the operating system hosting Flash Player.	`OS`
`System.capabilities.pixelAspectRatio`	Indicates the pixel aspect ratio of the screen.	`AR`
`System.capabilities.playerType`	Indicates the type of player: stand-alone, external, plug-in, or ActiveX.	`PT`
`System.capabilities.screenColor`	Indicates whether the screen is color, grayscale, or black and white.	`COL`
`System.capabilities.screenDPI`	Indicates the dots-per-inch screen resolution, in pixels.	`DP`
`System.capabilities.screenResolutionX`	Indicates the horizontal size of the screen.	`R`
`System.capabilities.screenResolutionY`	Indicates the vertical size of the screen.	`R`
`System.capabilities.serverString`	A URL-encoded string that specifies values for each `System.capabilities` property.	n/a
`System.capabilities.version`	A string containing Flash Player version and platform information.	`V`

System.capabilities.avHardwareDisable

Availability

Flash Player 7.

Usage

```
System.capabilities.avHardwareDisable:Boolean
```

Description

Read-only property; a Boolean value that specifies whether access to the user's camera and microphone has been administratively prohibited (`true`) or allowed (`false`). The server string is `AVD`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.avHardwareDisable);
```

See also

`Camera.get()`, `Microphone.get()`, `System.showSettings()`

System.capabilities.hasAccessibility

Availability

Flash Player 6.

Usage

```
System.capabilities.hasAccessibility:Boolean
```

Description

Read-only property: a Boolean value that is `true` if the player is running in an environment that supports communication between Flash Player and accessibility aids; `false` otherwise. The server string is `ACC`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.hasAccessibility);
```

See also

`Accessibility.isActive()`, `Accessibility.updateProperties()`, `_accProps`

System.capabilities.hasAudio

Availability

Flash Player 6.

Usage

```
System.capabilities.hasAudio:Boolean
```

Description

Read-only property: a Boolean value that is `true` if the player is running on a system that has audio capabilities; `false` otherwise. The server string is `A`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.hasAudio);
```

System.capabilities.hasAudioEncoder

Availability

Flash Player 6.

Usage

```
System.capabilities.hasAudioEncoder:Boolean
```

Description

Read-only property: a Boolean value that is `true` if the player can encode an audio stream, such as that coming from a microphone; `false` otherwise. The server string is `AE`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.hasAudioEncoder);
```

System.capabilities.hasEmbeddedVideo

Availability

Flash Player 6 r65.

Usage

```
System.capabilities.hasEmbeddedVideo:Boolean
```

Description

Read-only property: a Boolean value that is `true` if the player is running on a system that supports embedded video; `false` otherwise. The server string is `EV`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.hasEmbeddedVideo);
```

System.capabilities.hasMP3

Availability

Flash Player 6.

Usage

```
System.capabilities.hasMP3:Boolean
```

Description

Read-only property: a Boolean value that is `true` if the player is running on a system that has an MP3 decoder; `false` otherwise. The server string is `MP3`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.hasMP3);
```

System.capabilities.hasPrinting

Availability

Flash Player 6 r65.

Usage

```
System.capabilities.hasPrinting:Boolean
```

Description

Read-only property: a Boolean value that is `true` if the player is running on a system that supports printing; `false` otherwise. The server string is `PR`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.hasPrinting);
```

System.capabilities.hasScreenBroadcast

Availability

Flash Player 6 r79.

Usage

```
System.capabilities.hasScreenBroadcast:Boolean
```

Description

Read-only property: a Boolean value that is `true` if the player supports the development of screen broadcast applications to be run through the Flash Communication Server; `false` otherwise. The server string is `SB`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.hasScreenBroadcast);
```

System.capabilities.hasScreenPlayback

Availability

Flash Player 6 r79.

Usage

```
System.capabilities.hasScreenPlayback:Boolean
```

Description

Read-only property: a Boolean value that is `true` if the player supports the playback of screen broadcast applications that are being run through the Flash Communication Server; `false` otherwise. The server string is `SP`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.hasScreenPlayback);
```

System.capabilities.hasStreamingAudio

Availability

Flash Player 6 r65.

Usage

```
System.capabilities.hasStreamingAudio:Boolean
```

Description

Read-only property: a Boolean value that is `true` if the player can play streaming audio; `false` otherwise. The server string is `SA`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.hasStreamingAudio);
```

System.capabilities.hasStreamingVideo

Availability

Flash Player 6 r65.

Usage

```
System.capabilities.hasStreamingVideo:Boolean
```

Description

Read-only property: a Boolean value that is `true` if the player can play streaming video; `false` otherwise. The server string is `SV`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.hasStreamingVideo);
```

System.capabilities.hasVideoEncoder

Availability

Flash Player 6.

Usage

```
System.capabilities.hasVideoEncoder:Boolean
```

Description

Read-only property: a Boolean value that is `true` if the player can encode a video stream, such as that coming from a web camera; `false` otherwise. The server string is `VE`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.hasVideoEncoder);
```

System.capabilities.isDebugger

Availability

Flash Player 6.

Usage

```
System.capabilities.isDebugger:Boolean
```

Description

Read-only property; a Boolean value that indicates whether the player is an officially released version (`false`) or a special debugging version (`true`). The server string is `DEB`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.isDebugger);
```

System.capabilities.language

Availability

Flash Player 6. Behavior changed in Flash Player 7.

Usage

```
System.capabilities.language:String
```

Description

Read-only property; indicates the language of the system on which the player is running. This property is specified as a lowercase two-letter language code from ISO 639-1. For Chinese, an additional uppercase two-letter country code subtag from ISO 3166 distinguishes between Simplified and Traditional Chinese. The languages themselves are named with the English tags. For example, `fr` specifies French.

This property changed in two ways for Flash Player 7. First, the language code for English systems no longer includes the country code. In Flash Player 6, all English systems return the language code and the two-letter country code subtag (`en-US`). In Flash Player 7, English systems return only the language code (`en`). Second, on Microsoft Windows systems this property now returns the User Interface (UI) Language. In Flash Player 6 on the Microsoft Windows platform, `System.capabilities.language` returns the User Locale, which controls settings for formatting dates, times, currency and large numbers. In Flash Player 7 on the Microsoft Windows platform, this property now returns the UI Language, which refers to the language used for all menus, dialog boxes, error messages and help files.

Language	Tag
Czech	`cs`
Danish	`da`
Dutch	`nl`
English	`en`
Finnish	`fi`
French	`fr`
German	`de`
Hungarian	`hu`
Italian	`it`
Japanese	`ja`
Korean	`ko`
Norwegian	`no`
Other/unknown	`xu`
Polish	`pl`
Portuguese	`pt`

Language	Tag
Russian	`ru`
Simplified Chinese	`zh-CN`
Spanish	`es`
Swedish	`sv`
Traditional Chinese	`zh-TW`
Turkish	`tr`

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.language);
```

System.capabilities.localFileReadDisable

Availability

Flash Player 7.

Usage

```
System.capabilities.localFileReadDisable:Boolean
```

Description

Read-only property; a Boolean value that indicates whether read access to the user's hard disk has been administratively prohibited (`true`) or allowed (`false`). If set to `true`, Flash Player will be unable to read files (including the first SWF file that Flash Player launches with) from the user's hard disk. For example, attempts to read a file on the user's hard disk using `XML.load()`, `LoadMovie()`, or `LoadVars.load()` will fail if this property is set to `true`.

Reading runtime shared libraries will also be blocked if this property is set to `true`, but reading local shared objects is allowed without regard to the value of this property. The server string is `LFD`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.localFileReadDisable);
```

System.capabilities.manufacturer

Availability

Flash Player 6.

Usage

```
System.capabilities.manufacturer:String
```

Description

Read-only property; a string that indicates the manufacturer of Flash Player, in the format `"Macromedia OSName"` (*`OSName`* could be `"Windows"`, `"Macintosh"`, `"Linux"`, or `"Other OS Name"`). The server string is `M`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.manufacturer);
```

System.capabilities.os

Availability

Flash Player 6.

Usage

```
System.capabilities.os:String
```

Description

Read-only property; a string that indicates the current operating system. The `os` property can return the following strings: `"Windows XP"`, `"Windows 2000"`, `"Windows NT"`, `"Windows 98/ME"`, `"Windows 95"`, `"Windows CE"` (available only in Flash Player SDK, not in the desktop version), `"Linux"`, and `"MacOS"`. The server string is `OS`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.os);
```

System.capabilities.pixelAspectRatio

Availability

Flash Player 6.

Usage

```
System.capabilities.pixelAspectRatio:Number
```

Description

Read-only property; an integer that indicates the pixel aspect ratio of the screen. The server string is `AR`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.pixelAspectRatio);
```

System.capabilities.playerType

Availability

Flash Player 7.

Usage

```
System.capabilities.playerType;String
```

Description

Read-only property; a string that indicates the type of player. This property can have one of the following values:

- `"StandAlone"` for the Flash StandAlone Player
- `"External"` for the Flash Player version used by test movie mode,
- `"PlugIn"` for the Flash Player browser plug-in
- `"ActiveX"` for the Flash Player ActiveX Control used by Microsoft Internet Explorer

The server string is `PT`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.playerType);
```

System.capabilities.screenColor

Availability

Flash Player 6.

Usage

```
System.capabilities.screenColor:String
```

Description

Read-only property; a string that indicates the screen color. This property can have the value `"color"`, `"gray"` or `"bw"`, which represents color, grayscale, and black and white, respectively. The server string is `COL`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.screenColor);
```

System.capabilities.screenDPI

Availability

Flash Player 6.

Usage

```
System.capabilities.screenDPI:Number
```

Description

Read-only property; a number that indicates the dots-per-inch (dpi) resolution of the screen, in pixels. The server string is `DP`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.screenDPI);
```

System.capabilities.screenResolutionX

Availability

Flash Player 6.

Usage

```
System.capabilities.screenResolutionX:Number
```

Description

Read-only property; an integer that indicates the maximum horizontal resolution of the screen. The server string is `R` (which returns both the width and height of the screen).

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.screenResolutionX);
```

System.capabilities.screenResolutionY

Availability

Flash Player 6.

Usage

```
System.capabilities.screenResolutionY:Number
```

Description

Read-only property; an integer that indicates the maximum vertical resolution of the screen. The server string is `R` (which returns both the width and height of the screen).

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.screenResolutionY);
```

System.capabilities.serverString

Availability

Flash Player 6.

Usage

```
System.capabilities.serverString:String
```

Description

Read-only property; a URL-encoded string that specifies values for each `System.capabilities` property, as shown in the following example:

```
A=t&SA=t&SV=t&EV=t&MP3=t&AE=t&VE=t&ACC=f&PR=t&SP=t&SB=f&DEB=t&V=WIN%207%2C0%2C
19%2C0&M=Macromedia%20Windows&R=1600x1200&DP=72&COL=color&AR=1.0&OS=Windows%20
XP&L=en&PT=External&AVD=f&LFD=f&WD=f
```

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.serverString);
```

System.capabilities.version

Availability

Flash Player 6.

Usage

```
System.capabilities.version:String
```

Description

Read-only property; a string containing the Flash Player platform and version information (for example, `"WIN 7,0,19,0"`). The server string is `V`.

Example

The following example traces the value of this read-only property:

```
trace(System.capabilities.version);
```

System.security object

Availability

Flash Player 6.

Description

This object contains methods that specify how SWF files in different domains can communicate with each other.

Method summary for the System.security object

Method	Description
`System.security.allowDomain()`	Lets SWF files in the identified domains access objects and variables in the calling SWF file or in any other SWF file from the same domain as the calling SWF file.
`System.security.allowInsecureDomain()`	Lets SWF files in the identified domains access objects and variables in the calling SWF file, which is hosted using the HTTPS protocol.
`System.security.loadPolicyFile()`	Loads a cross-domain policy file from a specified location.

System.security.allowDomain()

Availability

Flash Player 6; behavior changed in Flash Player 7.

Usage

```
System.security.allowDomain("domain1":String, "domain2", ... "domainN") : Void
```

Parameters

domain1, domain2, ... domainN Strings that specify domains that can access objects and variables in the file containing the `System.Security.allowDomain()` call. The domains can be formatted in the following ways:

- `"domain.com"`
- `"http://domain.com"`
- `"http://IPaddress"`

Description

Method; lets SWF files and HTML files in the identified domains access objects and variables in the calling SWF file or in any other SWF file from the same domain as the calling SWF file.

In files playing in Flash Player 7 or later, the parameter(s) passed must follow exact-domain naming rules. For example, to allow access by SWF files hosted at either www.domain.com or store.domain.com, both domain names must be passed:

```
// For Flash Player 6
System.security.allowDomain("domain.com");
// Corresponding commands to allow access by SWF files
// that are running in Flash Player 7 or later
System.security.allowDomain("www.domain.com", "store.domain.com");
```

Also, for files running in Flash Player 7 or later, you can't use this method to let SWF files hosted using a secure protocol (HTTPS) allow access from SWF files hosted in nonsecure protocols; you must use System.security.allowInsecureDomain() instead.

Occasionally, you might encounter the following situation: You load a child SWF file from a different domain and want to allow the child SWF file to script the parent SWF file, but you don't know the final domain from which the child SWF file will come. This can happen, for example, when you use load-balancing redirects or third-party servers.

In this situation, you can use the MovieClip._url property as an argument to this method. For example, if you load a SWF file into `my_mc`, you can call `System.security.allowDomain(my_mc._url)`.

If you do this, be sure to wait until the SWF file in `my_mc` is loaded, because the `_url` property does not have its final, correct value until the file is completely loaded. The best way to determine when a child SWF finishes loading is to use MovieClipLoader.onLoadComplete.

The opposite situation can also occur; that is, you might create a child SWF file that wants to allow its parent to script it, but doesn't know what the domain of its parent will be. In this situation, call `System.security.allowDomain(_parent._url)` from the child SWF. In this situation, you don't have to wait for the parent SWF file to load; the parent will already be loaded by the time the child loads.

Example

The SWF file located at www.macromedia.com/MovieA.swf contains the following lines:

```
System.security.allowDomain("www.shockwave.com");
loadMovie("http://www.shockwave.com/MovieB.swf", my_mc);
```

Because MovieA contains the `allowDomain()` command, MovieB can access the objects and variables in MovieA. If MovieA didn't contain this command, the Flash security implementation would prevent MovieB from accessing MovieA's objects and variables.

See also

`MovieClip._url`, `MovieClipLoader.onLoadComplete`, `_parent`, `System.security.allowInsecureDomain()`

System.security.allowInsecureDomain()

Availability

Flash Player 7.

Usage

`System.security.allowInsecureDomain("`*`domain`*`":`*`String`*`) : `*`Void`*

Parameters

`domain` A string; an exact domain name, such as www.myDomainName.com or store.myDomainName.com.

Returns

Nothing.

Description

Method; lets SWF files and HTML files in the identified domains access objects and variables in the calling SWF file, which is hosted using the HTTPS protocol. It also lets the SWF files in the identified domains access any other SWF files in the same domain as the calling SWF file.

By default, SWF files hosted using the HTTPS protocol can be accessed only by other SWF files hosted using the HTTPS protocol. This implementation maintains the integrity provided by the HTTPS protocol.

Macromedia does not recommend using this method to override the default behavior because it compromises HTTPS security. However, you might need to do so, for example, if you must permit access to HTTPS files published for Flash Player 7 or later from HTTP files published for Flash Player 6.

A SWF file published for Flash Player 6 can use System.security.allowDomain() to permit HTTP to HTTPS access. However, because security is implemented differently in Flash Player 7, you must use `System.Security.allowInsecureDomain()` to permit such access in SWF files published for Flash Player 7 or later.

Note: It is sometimes necessary to call `System.security.allowInsecureDomain()` with an argument that exactly matches the domain of the SWF file in which this call appears. This is different from `System.security.allowDomain()`, which is never necessary to call with a SWF file's own domain as an argument. The reason this is sometimes necessary with `System.security.allowInsecureDomain()` is that, by default, a SWF file at http://foo.com is not allowed to script a SWF file at https://foo.com, even though the domains are identical.

Example

In the following example, you host a math test on a secure domain so that only registered students can access it. You have also developed a number of SWF files that illustrate certain concepts, which you host on an insecure domain. You want students to access the test from the SWF file that contains information about a concept.

```
// This SWF file is at https://myEducationSite.somewhere.com/mathTest.swf
// Concept files are at http://myEducationSite.somewhere.com
System.security.allowInsecureDomain("myEducationSite.somewhere.com");
```

See also

`System.security.allowDomain()`, `System.exactSettings`

System.security.loadPolicyFile()

Availability

Flash Player 7 (7.0.19.0)

Usage

```
System.security.loadPolicyFile(url:String) : Void
```

Parameters

`url` A string; the URL where the cross-domain policy file to be loaded is located.

Returns

Nothing.

Description

Method; loads a cross-domain policy file from a location specified by the `url` parameter. Flash Player uses policy files as a permission mechanism to permit Flash movies to load data from servers other than their own.

Flash Player 7.0.14.0 looked for policy files in only one location: /crossdomain.xml on the server to which a data-loading request was being made. For an XMLSocket connection attempt, Flash Player 7.0.14.0 looked for /crossdomain.xml on an HTTP server on port 80 in the subdomain to which the XMLSocket connection attempt was being made. Flash Player 7.0.14.0 (and all earlier players) also restricted XMLSocket connections to ports 1024 and higher.

With the addition of System.security.loadPolicyFile(), Flash Player 7.0.19.0 can load policy files from arbitrary locations, as shown in the following example:

```
System.security.loadPolicyFile("http://foo.com/sub/dir/pf.xml");
```

This causes Flash Player to retrieve a policy file from the specified URL. Any permissions granted by the policy file at that location will apply to all content at the same level or lower in the virtual directory hierarchy of the server. The following code continues the previous example:

```
loadVariables("http://foo.com/sub/dir/vars.txt") // allowed
loadVariables("http://foo.com/sub/dir/deep/vars2.txt") // allowed
loadVariables("http://foo.com/elsewhere/vars3.txt") // not allowed
```

You can load any number of policy files using `loadPolicyFile()`. When considering a request that requires a policy file, Flash Player always waits for the completion of any policy file downloads before denying a request. As a final fallback, if no policy file specified with `loadPolicyFile()` authorizes a request, Flash Player consults the original default location, /crossdomain.xml.

Using the `xmlsocket` protocol along with a specific port number, lets you retrieve policy files directly from an XMLSocket server, as shown in the following example:

```
System.security.loadPolicyFile("xmlsocket://foo.com:414");
```

This causes Flash Player to attempt to retrieve a policy file from the specified host and port. Any port can be used, not only ports 1024 and higher. Upon establishing a connection with the specified port, Flash Player transmits `<cross-domain-request/>`, terminated by a `null` byte. An XMLSocket server can be configured to serve both policy files and normal XMLSocket connections over the same port, in which case the server should wait for `<cross-domain-request/>` before transmitting a policy file. A server can also be set up to serve policy files over a separate port from normal connections, in which case it can send a policy file as soon as a connection is established on the dedicated policy file port. The server must send a null byte to terminate a policy file, and may thereafter close the connection; if the server does not close the connection, Flash Player will do so upon receiving the terminating `null` byte.

A policy file served by an XMLSocket server has the same syntax as any other policy file, except that it must also specify the ports to which access is granted. When a policy file comes from a port lower than 1024, it can grant access to any ports; when a policy file comes from port 1024 or higher, it can grant access only to other ports 1024 and higher. The allowed ports are specified in a `"to-ports"` attribute in the `<allow-access-from>` tag. Single port numbers, port ranges, and wildcards are all allowed. The following example shows an XMLSocket policy file:

```
<cross-domain-policy>
  <allow-access-from domain="*" to-ports="507" />
  <allow-access-from domain="*.foo.com" to-ports="507,516" />
  <allow-access-from domain="*.bar.com" to-ports="516-523" />
  <allow-access-from domain="www.foo.com" to-ports="507,516-523" />
  <allow-access-from domain="www.bar.com" to-ports="*" />
</cross-domain-policy>
```

A policy file obtained from the old default location—/crossdomain.xml on an HTTP server on port 80—implicitly authorizes access to all ports 1024 and above. There is no way to retrieve a policy file to authorize XMLSocket operations from any other location on an HTTP server; any custom locations for XMLSocket policy files must be on an XMLSocket server.

Because the ability to connect to ports lower than 1024 is new, a policy file loaded with `loadPolicyFile()` must always authorize this, even when a movie clip is connecting to its own subdomain.

System class

Availability

Flash Player 6.

Description

This is a top-level class that contains the capabilities object (see System.capabilities object), the security object (see System.security object), and the methods, properties, and event handlers listed in the following table.

Method summary for the System class

Method	Description
`System.setClipboard()`	Replaces the contents of the system Clipboard with a text string.
`System.showSettings()`	Displays a Flash Player Settings panel.

Property summary for the System class

Method	Description
`System.exactSettings`	Specifies whether to use superdomain or exact-domain matching rules when accessing local settings.
`System.useCodepage`	Tells Flash Player whether to use Unicode or the traditional code page of the operating system running the player to interpret external text files.

Event handler summary for the System class

Method	Description
`System.onStatus`	Provides a super event handler for certain objects.

System.exactSettings

Availability

Flash Player 7 or later.

Usage

```
System.exactSettings:Boolean
```

Description

Property; a Boolean value that specifies whether to use superdomain (`false`) or exact domain (`true`) matching rules when accessing local settings (such as camera or microphone access permissions) or locally persistent data (shared objects). The default value is `true` for files published for Flash Player 7 or later, and `false` for files published for Flash Player 6.

If this value is `true`, the settings and data for a SWF file hosted at here.xyz.com are stored in a directory called here.xyz.com, the settings and data for a SWF file hosted at there.xyz.com are stored in a directory called there.xyz.com, and so on. If this value is `false`, the settings and data for SWF files hosted at here.xyz.com, there.xyz.com, and xyz.com are shared, and are all stored in a directory called xyz.com.

If some of your files set this property to `false` and others set it to `true`, you might find that SWF files in different subdomains share settings and data. For example, if this property is `false` in a SWF file hosted at here.xyz.com and `true` in a SWF file hosted at xyz.com, both files will use the same settings and data—namely, those in the xyz.com directory. If this isn't the behavior you want, ensure that you set this property in each file to correctly represent where you want to store settings and data.

If you want to change this property from its default value, do so in the first frame of your document. The property can't be changed after any activity that requires access to local settings, such as `System.showSettings()` or `SharedObject.getLocal()`.

If you use loadMovie(), MovieClip.loadMovie(), or MovieClipLoader.loadClip() to load one SWF file into another, all the files published for Flash Player 7 share a single value for `System.exactSettings`, and all the files published for Flash Player 6 share a single value for `System.exactSettings`. Therefore, if you specify a value for this property in one file published for a particular Player version, you should do so in all the files that you plan to load. If you load multiple files, the setting specified in the last file that's loaded overwrites any previously specified setting. For more information on how domain matching is implemented in Flash, see "Flash Player security features" in *Using ActionScript in Flash*.

Usually you should find that the default value of `System.exactSettings` is fine. Often your only requirement is that when a SWF file saves a shared object in one session, the same SWF file can retrieve the same shared object in a later session. This situation will always be true, regardless of the value of `System.exactSettings`. But you might want to change `System.exactSettings` from its default so that a SWF file published for Flash Player 7 or later can retrieve shared objects originally created by a SWF file published for Flash Player 6. Because the player has stored the shared objects created by the Flash Player 6 SWF file in a folder that's specific to the superdomain of that SWF file, you should use superdomain rules for shared object retrieval in your Flash Player 7 SWF file. This step requires specifying `System.exactSettings = false` in your Flash Player 7 SWF file. It is also possible that you might have SWF files that are published for Flash Player 6 and Flash Player 7 SWF files that share the same shared object data. In this case, simply pick a value for `System.exactSettings` (either `true` or `false`) and use it consistently in your Flash Player 6 and Flash Player 7 SWF files.

Example

The following example shows how to specify superdomain matching rules:

```
System.exactSettings = false;
```

See also

`SharedObject.getLocal()`, `System.showSettings()`

System.onStatus

Availability

Flash Player 6.

Usage

```
System.onStatus = function(InfoObject:Object) {
  // your statements
}
```

Description

Event handler: provides a super event handler for certain objects.

The LocalConnection, NetStream, and SharedObject classes provide an `onStatus` event handler that uses an information object for providing information, status, or error messages. To respond to this event handler, you must create a function to process the information object, and you must know the format and contents of the returned information object.

In addition to these specific `onStatus` methods, Flash also provides a super function called `System.onStatus`, which serves as a secondary error message handler. If an instance of the LocalConnection, NetStream, or SharedObject class passes an information object with a level property of "error", but you have not defined an `onStatus` function for that particular instance, then Flash uses the function you define for `System.onStatus` instead.

Note: The Camera and Microphone classes also have `onStatus` handlers but do not pass information objects with a level property of `"error"`. Therefore, `System.onStatus` is not called if you don't specify a function for these handlers.

Example

The following example shows how to create a `System.onStatus` function to process information objects when a class-specific `onStatus` function does not exist:

```
// Create generic function
System.onStatus = function(genericError:Object){
  // Your script would do something more meaningful here
  trace("An error has occurred. Please try again.");
}
```

The following example shows how to create an onStatus function for an instance of the NetStream class:

```
// Create function for NetStream object

videoStream_ns.onStatus = function(infoObject:Object) {
  if (infoObject.code == "NetStream.Play.StreamNotFound") {
    trace("Could not find video file.");
  }
}
```

See also

`Camera.onStatus`, `LocalConnection.onStatus`, `Microphone.onStatus`,
  `NetStream.onStatus`, `SharedObject.onStatus`

System.setClipboard()

Availability

SWF files published for Flash Player 6 or later, playing in Flash Player 7 or later.

Usage

`System.setClipboard(`*`string:String`*`) : `*`Boolean`*

Parameters

`string` A plain-text string of characters to place on the system Clipboard, replacing its current contents (if any).

Returns

A Boolean value: `true` if the text is successfully placed on the Clipboard; `false` otherwise.

Description

Method; replaces the contents of the Clipboard with a specified text string.

Example

The following example places the phrase `"Hello World"` onto the system Clipboard:

```
System.setClipboard("Hello world");
```

The following example creates two text fields at runtime, called `in_txt` and `out_txt`. When you select text in the `in_txt` field, you can click the `copy_btn` to copy the data to the Clipboard. Then you can paste the text into the `out_txt` field.

```
this.createTextField("in_txt", this.getNextHighestDepth(), 10, 10, 160, 120);
in_txt.multiline = true;
in_txt.border = true;
in_txt.text = "lorum ipsum...";
this.createTextField("out_txt", this.getNextHighestDepth(), 10, 140, 160,
  120);
out_txt.multiline = true;
out_txt.border = true;
out_txt.type = "input";

copy_btn.onRelease = function() {
  System.setClipboard(in_txt.text);
  Selection.setFocus("out_txt");
};
```

System.showSettings()

Availability

Flash Player 6.

Usage

```
System.showSettings([panel:Number]) : Void
```

Parameters

`panel` A number; an optional number that specifies which Flash Player Settings panel to display, as shown in the following table:

Value passed for `panel`	Settings panel displayed
None (parameter is omitted) or an unsupported value	The panel that was open the last time the user closed the Player Settings panel.
0	Privacy
1	Local Storage
2	Microphone
3	Camera

Returns

Nothing.

Description

Method; shows the specified Flash Player Settings panel, which lets users do any of the following actions:

- Allow or deny access to the camera and microphone
- Specify the local disk space available for shared objects
- Select a default camera and microphone
- Specify microphone gain and echo suppression settings

For example, if your application requires the use of a camera, you can tell the user to select Allow in the Privacy Settings panel and then issue a `System.showSettings(0)` command. (Ensure that your Stage size is at least 215 x 138 pixels; this is the minimum size Flash requires to display the panel.)

Example

The following example shows how to display the Flash Player Settings Local Storage panel:

```
System.showSettings(1);
```

See also

`Camera.get()`, `Microphone.get()`, `SharedObject.getLocal()`

System.useCodepage

Availability

Flash Player 6.

Usage

```
System.useCodepage:Boolean
```

Description

Property; a Boolean value that tells Flash Player whether to use Unicode or the traditional code page of the operating system running the player to interpret external text files. The default value of `System.useCodepage` is `false`.

- When the property is set to `false`, Flash Player interprets external text files as Unicode. (These files must be encoded as Unicode when you save them.)
- When the property is set to `true`, Flash Player interprets external text files using the traditional code page of the operating system running the player.

Text that you load as an external file (using the loadVariables() or getURL() statements, or the LoadVars class or XML class) must be encoded as Unicode when you save the text file in order for Flash Player to recognize it as Unicode. To encode external files as Unicode, save the files in an application that supports Unicode, such as Notepad on Windows 2000.

If you load external text files that are not Unicode-encoded, you should set `System.useCodepage` to `true`. Add the following code as the first line of code in the first frame of the SWF file that is loading the data:

```
System.useCodepage = true;
```

When this code is present, Flash Player interprets external text using the traditional code page of the operating system running Flash Player. This is generally CP1252 for an English Windows operating system and Shift-JIS for a Japanese operating system. If you set `System.useCodepage` to `true`, Flash Player 6 and later treat text as Flash Player 5 does. (Flash Player 5 treated all text as if it were in the traditional code page of the operating system running the player.)

If you set `System.useCodepage` to `true`, remember that the traditional code page of the operating system running the player must include the characters used in your external text file in order for the text to display. For example, if you load an external text file that contains Chinese characters, those characters cannot display on a system that uses the CP1252 code page because that code page does not include Chinese characters.

To ensure that users on all platforms can view external text files used in your SWF files, you should encode all external text files as Unicode and leave `System.useCodepage` set to `false` by default. This way, Flash Player 6 and later interprets the text as Unicode.

targetPath()

Availability

Flash Player 5.

Usage

`targetpath(`*`movieClipObject:MovieClip`*`) : String`

Parameters

`movieClipObject` Reference (for example, `_root` or `_parent`) to the movie clip for which the target path is being retrieved.

Returns

A string containing the target path of the specified movie clip.

Description

Function; returns a string containing the target path of *`movieClipObject`*. The target path is returned in dot (.) notation. To retrieve the target path in slash (/) notation, use the `_target` property.

Example

The following example traces the target path of a movie clip as soon as it loads:

```
this.createEmptyMovieClip("myClip_mc", this.getNextHighestDepth());
trace(targetPath(myClip_mc));//traces _level0.myClip_mc
```

See also

`eval()`

TextField.StyleSheet class

Availability

Flash Player 7.

Description

The TextField.StyleSheet class lets you create a style sheet object that contains text formatting rules such as font size, color, and other formatting styles. You can then apply styles defined by a style sheet to a TextField object that contains HTML- or XML-formatted text. The text contained by the TextField object is then automatically formatted according to the tag styles defined by the style sheet object. You can use text styles to define new formatting tags, redefine built-in HTML tags, or create style classes that can be applied to certain HTML tags.

To apply styles to a TextField object, assign the style sheet object to a TextField object's `styleSheet` property.

For more information, see "Formatting text with Cascading Style Sheets" in *Using ActionScript in Flash*.

Method summary for the TextField.StyleSheet class

Method	Description
`TextField.StyleSheet.clear()`	Removes all styles from the style sheet object.
`TextField.StyleSheet.getStyle()`	Returns a copy of the style sheet object associated with a specified style name.
`TextField.StyleSheet.getStyleNames()`	Returns an array that contains the names of all of the styles registered in the style sheet object.
`TextField.StyleSheet.load()`	Begins loading a CSS file into the style sheet object.
`TextField.StyleSheet.parseCSS()`	Parses a string of CSS text and creates the specified style.
`TextField.StyleSheet.setStyle()`	Adds a new style to the style sheet object.
`TextField.StyleSheet.transform()`	Extends the CSS parsing capability.

Event handler summary for the TextField.StyleSheet class

Method	Description
`TextField.StyleSheet.onLoad`	Callback handler invoked when a TextField.StyleSheet.load() operation has completed.

Constructor for the TextField.StyleSheet class

Availability

Flash Player 7.

Usage

```
new TextField.StyleSheet() : TextField.StyleSheet
```

Returns

A reference to a TextField.StyleSheet object.

Description

Constructor; creates a TextField.StyleSheet object.

Example

The following example loads in a style sheet and outputs the styles that load into the document. Add the following ActionScript to your AS or FLA file:

```
var my_styleSheet:TextField.StyleSheet = new TextField.StyleSheet();
my_styleSheet.onLoad = function(success:Boolean) {
  if (success) {
    trace("Styles loaded:");
    var styles_array:Array = my_styleSheet.getStyleNames();
    trace(styles_array.join(newline));
  } else {
    trace("Error loading CSS");
  }
};
my_styleSheet.load("styles.css");
```

The styles.css file contains two styles called .heading and .mainbody, so the following information is displayed in the Output panel:

```
Styles loaded:
.heading
.mainBody
```

The complete code for styles.css is found in the example for TextField.StyleSheet.getStyle().

TextField.StyleSheet.clear()

Availability

Flash Player 7.

Usage

```
styleSheet.clear() : Void
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all styles from the specified style sheet object.

Example

The following example loads a style sheet called styles.css into a SWF file, and displays the styles that are loaded in the Output panel. When you click `clear_btn`, all styles from the `my_styleSheet` object are removed.

```
// Create a new style sheet object
var my_styleSheet:TextField.StyleSheet = new TextField.StyleSheet();

my_styleSheet.onLoad = function(success:Boolean) {
  if (success) {
    trace("Styles loaded.");
    var styles_array:Array = my_styleSheet.getStyleNames();
    for (var i = 0; i<styles_array.length; i++) {
      trace("\t"+styles_array[i]);
    }
    trace("");
  } else {
    trace("Error loading CSS");
  }
};

// Start the loading operation
my_styleSheet.load("styles.css");

clear_btn.onRelease = function() {
  my_styleSheet.clear();
  trace("Styles cleared.");
  var styles_array:Array = my_styleSheet.getStyleNames();
  for (var i = 0; i<styles_array.length; i++) {
    trace("\t"+styles_array[i]);
  }
  trace("");
};
```

TextField.StyleSheet.getStyle()

Availability

Flash Player 7.

Usage

```
styleSheet.getStyle(styleName:String) : Object
```

Parameters

`styleName` A string that specifies the name of the style to retrieve.

Returns

An object.

Description

Method; returns a copy of the style object associated with the style named `styleName`. If there is no style object associated with `styleName`, `null` is returned.

Example

The following example loads styles from a CSS file, parses the stylesheet and displays style names and property values in the Output panel. Create a new ActionScript file called StyleSheetTracer.as and enter the following code:

```
import TextField.StyleSheet;
class StyleSheetTracer {
  // StyleSheetTracer.displayFromURL
  //
  // This method displays the CSS style sheet at
  // URL "url" to the Output Panel.
  static function displayFromURL(url:String):Void {
    // Create a new style sheet object
    var my_styleSheet:StyleSheet = new StyleSheet();
    // The load operation is asynchronous, so set up
    // a callback function to display the loaded style sheet.
    my_styleSheet.onLoad = function(success:Boolean) {
      if (success) {
        StyleSheetTracer.display(this);
      } else {
        trace("Error loading style sheet "+url);
      }
    };
    // Start the loading operation.
    my_styleSheet.load(url);
  }
  static function display(my_styleSheet:StyleSheet):Void {
    var styleNames:Array = my_styleSheet.getStyleNames();
    if (!styleNames.length) {
      trace("This is an empty style sheet.");
    } else {
      for (var i = 0; i<styleNames.length; i++) {
        var styleName:String = styleNames[i];
        trace("Style "+styleName+":");
```

```
          var styleObject:Object = my_styleSheet.getStyle(styleName);
          for (var propName in styleObject) {
            var propValue = styleObject[propName];
            trace("\t"+propName+": "+propValue);
          }
          trace("");
        }
      }
    }
  }
```

Create a new CSS document called styles.css, which has two styles called `heading` and `mainBody` that define properties for `font-family`, `font-size` and `font-weight`. Enter the following code:

```
/* In styles.css */
.heading {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 24px;
  font-weight: bold;
}
.mainBody {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 12px;
  font-weight: normal;
}
```

And finally, in a FLA or AS file, enter the following ActionScript to load the external style sheet, styles.css.

```
StyleSheetTracer.displayFromURL("styles.css");
```

This displays the following in the Output panel:`Style .heading:`

```
  fontWeight: bold
  fontSize: 24px
  fontFamily: Arial, Helvetica, sans-serif

Style .mainBody:
  fontWeight: normal
  fontSize: 12px
  fontFamily: Arial, Helvetica, sans-serif
```

See also

`TextField.StyleSheet.setStyle()`

TextField.StyleSheet.getStyleNames()

Availability

Flash Player 7.

Usage

```
styleSheet.getStyleNames() : Array
```

Parameters

None.

Returns

An array.

Description

Method; returns an array that contains the names (as strings) of all of the styles registered in this style sheet.

Example

This example creates a style sheet object named `styleSheet` that contains two styles, `heading` and `bodyText`. It then invokes the style sheet object's `getStyleNames()` method, assigns the results to the array `names_array`, and displays the contents of the array in the Output panel.

```
var my_styleSheet:TextField.StyleSheet = new TextField.StyleSheet();
my_styleSheet.setStyle("heading", {fontsize:'24px'});
my_styleSheet.setStyle("bodyText", {fontsize:'12px'});
var names_array:Array = my_styleSheet.getStyleNames();
trace(names_array.join("\n"));
```

The following is displayed in the Output panel:

```
bodyText
heading
```

See also

`TextField.StyleSheet.getStyle()`

TextField.StyleSheet.load()

Availability

Flash Player 7.

Usage

styleSheet.load(*url:String*) : *Void*

Parameters

url The URL of a CSS file to load. The URL must be in the same domain as the URL where the SWF file currently resides.

Returns

Nothing.

Description

Method; starts loading the CSS file into *styleSheet*. The load operation is asynchronous; use the TextField.StyleSheet.onLoad callback handler to determine when the file has finished loading.

The CSS file must reside in exactly the same domain as the SWF file that is loading it. For more information about restrictions on loading data across domains, see "Flash Player security features" in *Using ActionScript in Flash*.

Example

For an example of asynchronously loading style sheets using ActionScript 2.0, see the entry for TextField.StyleSheet.getStyle().

The following example loads the CSS file named styles.css into the style sheet object `styleObj`. When the file has finished loading successfully, the style sheet object is applied to a TextField object named `news_txt`.

```
this.createTextField("news_txt", 999, 10, 10, 320, 240);
news_txt.multiline = true;
news_txt.wordWrap = true;
news_txt.html = true;

var my_styleSheet:TextField.StyleSheet = new TextField.StyleSheet();
my_styleSheet.onLoad = function(success:Boolean) {
  if (success) {
    news_txt.styleSheet = my_styleSheet;
    news_txt.htmlText = "<p class=\"heading\">Heading goes here!</p><p
  class=\"mainBody\">Lorem ipsum dolor sit amet, consectetuer adipiscing elit,
  sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat
  volutpat.</p>";
  }
};
my_styleSheet.load("styles.css");
```

For the code contained in styles.css, see the entry for TextField.StyleSheet.getStyle().

See also

`TextField.StyleSheet.onLoad`

TextField.StyleSheet.onLoad

Availability

Flash Player 7.

Usage

styleSheet.onLoad = function (*success*:*Boolean*) {}

Parameters

success	A Boolean value indicating whether the CSS file was successfully loaded.

Returns

Nothing.

Description

Callback handler; invoked when a TextField.StyleSheet.load() operation has completed. If the style sheet loaded successfully, the *success* parameter is `true`. If the document was not received, or if an error occurred in receiving the response from the server, the *success* parameter is `false`.

Example

The following example loads the CSS file named styles.css into the style sheet object `styleObj`. When the file has finished loading successfully, the style sheet object is applied to a TextField object named `news_txt`.

```
this.createTextField("news_txt", 999, 10, 10, 320, 240);
news_txt.multiline = true;
news_txt.wordWrap = true;
news_txt.html = true;

var my_styleSheet:TextField.StyleSheet = new TextField.StyleSheet();
my_styleSheet.onLoad = function(success:Boolean) {
  if (success) {
    news_txt.styleSheet = my_styleSheet;
    news_txt.htmlText = "<p class=\"heading\">Heading goes here!</p><p
  class=\"mainBody\">Lorem ipsum dolor sit amet, consectetuer adipiscing elit,
  sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat
  volutpat.</p>";
  }
};
my_styleSheet.load("styles.css");
```

For the code contained in styles.css, see the entry for TextField.StyleSheet.getStyle(). For an example of asynchronously loading style sheets using ActionScript 2.0, see the entry for TextField.StyleSheet.getStyle().

See also

`TextField.StyleSheet.load()`

TextField.StyleSheet.parseCSS()

Availability

Flash Player 7.

Usage

`styleSheet``.parseCSS(`*`cssText:String`*`) : `*`Boolean`*

Parameters

`cssText` The CSS text to parse (a string).

Returns

A Boolean value indicating if the text was parsed successfully (`true`) or not (`false`).

Description

Method; parses the CSS in *`cssText`* and loads the style sheet with it. If a style in *`cssText`* is already in *`styleSheet`*, the properties in *`styleSheet`* are retained, and only the ones in *`cssText`* are added or changed in *`styleSheet`*.

To extend the native CSS parsing capability, you can override this method by creating a subclass of the TextField.StyleSheet class. For more information, see "Creating subclasses" in *Using ActionScript in Flash*.

Example

The following example parses the CSS in `css_str`. The ActionScript displays information about whether it parsed successfully, and then displays the parsed CSS in the Output panel. Add the following ActionScript to your AS or FLA file:

```
var css_str:String = ".heading {font-family: Arial, Helvetica, sans-serif;
  font-size: 24px; font-weight: bold; }";
var my_styleSheet:TextField.StyleSheet = new TextField.StyleSheet();
if (my_styleSheet.parseCSS(css_str)) {
  trace("parsed successfully");
  dumpStyles(my_styleSheet);
} else {
  trace("unable to parse CSS");
}
//
function dumpStyles(styles:TextField.StyleSheet):Void {
  var styleNames_array:Array = styles.getStyleNames();
  for (var i = 0; i<styleNames_array.length; i++) {
    var styleName_str:String = styleNames_array[i];
    var styleObject:Object = styles.getStyle(styleName_str);
    trace(styleName_str);
    for (var prop in styleObject) {
      trace("\t"+prop+": "+styleObject[prop]);
    }
    trace("");
  }
}
```

TextField.StyleSheet.setStyle()

Availability

Flash Player 7.

Usage

styleSheet.setStyle(*name:String*, *style:Object*) *: Void*

Parameters

name A string that specifies the name of the style to add to the style sheet.

style An object that describes the style, or `null`.

Returns

Nothing.

Description

Method; adds a new style with the specified name to the style sheet object. If the named style does not already exist in the style sheet, it is added. If the named style already exists in the style sheet, it is replaced. If the *style* parameter is `null`, the named style is removed.

Flash Player creates a copy of the style object that you pass to this method.

Example

The following code adds a style named `emphasized` to the style sheet `myStyleSheet`. The style includes two style properties: `color` and `fontWeight`. The style object is defined with the `{}` operator.

```
myStyleSheet.setStyle("emphasized", {color:'#000000',fontWeight:'bold'});
```

You could also create a style object using an instance of the Object class, and then pass that object as the *style* parameter, as the next example shows.

```
var my_styleSheet:TextField.StyleSheet = new TextField.StyleSheet();

var styleObj:Object = new Object();
styleObj.color = "#000000";
styleObj.fontWeight = "bold";
my_styleSheet.setStyle("emphasized", styleObj);
delete styleObj;

var styleNames_array:Array = my_styleSheet.getStyleNames();
for (var i=0;i<styleNames_array.length;i++) {
  var styleName:String = styleNames_array[i];
  var thisStyle:Object = my_styleSheet.getStyle(styleName);
  trace(styleName);
  for (var prop in thisStyle) {
    trace("\t"+prop+": "+thisStyle[prop]);
  }
  trace("");
}
```

This displays the following information in the Output panel:

```
emphasized
  fontWeight: bold
  color: #000000
```

Note: The line of code `delete styleObj` deletes the original style object passed to `setStyle()`. While not necessary, this step reduces memory usage, because Flash Player creates a copy of the style object you pass to `setStyle()`.

See also

{} (object initializer), “Formatting text with Cascading Style Sheets” in *Using ActionScript in Flash*

TextField.StyleSheet.transform()

Availability

Flash Player 7.

Usage

styleSheet.transform(*style:Object*) : *TextFormat*

Parameters

style An object that describes the style, containing style rules as properties of the object, or null.

Returns

A TextFormat object containing the result of the mapping of CSS rules to text format properties.

Description

Method; extends the CSS parsing capability.

Advanced developers can override this method by extending the TextField.StyleSheet class. For more information, see “Creating subclasses” in *Using ActionScript in Flash*.

Example

The following code subclasses this method:

```
class advCSS extends TextField.StyleSheet {
  // override the transform method
  function transform(style:Object):TextFormat {
    for (var z in style) {
      if (z == "margin") {
        style.marginLeft = style[z];
        style.marginRight = style[z];
        delete style[z];
        break;
      }
    }
    return super.transform(style);
  }
}
// end class definition
```

TextField class

Availability

Flash Player 6.

Description

All dynamic and input text fields in a SWF file are instances of the TextField class. You can give a text field an instance name in the Property inspector and use the methods and properties of the TextField class to manipulate it with ActionScript. TextField instance names are displayed in the Movie Explorer and in the Insert Target Path dialog box in the Actions panel.

The TextField class inherits from the Object class.

To create a text field dynamically, you do not use the `new` operator. Instead, you use MovieClip.createTextField().

Method summary for the TextField class

Method	Description
`TextField.addListener()`	Registers an object to receive notification when the `onChanged` and `onScroller` event handlers are invoked.
`TextField.getFontList()`	Returns names of fonts on the player's host system as an array.
`TextField.getDepth()`	Returns the depth of a text field.
`TextField.getNewTextFormat()`	Gets the default text format assigned to newly inserted text.
`TextField.getTextFormat()`	Returns a TextFormat object containing formatting information for some or all text in a text field.
`TextField.removeListener()`	Removes a listener object.
`TextField.removeTextField()`	Removes a text field that was created with MovieClip.createTextField().
`TextField.replaceSel()`	Replaces the current selection.
`TextField.replaceText()`	Replaces a range of characters.
`TextField.setNewTextFormat()`	Sets a TextFormat object for text that is inserted by a user or by a method.
`TextField.setTextFormat()`	Sets a TextFormat object for a specified range of text in a text field.

Property summary for the TextField class

Property	Description
`TextField._alpha`	The transparency value of a text field instance.
`TextField.autoSize`	Controls automatic alignment and sizing of a text field.
`TextField.background`	Indicates if the text field has a background fill.
`TextField.backgroundColor`	Indicates the color of the background fill.

Property	Description
`TextField.border`	Indicates if the text field has a border.
`TextField.borderColor`	Indicates the color of the border.
`TextField.bottomScroll`	Read-only; the bottommost visible line in a text field.
`TextField.embedFonts`	Indicates whether the text field uses embedded font outlines or device fonts.
`TextField._height`	The height of a text field instance in pixels. This affects only the bounding box of the text field; it does not affect the border thickness or text font size.
`TextField.hscroll`	Indicates the horizontal scroll value of a text field.
`TextField.html`	A flag that indicates whether the text field contains an HTML representation.
`TextField.htmlText`	Contains the HTML representation of a text field's contents.
`TextField.length`	Read-only; the number of characters in a text field.
`TextField.maxChars`	The maximum number of characters that a text field can contain.
`TextField.maxhscroll`	Read-only; the maximum value of `TextField.hscroll`.
`TextField.maxscroll`	Read-only; the maximum value of `TextField.scroll`.
`TextField.menu`	Associates a ContextMenu object with a text field.
`TextField.mouseWheelEnabled`	Indicates whether Flash Player should automatically scroll multiline text fields when the mouse pointer is positioned over a text field and the user rolls the mouse wheel.
`TextField.multiline`	Indicates if the text field contains multiple lines.
`TextField._name`	The instance name of a text field instance.
`TextField._parent`	A reference to the instance that is the parent of this instance; either of type Button or MovieClip.
`TextField.password`	Indicates if a text field hides the input characters.
`TextField._quality`	Indicates the rendering quality of a SWF file.
`TextField.restrict`	The set of characters that a user can enter into a text field.
`TextField._rotation`	The degree of rotation of a text field instance.
`TextField.scroll`	Indicates the current scrolling position of a text field.
`TextField.selectable`	Indicates whether a text field is selectable.
`TextField.tabEnabled`	Indicates whether a movie clip is included in automatic tab ordering.
`TextField.tabIndex`	Indicates the tab order of an object.
`TextField._target`	Read-only; the target path of the specified text field instance.
`TextField.text`	The current text in the text field.
`TextField.textColor`	The color of the current text in the text field.

Property	Description
`TextField.textHeight`	The height of the text field's bounding box.
`TextField.textWidth`	The width of the text field's bounding box.
`TextField.type`	Indicates whether a text field is an input text field or dynamic text field.
`TextField._url`	Read-only; the URL of the SWF file that created the text field instance.
`TextField.variable`	The variable name associated with the text field.
`TextField._visible`	A Boolean value that determines whether a text field instance is hidden or visible.
`TextField._width`	The width of a text field instance in pixels. This only affects the bounding box of the text field, it does not affect the border thickness or text font size.
`TextField.wordWrap`	Indicates whether the text field word-wraps.
`TextField._x`	The *x* coordinate of a text field instance
`TextField._xmouse`	Read-only; the *x* coordinate of the pointer relative to a text field instance.
`TextField._xscale`	The value specifying the percentage for horizontally scaling a text field instance.
`TextField._y`	The *y* coordinate of a text field instance.
`TextField._ymouse`	Read-only; the *y* coordinate of the pointer relative to a text field instance.
`TextField._yscale`	The value specifying the percentage for vertically scaling a text field instance.

Event handler summary for the TextField class

Event handler	Description
`TextField.onChanged`	Invoked when the text field is changed.
`TextField.onKillFocus`	Invoked when the text field loses focus.
`TextField.onScroller`	Invoked when one of the text field scroll properties changes.
`TextField.onSetFocus`	Invoked when the text field receives focus.

Listener summary for the TextField class

Method	Description
`TextField.onChanged`	Notified when the text field is changed.
`TextField.onScroller`	Notified when the `scroll` or `maxscroll` property of a text field changes.

TextField.addListener()

Availability

Flash Player 6.

Usage

`my_txt.addListener(listener:Object) : Void`

Parameters

listener An object with an `onChanged` or `onScroller` event handler.

Returns

Nothing.

Description

Method; registers an object to receive notification when the `onChanged` and `onScroller` event handlers have been invoked. When a text field changes or is scrolled, the `TextField.onChanged` and `TextField.onScroller` event handlers are invoked, followed by the `onChanged` and `onScroller` event handlers of any objects registered as listeners. Multiple objects can be registered as listeners.

To remove a listener object from a text field, call `TextField.removeListener()`.

A reference to the text field instance is passed as a parameter to the `onScroller` and `onChanged` handlers by the event source. You can capture this data by putting a parameter in the event handler method. For example, the following code uses `txt` as the parameter that is passed to the `onScroller` event handler. The parameter is then used in a `trace` statement to send the instance name of the text field to the Output panel.

```
my_txt.onScroller = function(textfield_txt:TextField) {
  trace(textfield_txt._name+" scrolled");
};
```

Example

The following example defines an `onChanged` handler for the input text field `my_txt`. It then defines a new listener object, `txtListener`, and defines an `onChanged` handler for that object. This handler will be invoked when the text field `my_txt` is changed. The final line of code calls `TextField.addListener` to register the listener object `txtListener` with the text field `my_txt` so that it will be notified when `my_txt` changes.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
my_txt.border = true;
my_txt.type = "input";

my_txt.onChanged = function(textfield_txt:TextField) {
  trace(textfield_txt._name+" changed");
};
var txtListener:Object = new Object();
txtListener.onChanged = function(textfield_txt:TextField) {
  trace(textfield_txt._name+" changed and notified myListener");
};
my_txt.addListener(txtListener);
```

See also

`TextField.onChanged`, `TextField.onScroller`, `TextField.removeListener()`

TextField._alpha

Availability

Flash Player 6.

Usage

my_txt._alpha:*Number*

Description

Property; sets or retrieves the alpha transparency value of the text field specified by *my_txt*. Valid values are 0 (fully transparent) to 100 (fully opaque). The default value is 100. Transparency values are not supported for text files that use device fonts. You must use embedded fonts to use the `_alpha` transparency property with a text field.

Example

The following code sets the `_alpha` property of a text field named `my_txt` to 20%. Create a new font symbol in the library by selecting New Font from the Library options menu. Then set the linkage of the font to `my font`.Add the following ActionScript to your FLA or AS file:

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.font = "my font";
// where 'my font' is the linkage name of a font in the Library
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
my_txt.border = true;
my_txt.embedFonts = true;
my_txt.text = "Hello World";
my_txt.setTextFormat(my_fmt);
my_txt._alpha = 20;
```

See also

`Button._alpha`, `MovieClip._alpha`

TextField.autoSize

Availability

Flash Player 6.

Usage

```
my_txt.autoSize:String
my_txt.autoSize:Boolean
```

Description

Property; controls automatic sizing and alignment of text fields. Acceptable values for autoSize are `"none"` (the default), `"left"`, `"right"`, and `"center"`. When you set the `autoSize` property, `true` is a synonym for `"left"` and `false` is a synonym for `"none"`.

The values of `autoSize` and TextField.wordWrap determine whether a text field expands or contracts to the left side, right side, or bottom side. The default value for each of these properties is `false`.

If `autoSize` is set to `"none"` (the default) or `false`, then no resizing will occur.

If `autoSize` is set to `"left"` or `true`, then the text is treated as left-justified text, meaning the left side of the text field will remain fixed and any resizing of a single line text field will be on the right side. If the text includes a line break (for example, `"\n" or "\r"`), then the bottom side will also be resized to fit the next line of text. If `wordWrap` is also set to `true`, then only the bottom side of the text field will be resized and the right side will remain fixed.

If `autoSize` is set to `"right"`, then the text is treated as right-justified text, meaning the right side of the text field will remain fixed and any resizing of a single line text field will be on the left side. If the text includes a line break (for example, `"\n" or "\r"`), then the bottom side will also be resized to fit the next line of text. If `wordWrap` is also set to `true`, then only the bottom side of the text field will be resized and the left side will remain fixed.

If `autoSize` is set to `"center"`, then the text is treated as center-justified text, meaning any resizing of a single line text field will be equally distributed to both the right and left sides. If the text includes a line break (for example, `"\n" or "\r"`), then the bottom side will also be resized to fit the next line of text. If `wordWrap` is also set to `true`, then only the bottom side of the text field will be resized and the left and right sides will remain fixed.

Example

You can use the following code and enter different values for `autoSize` to see how the field resizes when these values change. A mouse click while the SWF file is playing will replace each text field's `"short text"` string with longer text using several different settings for `autoSize`.

```
this.createTextField("left_txt", 997, 10, 10, 70, 30);
this.createTextField("center_txt", 998, 10, 50, 70, 30);
this.createTextField("right_txt", 999, 10, 100, 70, 30);
this.createTextField("true_txt", 1000, 10, 150, 70, 30);
this.createTextField("false_txt", 1001, 10, 200, 70, 30);

left_txt.text = "short text";
left_txt.border = true;
```

```
center_txt.text = "short text";
center_txt.border = true;

right_txt.text = "short text";
right_txt.border = true;

true_txt.text = "short text";
true_txt.border = true;

false_txt.text = "short text";
false_txt.border = true;

// create a mouse listener object to detect mouse clicks
var myMouseListener:Object = new Object();
// define a function that executes when a user clicks the mouse
myMouseListener.onMouseDown = function() {
  left_txt.autoSize = "left";
  left_txt.text = "This is much longer text";
  center_txt.autoSize = "center";
  center_txt.text = "This is much longer text";
  right_txt.autoSize = "right";
  right_txt.text = "This is much longer text";
  true_txt.autoSize = true;
  true_txt.text = "This is much longer text";
  false_txt.autoSize = false;
  false_txt.text = "This is much longer text";
};
// register the listener object with the Mouse object
Mouse.addListener(myMouseListener);
```

TextField.background

Availability

Flash Player 6.

Usage

`my_txt.background:Boolean`

Description

Property; if `true`, the text field has a background fill. If `false`, the text field has no background fill.

Example

The following example creates a text field with a button that toggles the background color of the field.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320, 240);
my_txt.border = true;
my_txt.text = "Lorum ipsum";
my_txt.backgroundColor = 0xFF0000;

toggle_btn.onRelease = function() {
  my_txt.background = !my_txt.background;
};
```

TextField.backgroundColor

Availability

Flash Player 6.

Usage

```
my_txt.backgroundColor:Number
```

Description

Property; the color of the text field background. Default is `0xFFFFFF` (white). This property may be retrieved or set, even if there currently is no background, but the color is only visible if the text field has a border.

Example

See the example for TextField.background.

See also

`TextField.background`

TextField.border

Availability

Flash Player 6.

Usage

my_txt.border:*Boolean*

Description

Property; if `true`, the text field has a border. If `false`, the text field has no border.

Example

The following example creates a text field called `my_txt`, sets the border property to `true`, and displays some text in the field.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320, 240);
my_txt.border = true;
my_txt.text = "Lorum ipsum";
```

TextField.borderColor

Availability

Flash Player 6.

Usage

my_txt.borderColor:*Number*

Description

Property; the color of the text field border, the Default is `0x000000` (black). This property may be retrieved or set, even if there is currently no border.

Example

The following example creates a text field called `my_txt`, sets the border property to `true`, and displays some text in the field.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320, 240);
my_txt.border = true;
my_txt.borderColor = 0x00FF00;
my_txt.text = "Lorum ipsum";
```

See also

`TextField.border`

TextField.bottomScroll

Availability

Flash Player 6.

Usage

my_txt.bottomScroll:*Number*

Description

Read-only property; an integer (one-based index) that indicates the bottommost line that is currently visible in *my_txt*. Think of the text field as a "window" onto a block of text. The property TextField.scroll is the one-based index of the topmost visible line in the window.

All the text between lines TextField.scroll and TextField.bottomScroll is currently visible in the text field.

Example

The following example creates a text field and fills it with text. The scroll and bottomScroll properties for the text field are then traced for the comment_txt field.

```
this.createTextField("comment_txt", this.getNextHighestDepth(), 0, 0, 160, 
  120);
comment_txt.html = true;
comment_txt.selectable = true;
comment_txt.multiline = true;
comment_txt.wordWrap = true;
comment_txt.htmlText = "<b>What is hexadecimal?</b><br>The hexadecimal color 
  system uses six digits to represent color values. Each digit has sixteen 
  possible values or characters. The characters range from 0 to 9 and then A to 
  F. Black is represented by (#000000) and white, at the opposite end of the 
  color system, is (#FFFFFF).";
my_btn.onRelease = function() {
  trace("scroll: "+comment_txt.scroll);
  trace("bottomScroll: "+comment_txt.bottomScroll);
};
```

TextField.condenseWhite

Availability

Flash Player 6.

Usage

my_txt.condenseWhite:*Boolean*

Description

Property; a Boolean value that specifies whether extra white space (spaces, line breaks, and so on) in an HTML text field should be removed when the field is rendered in a browser. The default value is `false`.

If you set this value to `true`, you must use standard HTML commands such as `<BR>` and `<P>` to place line breaks in the text field.

If *my_txt*.html is `false`, this property is ignored.

Example

The following example creates two text fields, called `first_txt` and `second_txt`. The white space is removed from the second text field. Add the following ActionScript to your FLA or AS file:

```
var my_str:String = "Hello\tWorld\nHow are you?\t\t\tEnd";

this.createTextField("first_txt", this.getNextHighestDepth(), 10, 10, 160, 
  120);
first_txt.html = true;
first_txt.multiline = true;
first_txt.wordWrap = true;
first_txt.condenseWhite = false;
first_txt.border = true;
first_txt.htmlText = my_str;

this.createTextField("second_txt", this.getNextHighestDepth(), 180, 10, 160, 
  120);
second_txt.html = true;
second_txt.multiline = true;
second_txt.wordWrap = true;
second_txt.condenseWhite = true;
second_txt.border = true;
second_txt.htmlText = my_str;
```

See also

TextField.html

TextField.embedFonts

Availability

Flash Player 6.

Usage

my_txt.embedFonts:*Boolean*

Description

Property; a Boolean value that, when `true`, renders the text field using embedded font outlines. If `false`, it renders the text field using device fonts.

Example

In this example, you need to create a dynamic text field called `my_txt`, and then use the following ActionScript to embed fonts and rotate the text field. The reference to `my font` refers to a Font symbol in the library, with linkage set to `my font`.

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.font = "my font";

this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 120);
my_txt.wordWrap = true;
my_txt.embedFonts = true;
my_txt.text = "Hello world";
my_txt.setTextFormat(my_fmt);
my_txt._rotation = 45;
```

TextField.getDepth()

Availability

Flash Player 6.

Usage

```
my_txt.getDepth() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the depth of a text field.

Example

The following example demonstrates text fields residing at different depths. Create a dynamic text field on the Stage. Add the following ActionScript to your FLA or AS file, which dynamically creates two text fields at runtime and outputs their depths.

```
this.createTextField("first_mc", this.getNextHighestDepth(), 10, 10, 100, 22);
this.createTextField("second_mc", this.getNextHighestDepth(), 10, 10, 100,
  22);
for (var prop in this) {
  if (this[prop] instanceof TextField) {
    var this_txt:TextField = this[prop];
    trace(this_txt._name+" is a TextField at depth: "+this_txt.getDepth());
  }
}
```

TextField.getFontList()

Availability

Flash Player 6.

Usage

```
TextField.getFontList() : Array
```

Parameters

None.

Returns

An array.

Description

Method; a static method of the global TextField class. You don't specify a specific text field (such as `my_txt`) when you call this method. This method returns names of fonts on the player's host system as an array. (It does not return names of all fonts in currently loaded SWF files.) The names are of type String.

Example

The following code displays a font list returned by `getFontList()`:

```
var font_array:Array = TextField.getFontList();
font_array.sort();
trace("You have "+font_array.length+" fonts currently installed");
trace("--------------------------------------");
for (var i = 0; i<font_array.length; i++) {
  trace("Font #"+(i+1)+":\t"+font_array[i]);
}
```

TextField.getNewTextFormat()

Availability

Flash Player 6.

Usage

my_txt.getNewTextFormat() *: TextFormat*

Parameters

None.

Returns

A TextFormat object.

Description

Method; returns a TextFormat object containing a copy of the text field's text format object. The text format object is the format that newly inserted text, such as text inserted with the `replaceSel()` method or text entered by a user, receives. When `getNewTextFormat()` is invoked, the TextFormat object returned has all of its properties defined. No property is `null`.

Example

The following example displays the specified text field's (`my_txt`) text format object.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 120);
var my_fmt:TextFormat = my_txt.getNewTextFormat();
trace("TextFormat has the following properties:");
for (var prop in my_fmt) {
  trace(prop+": "+my_fmt[prop]);
}
```

TextField.getTextFormat()

Availability

Flash Player 6.

Usage

my_txt.getTextFormat() *: Object*

my_txt.getTextFormat(*index:Number*) *: Object*

my_txt.getTextFormat(*beginIndex:Number*, *endIndex:Number*) *: Object*

Parameters

index An integer that specifies a character in a string.

beginIndex, endIndex Integers that specify the starting and ending locations of a span of text within *my_txt*.

Returns

An object.

Description

Method; returns a TextFormat object containing a particular kind of information.

Usage 1: returns a TextFormat object containing formatting information for all text in a text field. Only properties that are common to all text in the text field are set in the resulting TextFormat object. Any property which is *mixed*, meaning that it has different values at different points in the text, has a value of `null`.

Usage 2: Returns a TextFormat object containing a copy of the text field's text format at *index*.

Usage 3: Returns a TextFormat object containing formatting information for the span of text from *beginIndex* to *endIndex*. Only properties that are common to all of the text in the specified range is set in the resulting TextFormat object. Any property that is mixed (it has different values at different points in the range) has its value set to null.

Example

The following ActionScript traces all of the formatting information for a text field that is created at runtime.

```
this.createTextField("dyn_txt", this.getNextHighestDepth(), 0, 0, 100, 200);
dyn_txt.text = "Frank";
dyn_txt.setTextFormat(new TextFormat());
var my_fmt:TextFormat = dyn_txt.getTextFormat();
for (var prop in my_fmt) {
  trace(prop+": "+my_fmt[prop]);
}
```

See also

`TextField.getNewTextFormat()`, `TextField.setNewTextFormat()`,
  `TextField.setTextFormat()`

TextField._height

Availability

Flash Player 6.

Usage

```
my_txt._height:Number
```

Description

Property; the height of the text field, in pixels.

Example

The following code example sets the height and width of a text field:

```
my_txt._width = 200;
my_txt._height = 200;
```

TextField.hscroll

Availability

Flash Player 6.

Usage

`my_txt.hscroll:Number`

Returns

An integer.

Description

Property; indicates the current horizontal scrolling position. If the `hscroll` property is 0, the text is not horizontally scrolled.

For more information on scrolling text, see "Creating scrolling text" in *Using ActionScript in Flash*.

The units of horizontal scrolling are pixels, while the units of vertical scrolling are lines. Horizontal scrolling is measured in pixels because most fonts you typically use are proportionally spaced; meaning, the characters can have different widths. Flash performs vertical scrolling by line because users usually want to see a line of text in its entirety, as opposed to seeing a partial line. Even if there are multiple fonts on a line, the height of the line adjusts to fit the largest font in use.

Note: The hscroll property is zero-based—not one-based like the vertical scrolling property `TextField.scroll`.

Example

The following example scrolls the `my_txt` text field horizontally using two buttons called `scrollLeft_btn` and `scrollRight_btn`. The amount of scroll displays in a text field called `scroll_txt`. Add the following ActionScript to your FLA or AS file:

```
this.createTextField("scroll_txt", this.getNextHighestDepth(), 10, 10, 160,
  20);
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 30, 160, 22);
my_txt.border = true;
my_txt.multiline = false;
my_txt.wordWrap = false;
my_txt.text = "Lorem ipsum dolor sit amet, consectetuer adipiscing...";

scrollLeft_btn.onRelease = function() {
  my_txt.hscroll -= 10;
  scroll_txt.text = my_txt.hscroll+" of "+my_txt.maxhscroll;
};
scrollRight_btn.onRelease = function() {
  my_txt.hscroll += 10;
  scroll_txt.text = my_txt.hscroll+" of "+my_txt.maxhscroll;
};
```

See also

`TextField.maxhscroll`, `TextField.scroll`

TextField.html

Availability

Flash Player 6.

Usage

my_txt.html:*Boolean*

Description

Property; a flag that indicates whether the text field contains an HTML representation. If the `html` property is `true`, the text field is an HTML text field. If `html` is `false`, the text field is a non-HTML text field.

Example

The following example creates a text field that sets the `html` property to `true`. HTML-formatted text displays in the text field.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 22);
my_txt.html = true;
my_txt.htmlText = "<b> this is bold text </b>";
```

See also

`TextField.htmlText`

TextField.htmlText

Availability

Flash Player 6.

Usage

my_txt`.htmlText:`*String*

Description

Property; if the text field is an HTML text field, this property contains the HTML representation of the text field's contents. If the text field is not an HTML text field, it behaves identically to the `text` property. You can indicate that a text field is an HTML text field in the Property inspector, or by setting the text field's `html` property to `true`.

Example

The following example creates a text field that sets the `html` property to `true`. HTML-formatted text displays in the text field.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 22);
my_txt.html = true;
my_txt.htmlText = "<b> this is bold text </b>";
```

See also

`TextField.html`

TextField.length

Availability

Flash Player 6.

Usage

```
my_txt.length:Number
```

Returns

A number.

Description

Read-only property; indicates the number of characters in a text field. This property returns the same value as `text.length`, but is faster. A character such as tab (`\t`) counts as one character.

Example

The following example outputs the number of characters in the `date_txt` text field, which displays the current date.

```
var today:Date = new Date();
this.createTextField("date_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
date_txt.autoSize = true;
date_txt.text = today.toString();
trace(date_txt.length);
```

TextField.maxChars

Availability

Flash Player 6.

Usage

```
my_txt.maxChars:Number
```

Description

Property; indicates the maximum number of characters that the text field can contain. A script may insert more text than `maxChars` allows; the `maxChars` property indicates only how much text a user can enter. If the value of this property is `null`, there is no limit on the amount of text a user can enter.

Example

The following example creates a text field called `age_txt` that only lets users enter up to two numbers in the field.

```
this.createTextField("age_txt", this.getNextHighestDepth(), 10, 10, 30, 22);
age_txt.type = "input";
age_txt.border = true;
age_txt.maxChars = 2;
age_txt.restrict = "0-9";
```

TextField.maxhscroll

Availability

Flash Player 6.

Usage

```
my_txt.maxhscroll:Number
```

Description

Read-only property; indicates the maximum value of `TextField.hscroll`.

Example

See the example for `TextField.hscroll`.

TextField.maxscroll

Availability

Flash Player 6.

Usage

TextField.maxscroll:*Number*

Description

Read-only property; indicates the maximum value of TextField.scroll.

For more information on scrolling text, see "Creating scrolling text" in *Using ActionScript in Flash*.

Example

The following example sets the maximum value for the scrolling text field `my_txt`. Create two buttons, `scrollUp_btn` and `scrollDown_btn`, to scroll the text field. Add the following ActionScript to your FLA or AS file.

```
this.createTextField("scroll_txt", this.getNextHighestDepth(), 10, 10, 160, 
  20);
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 30, 320, 240);
my_txt.multiline = true;
my_txt.wordWrap = true;
for (var i = 0; i<10; i++) {
  my_txt.text += "Lorem ipsum dolor sit amet, consectetuer adipiscing elit, 
  sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat 
  volutpat.";
}
scrollUp_btn.onRelease = function() {
  my_txt.scroll--;
  scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
scrollDown_btn.onRelease = function() {
  my_txt.scroll++;
  scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
```

TextField.menu

Availability

Flash Player 7.

Usage

my_txt.menu = *contextMenu*

Parameters

contextMenu A ContextMenu object.

Description

Property; associates the ContextMenu object *contextMenu* with the text field *my_txt*. The ContextMenu class lets you modify the context menu that appears when the user right-clicks (Windows) or Control-clicks (Macintosh) in Flash Player.

This property works only with selectable (editable) text fields; it has no effect on nonselectable text fields.

Example

The following example assigns the ContextMenu object `menu_cm` to the text field `news_txt`. The ContextMenu object contains a custom menu item labeled "Resize" with an associated callback handler named `doResize()`, which could be used to add resizing functionality (not shown):

```
this.createTextField("news_txt", this.getNextHighestDepth(), 10, 10, 320,
  240);
news_txt.border = true;
news_txt.wordWrap = true;
news_txt.multiline = true;
news_txt.text = "To see the custom context menu item, right click (PC) or
  control click (Mac) within the text field.";
var menu_cm:ContextMenu = new ContextMenu();
menu_cm.customItems.push(new ContextMenuItem("Resize", doResize));
function doResize(obj:TextField, item:ContextMenuItem):Void {
  // "Resize" code here
  trace("you selected: "+item.caption);
}
news_txt.menu = menu_cm;
```

When you right-click or Control-click within the area of the text field, you see the custom menu item.

Caution: You cannot use a menu item that is already used by Flash. For example, Print... (with three dots) is reserved by Flash, so you cannot use this menu item; however, you could use Print.. (two dots) or any menu item not already used by Flash.

See also

Button.menu, ContextMenu class, ContextMenuItem class, MovieClip.menu

TextField.mouseWheelEnabled

Availability

Flash Player 7.

Usage

my_txt.mouseWheelEnabled:*Boolean*

Description

Property; a Boolean value that indicates whether Flash Player should automatically scroll multiline text fields when the mouse pointer clicks a text field and the user rolls the mouse wheel. By default, this value is `true`. This property is useful if you want to prevent mouse wheel scrolling of text fields, or implement your own text field scrolling.

Example

The following example creates two text fields. The scrollable_txt field has the `mouseWheelEnabled` property set to true, so `scrollable_txt` scrolls when you click the field and roll the mouse wheel. The `nonscrollable_txt` field does not scroll if you click the field and roll the mouse wheel.

```
var font_array:Array = TextField.getFontList().sort();

this.createTextField("scrollable_txt", this.getNextHighestDepth(), 10, 10,
  240, 320);
scrollable_txt.border = true;
scrollable_txt.wordWrap = true;
scrollable_txt.multiline = true;
scrollable_txt.text = font_array.join("\n");

this.createTextField("nonscrollable_txt", this.getNextHighestDepth(), 260, 10,
  240, 320);
nonscrollable_txt.border = true;
nonscrollable_txt.wordWrap = true;
nonscrollable_txt.multiline = true;
nonscrollable_txt.mouseWheelEnabled = false;
nonscrollable_txt.text = font_array.join("\n");
```

See also

`Mouse.onMouseWheel`

TextField.multiline

Availability

Flash Player 6.

Usage

my_txt.multiline:*Boolean*

Description

Property; indicates whether the text field is a multiline text field. If the value is `true`, the text field is multiline; if the value is `false`, the text field is a single-line text field.

Example

The following example creates a multiline text field called `fontList_txt` that displays a long, multiline list of fonts.

```
var font_array:Array = TextField.getFontList().sort();

this.createTextField("fontList_txt", this.getNextHighestDepth(), 10, 10, 240,
  320);
fontList_txt.border = true;
fontList_txt.wordWrap = true;
fontList_txt.multiline = true;
fontList_txt.text = font_array.join("\n");
```

TextField._name

Availability

Flash Player 6.

Usage

my_txt._name:*String*

Description

Property; the instance name of the text field specified by *my_txt*.

Example

The following example demonstrates text fields residing at different depths. Create a dynamic text field on the Stage. Add the following ActionScript to your FLA or AS file, which dynamically creates two text fields at runtime and displays their depths in the Output panel.

```
this.createTextField("first_mc", this.getNextHighestDepth(), 10, 10, 100, 22);
this.createTextField("second_mc", this.getNextHighestDepth(), 10, 10, 100,
  22);
for (var prop in this) {
  if (this[prop] instanceof TextField) {
    var this_txt:TextField = this[prop];
    trace(this_txt._name+" is a TextField at depth: "+this_txt.getDepth());
  }
}
```

When you test the document, the instance name and depth is displayed in the Output panel.

TextField.onChanged

Availability

Flash Player 6.

Usage

```
my_txt.onChanged = function(){
  // your statements here
}
myListenerObj.onChanged = function(){
  // your statements here
}
```

Parameters

None.

Returns

The instance name of the text field.

Description

Event handler/listener; invoked when the content of a text field changes. By default, it is undefined; you can define it in a script.

A reference to the text field instance is passed as a parameter to the `onChanged` handler. You can capture this data by putting a parameter in the event handler method. For example, the following code uses `textfield_txt` as the parameter that is passed to the `onChanged` event handler. The parameter is then used in a `trace()` statement to send the instance name of the text field to the Output panel:

```
this.createTextField("myInputText_txt", 99, 10, 10, 300, 20);
myInputText_txt.border = true;
myInputText_txt.type = "input";

myInputText_txt.onChanged = function(textfield_txt:TextField) {
  trace("the value of "+textfield_txt._name+" was changed. New value is:
  "+textfield_txt.text);
};
```

The `onChanged` handler is fired only when the change results from user interaction; for example, when the user is typing something on the keyboard, changing something in the text field using the mouse, or selecting a menu item. Programmatic changes to the text field do not trigger the `onChanged` event because the code recognizes changes that are made to the text field.

TextField.onKillFocus

Availability

Flash Player 6.

Usage

```
my_txt.onKillFocus = function(newFocus:Object){
  // your statements here
}
```

Parameters

newFocus	The object that is receiving the focus.

Returns

Nothing.

Description

Event handler; invoked when a text field loses keyboard focus. The `onKillFocus` method receives one parameter, *newFocus*, which is an object representing the new object receiving the focus. If no object receives the focus, *newFocus* contains the value `null`.

Example

The following example creates two text fields called `first_txt` and `second_txt`. When you give focus to a text field, information about the text field with current focus and the text field that lost focus is displayed in the Output panel.

```
this.createTextField("first_txt", 1, 10, 10, 300, 20);
first_txt.border = true;
first_txt.type = "input";
this.createTextField("second_txt", 2, 10, 40, 300, 20);
second_txt.border = true;
second_txt.type = "input";
first_txt.onKillFocus = function(newFocus:Object) {
  trace(this._name+" lost focus. New focus changed to: "+newFocus._name);
};
first_txt.onSetFocus = function(oldFocus:Object) {
  trace(this._name+" gained focus. Old focus changed from: "+oldFocus._name);
}
```

See Also

`TextField.onSetFocus`

TextField.onScroller

Availability

Flash Player 6.

Usage

```
my_txt.onScroller = function(textFieldInstance:TextField){
  // your statements here
}
myListenerObj.onScroller = function(textFieldInstance:TextField){
  // your statements here
}
```

Parameters

textFieldInstance A reference to the TextField object whose scroll position was changed.

Returns

Nothing.

Description

Event handler/listener; invoked when one of the text field scroll properties changes.

A reference to the text field instance is passed as a parameter to the `onScroller` handler. You can capture this data by putting a parameter in the event handler method. For example, the following code uses `my_txt` as the parameter that is passed to the `onScroller` event handler. The parameter is then used in a `trace()` statement to send the instance name of the text field to the Output panel.

```
myTextField.onScroller = function (my_txt:TextField) {
  trace (my_txt._name + " scrolled");
};
```

The `TextField.onScroller` event handler is commonly used to implement scroll bars. Scroll bars typically have a *thumb* or other indicator that shows the current horizontal or vertical scrolling position in a text field. Text fields can be navigated using the mouse and keyboard, which causes the scroll position to change. The scroll bar code needs to be notified if the scroll position changes because of such user interaction, which is what `TextField.onScroller` is used for.

`onScroller` is called whether the scroll position changed because of a users interaction with the text field, or programmatic changes. The `onChanged` handler fires only if a user interaction causes the change. These two options are necessary because often one piece of code changes the scrolling position, while the scroll bar code is unrelated and won't know that the scroll position changed without being notified.

Example

The following example creates a text field called `my_txt`, and uses two buttons called `scrollUp_btn` and `scrollDown_btn` to scroll the contents of the text field. When the `onScroller` event handler is called, a trace statement is used to display information in the Output panel. Create two buttons with instance names `scrollUp_btn` and `scrollDown_btn`, and add the following ActionScript to your FLA or AS file:

```
this.createTextField("scroll_txt", this.getNextHighestDepth(), 10, 10, 160,
  20);
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 30, 320, 240);
my_txt.multiline = true;
my_txt.wordWrap = true;

for (var i = 0; i<10; i++) {
  my_txt.text += "Lorem ipsum dolor sit amet, consectetuer adipiscing elit,
  sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat
  volutpat.";
}
scrollUp_btn.onRelease = function() {
  my_txt.scroll--;
};
scrollDown_btn.onRelease = function() {
  my_txt.scroll++;
};
my_txt.onScroller = function() {
  trace("onScroller called");
  scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
```

See also

`TextField.hscroll`, `TextField.maxhscroll`, `TextField.maxscroll`, `TextField.scroll`

TextField.onSetFocus

Availability

Flash Player 6.

Usage

```
my_txt.onSetFocus = function(oldFocus:Object){
  // your statements here
}
```

Parameters

oldFocus	The object to lose focus.

Returns

Nothing.

Description

Event handler; invoked when a text field receives keyboard focus. The *oldFocus* parameter is the object that loses the focus. For example, if the user presses the Tab key to move the input focus from a button to a text field, *oldFocus* contains the text field instance.

If there is no previously focused object, *oldFocus* contains a `null` value.

Example

See the example for TextField.onKillFocus.

See Also

`TextField.onKillFocus`

TextField._parent

Availability

Flash Player 6.

Usage

```
my_txt._parent.property
_parent.property
```

Description

Property; a reference to the movie clip or object that contains the current text field or object. The current object is the one containing the ActionScript code that references `_parent`.

Use `_parent` to specify a relative path to movie clips or objects that are above the current text field. You can use `_parent` to climb up multiple levels in the display list as in the following:

```
_parent._parent._alpha = 20;
```

Example

The following ActionScript creates two text fields and outputs information about the `_parent` of each object. The first text field, `first_txt`, is created on the main Timeline. The second text field, `second_txt`, is created inside the movie clip called `holder_mc`.

```
this.createTextField("first_txt", this.getNextHighestDepth(), 10, 10, 160,
  22);
first_txt.border = true;
trace(first_txt._name+"'s _parent is: "+first_txt._parent);

this.createEmptyMovieClip("holder_mc", this.getNextHighestDepth());
holder_mc.createTextField("second_txt", holder_mc.getNextHighestDepth(), 10,
  40, 160, 22);
holder_mc.second_txt.border = true;
trace(holder_mc.second_txt._name+"'s _parent is:
  "+holder_mc.second_txt._parent);
```

The following information is displayed in the Output panel:

```
first_txt's _parent is: _level0
second_txt's _parent is: _level0.holder_mc
```

See also

`Button._parent`, `MovieClip._parent`, `_root`, `targetPath()`

TextField.password

Availability

Flash Player 6.

Usage

```
my_txt.password:Boolean
```

Description

Property; if the value of `password` is `true`, the text field is a password text field and hides the input characters using asterisks instead of the actual characters. If `false`, the text field is not a password text field. When password mode is enabled, the *Cut* and *Copy* commands and their corresponding keyboard accelerators will not function. This security mechanism prevents an unscrupulous user from using the shortcuts to discover a password on an unattended computer.

Example

The following example creates two text fields: `username_txt` and `password_txt`. Text is entered into both text fields; however, `password_txt` has the `password` property set to `true`. Therefore, the characters display as asterisks instead of as characters in the `password_txt` field.

```
this.createTextField("username_txt", this.getNextHighestDepth(), 10, 10, 100,
  22);
username_txt.border = true;
username_txt.type = "input";
username_txt.maxChars = 16;
username_txt.text = "hello";

this.createTextField("password_txt", this.getNextHighestDepth(), 10, 40, 100,
  22);
password_txt.border = true;
password_txt.type = "input";
password_txt.maxChars = 16;
password_txt.password = true;
password_txt.text = "world";
```

TextField._quality

Availability

Flash Player 6.

Usage

`my_txt._quality:String`

Description

Property (global); sets or retrieves the rendering quality used for a SWF file. Device fonts are always aliased and, therefore, are unaffected by the `_quality` property.

Note: Although you can specify this property for a TextField object, it is actually a global property, and you can specify its value simply as `_quality`. For more information, see the _quality property.

The `_quality` property can be set to the following values:

- `"LOW"` Low rendering quality. Graphics are not anti-aliased, and bitmaps are not smoothed.
- `"MEDIUM"` Medium rendering quality. Graphics are anti-aliased using a 2 x 2 pixel grid, but bitmaps are not smoothed. Suitable for movies that do not contain text.
- `"HIGH"` High rendering quality. Graphics are anti-aliased using a 4 x 4 pixel grid, and bitmaps are smoothed if the movie is static. This is the default rendering quality setting used by Flash.
- `"BEST"` Very high rendering quality. Graphics are anti-aliased using a 4 x 4 pixel grid and bitmaps are always smoothed.

Example

The following example sets the rendering quality to `LOW`:

```
my_txt._quality = "LOW";
```

TextField.removeListener()

Availability

Flash Player 6.

Usage

my_txt.removeListener(*listener:Object*)

Parameters

listener The object that will no longer receive notifications from TextField.onChanged or TextField.onScroller.

Returns

If *listener* was successfully removed, the method returns a `true` value. If *listener* was not successfully removed (for example, if *listener* was not on the TextField object's listener list), the method returns a value of `false`.

Description

Method; removes a listener object previously registered to a text field instance with TextField.addListener().

Example

The following example creates an input text field called `my_txt`. When the user types into the field, information about the number of characters in the text field is displayed in the Output panel. If the user clicks the `removeListener_btn` instance, then the listener is removed and information is no longer displayed.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 20);
my_txt.border = true;
my_txt.type = "input";

var txtListener:Object = new Object();
txtListener.onChanged = function(textfield_txt:TextField) {
  trace(textfield_txt+" changed. Current length is: "+textfield_txt.length);
};
my_txt.addListener(txtListener);

removeListener_btn.onRelease = function() {
  trace("Removing listener...");
  if (!my_txt.removeListener(txtListener)) {
    trace("Error! Unable to remove listener");
  }
};
```

TextField.removeTextField()

Availability

Flash Player 6.

Usage

```
my_txt.removeTextField() : Void
```

Description

Method; removes the text field specified by *my_txt*. This operation can only be performed on a text field that was created with MovieClip.createTextField(). When you call this method, the text field is removed. This method is similar to `MovieClip.removeMovieClip()`.

Example

The following example creates a text field that you can remove from the Stage when you click the remove_btn instance. Create a button and call it `remove_btn`, and then add the following ActionScript to your FLA or AS file.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 300, 22);
my_txt.text = new Date().toString();
my_txt.border = true;

remove_btn.onRelease = function() {
  my_txt.removeTextField();
};
```

TextField.replaceSel()

Availability

Flash Player 6.

Usage

my_txt.replaceSel(*text:String*) : *Void*

Parameters

text A string.

Returns

Nothing.

Description

Method; replaces the current selection with the contents of the *text* parameter. The text is inserted at the position of the current selection, using the current default character format and default paragraph format. The text is not treated as HTML, even if the text field is an HTML text field.

You can use the `replaceSel()` method to insert and delete text without disrupting the character and paragraph formatting of the rest of the text.

You must use Selection.setFocus() to focus the field before issuing this command.

Example

The following example code creates a multiline text field with text on the Stage. When you select some text and then right-click or Control-click over the text field, you can select Enter current date from the context menu. This selection calls a function that replaces the selected text with the current date.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320, 240);
my_txt.border = true;
my_txt.wordWrap = true;
my_txt.multiline = true;
my_txt.type = "input";
my_txt.text = "Select some sample text from the text field and then right-
  click/control click and select 'Enter current date' from the context menu to
  replace the currently selected text with the current date.";

var my_cm:ContextMenu = new ContextMenu();
my_cm.customItems.push(new ContextMenuItem("Enter current date", enterDate));
function enterDate(obj:Object, menuItem:ContextMenuItem) {
  var today_str:String = new Date().toString();
  var date_str:String = today_str.split(" ", 3).join(" ");
  my_txt.replaceSel(date_str);
}
my_txt.menu = my_cm;
```

See also

`Selection.setFocus()`

TextField.replaceText()

Availability

Flash Player 7.

Usage

my_txt.replaceText(*beginIndex:Number*, *endIndex:Number*, *text:String*) : *Void*

Description

Method; replaces a range of characters, specified by the *beginIndex* and *endIndex* parameters, in the specified text field with the contents of the *text* parameter.

Example

The following example creates a text field called `my_txt` and assigns the text dog@house.net to the field. The `indexOf()` method is used to find the first occurrence of the specified symbol (@). If the symbol is found, the specified text (between the index of 0 and the symbol) replaces with the string `bird`. If the symbol is not found, an error message is displayed in the Output panel.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 320, 22);
my_txt.autoSize = true;
my_txt.text = "dog@house.net";

var symbol:String = "@";
var symbolPos:Number = my_txt.text.indexOf(symbol);
if (symbolPos>-1) {
  my_txt.replaceText(0, symbolPos, "bird");
} else {
  trace("symbol '"+symbol+"' not found.");
}
```

TextField.restrict

Availability

Flash Player 6.

Usage

```
my_txt.restrict:String
```

Description

Property; indicates the set of characters that a user may enter into the text field. If the value of the `restrict` property is `null`, you can enter any character. If the value of the `restrict` property is an empty string, you can't enter any character. If the value of the `restrict` property is a string of characters, you can enter only characters in the string into the text field. The string is scanned from left to right. A range may be specified using the dash (-). This only restricts user interaction; a script may put any text into the text field. This property does not synchronize with the Embed Font Outlines check boxes in the Property inspector.

If the string begins with ^, all characters are initially accepted and succeeding characters in the string are excluded from the set of accepted characters. If the string does not begin with ^, no characters are initially accepted and succeeding characters in the string are included in the set of accepted characters.

Example

The following example allows only uppercase characters, spaces, and numbers to be entered into a text field:

```
my_txt.restrict = "A-Z 0-9";
```

The following example includes all characters, but excludes lowercase letters:

```
my_txt.restrict = "^a-z";
```

You can use a backslash to enter a ^ or - verbatim. The accepted backslash sequences are \-, \^ or \\. The backslash must be an actual character in the string, so when specified in ActionScript, a double backslash must be used. For example, the following code includes only the dash (-) and caret (^):

```
my_txt.restrict = "\\-\\^";
```

The ^ may be used anywhere in the string to toggle between including characters and excluding characters. The following code includes only uppercase letters, but excludes the uppercase letter Q:

```
my_txt.restrict = "A-Z^Q";
```

You can use the `\u` escape sequence to construct `restrict` strings. The following code includes only the characters from ASCII 32 (space) to ASCII 126 (tilde).

```
my_txt.restrict = "\u0020-\u007E";
```

TextField._rotation

Availability

Flash Player 6.

Usage

my_txt._rotation:*Number*

Description

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

Rotation values are not supported for text files that use device fonts. You must use embedded fonts to use `_rotation` with a text field.

Example

In this example, you need to create a dynamic text field called `my_txt`, and then use the following ActionScript to embed fonts and rotate the text field. The reference to `my font` refers to a Font symbol in the library, with linkage set to `my font`.

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.font = "my font";

this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 160, 120);
my_txt.wordWrap = true;
my_txt.embedFonts = true;
my_txt.text = "Hello world";
my_txt.setTextFormat(my_fmt);
my_txt._rotation = 45;
```

Apply additional formatting for the text field using the TextFormat class.

See also

`Button._rotation`, `MovieClip._rotation`

TextField.scroll

Availability

Flash Player 6.

Usage

my_txt.scroll

Description

Property; defines the vertical position of text in a text field. The `scroll` property is useful for directing users to a specific paragraph in a long passage, or creating scrolling text fields. This property can be retrieved and modified.

The units of horizontal scrolling are pixels, while the units of vertical scrolling are lines. Horizontal scrolling is measured in pixels because most fonts you typically use are proportionally spaced; meaning, the characters can have different widths. Flash performs vertical scrolling by line because users usually want to see a line of text in its entirety, as opposed to seeing a partial line. Even if there are multiple fonts on a line, the height of the line adjusts to fit the largest font in use.

For more information on scrolling text, see "Creating scrolling text" in *Using Flash*.

Example

The following example sets the maximum value for the scrolling text field `my_txt`. Create two buttons, `scrollUp_btn` and `scrollDown_btn`, to scroll the text field. Add the following ActionScript to your FLA or AS file.

```
this.createTextField("scroll_txt", this.getNextHighestDepth(), 10, 10, 160,
  20);
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 30, 320, 240);
my_txt.multiline = true;
my_txt.wordWrap = true;
for (var i = 0; i<10; i++) {
  my_txt.text += "Lorem ipsum dolor sit amet, consectetuer adipiscing elit,
  sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat
  volutpat.";
}
scrollUp_btn.onRelease = function() {
  my_txt.scroll--;
  scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
scrollDown_btn.onRelease = function() {
  my_txt.scroll++;
  scroll_txt.text = my_txt.scroll+" of "+my_txt.maxscroll;
};
```

See also

`TextField.hscroll`, `TextField.maxscroll`

TextField.selectable

Availability

Flash Player 6.

Usage

```
my_txt.selectable:Boolean
```

Description

Property; a Boolean value that indicates whether the text field is selectable. The value `true` indicates that the text is selectable. The `selectable` property controls whether a text field is selectable, and not whether a text field is editable. A dynamic text field can be selectable even if it's not editable. If a dynamic text field is not selectable, that means you cannot select its text.

If selectable is set to `false`, the text in the text field does not respond to selection commands from the mouse or keyboard, and the text cannot be copied using the Copy command. If selectable is set to `true`, the text in the text field can be selected using the mouse or keyboard. You can select text this way even if the text field is a dynamic text field instead of an input text field. The text can be copied using the Copy command.

Example

The following example creates a selectable text field that constantly updates with the current date and time.

```
this.createTextField("date_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
date_txt.autoSize = true;
date_txt.selectable = false;

var date_interval:Number = setInterval(updateTime, 500, date_txt);
function updateTime(my_txt:TextField) {
  my_txt.text = new Date().toString();
}
```

TextField.setNewTextFormat()

Availability

Flash Player 6.

Usage

my_txt.setNewTextFormat(*textFormat:TextFormat*) *: Void*

Parameters

textFormat	A TextFormat object.

Returns

Nothing.

Description

Method; sets the default new text format of a text field; that is, the text format to be used for newly inserted text, such as text inserted with the `replaceSel()` method or text entered by a user. When text is inserted, the newly inserted text is assigned the default new text format.

The new default text format is specified by `textFormat`, which is a TextFormat object.

Example

In the following example, a new text field (called `my_txt`) is created at runtime and several properties are set. The format of the newly inserted text is applied.

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.bold = true;
my_fmt.font = "Arial";
my_fmt.color = 0xFF9900;

this.createTextField("my_txt", 999, 0, 0, 400, 300);
my_txt.wordWrap = true;
my_txt.multiline = true;
my_txt.border = true;
my_txt.type = "input";
my_txt.setNewTextFormat(my_fmt);
my_txt.text = "Oranges are a good source of vitamin C";
```

See also

`TextField.getNewTextFormat()`, `TextField.getTextFormat()`,
`TextField.setTextFormat()`

TextField.setTextFormat()

Availability

Flash Player 6.

Usage

```
my_txt.setTextFormat (textFormat:TextFormat) : Void
my_txt.setTextFormat (index:Number, textFormat:TextFormat) : Void
my_txt.setTextFormat (beginIndex:Number, endIndex:Number,
   textFormat:TextFormat) : Void
```

Parameters

`textFormat` A TextFormat object, which contains character and paragraph formatting information.

`index` An integer that specifies a character within `my_txt`.

`beginIndex` An integer.

`endIndex` An integer that specifies the first character after the desired text span.

Returns

Nothing.

Description

Method; applies the text formatting specified by textFormat to some or all of the text in a text field. `textFormat` must be a TextFormat object that specifies the text formatting changes desired. Only the non-null properties of textFormat are applied to the text field. Any property of `textFormat` that is set to null will not be applied. By default, all of the properties of a newly created TextFormat object are set to `null`.

There are two types of formatting information in a TextFormat object: character level, and paragraph level formatting. Each character in a text field might have its own character formatting settings, such as font name, font size, bold, and italic.

For paragraphs, the first character of the paragraph is examined for the paragraph formatting settings for the entire paragraph. Examples of paragraph formatting settings are left margin, right margin, and indentation.

The `setTextFormat()` method changes the text formatting applied to an individual character, to a range of characters, or to the entire body of text in a text field.

Usage 1: Applies the properties of `textFormat` to all text in the text field.

Usage 2: Applies the properties of `textFormat` to the character at position index.

Usage 3: Applies the properties of the `textFormat` parameter to the span of text from the beginIndex parameter to the endIndex parameter.

Notice that any text inserted manually by the user, or replaced by means of TextField.replaceSel(), receives the text field's default formatting for new text, and not the formatting specified for the text insertion point. To set a text field's default formatting for new text, use TextField.setNewTextFormat().

Example

The following example sets the text format for two different strings of text. The `setTextFormat()` method is called and applied to the `my_txt` text field.

```
var format1_fmt:TextFormat = new TextFormat();
format1_fmt.font = "Arial";
var format2_fmt:TextFormat = new TextFormat();
format2_fmt.font = "Courier";

var string1:String = "Sample string number one."+newline;
var string2:String = "Sample string number two."+newline;

this.createTextField("my_txt", this.getNextHighestDepth(), 0, 0, 300, 200);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.text = string1;
var firstIndex:Number = my_txt.length;
my_txt.text += string2;
var secondIndex:Number = my_txt.length;

my_txt.setTextFormat(0, firstIndex, format1_fmt);
my_txt.setTextFormat(firstIndex, secondIndex, format2_fmt);
```

See also

TextField.setNewTextFormat(), TextFormat class

TextField.styleSheet

Availability

Flash Player 7.

Usage

```
my_txt.styleSheet = TextField StyleSheet
```

Description

Property; attaches a style sheet to the text field specified by `my_txt`. For information on creating style sheets, see the TextField.StyleSheet class entry and "Formatting text with Cascading Style Sheets" in *Using ActionScript in Flash*.

The style sheet associated with a text field may be changed at any time. If the style sheet in use is changed, the text field is redrawn using the new style sheet. The style sheet may be set to `null` or `undefined` to remove the style sheet. If the style sheet in use is removed, the text field is redrawn without a style sheet. The formatting done by a style sheet is not retained if the style sheet is removed.

Example

The following example creates a new text field at runtime, called `news_txt`. Three buttons on the Stage, `css1_btn`, `css2_btn` and `clearCss_btn`, are used to change the style sheet that is applied to `news_txt`, or clear the style sheet from the text field. Add the following ActionScript to your FLA or AS file:

```
this.createTextField("news_txt", this.getNextHighestDepth(), 0, 0, 300, 200);
news_txt.wordWrap = true;
news_txt.multiline = true;
news_txt.html = true;
var newsText:String = "<p class='headline'>Description</p> Method; starts
  loading the CSS file into styleSheet. The load operation is asynchronous;
  use the <span class='bold'>TextField.StyleSheet.onLoad</span> callback
  handler to determine when the file has finished loading. <span
  class='important'>The CSS file must reside in exactly the same domain as the
  SWF file that is loading it.</span> For more information about restrictions
  on loading data across domains, see Flash Player security features.";
news_txt.htmlText = newsText;

css1_btn.onRelease = function() {
  var styleObj:TextField.StyleSheet = new TextField.StyleSheet();
  styleObj.onLoad = function(success:Boolean) {
    if (success) {
      news_txt.styleSheet = styleObj;
      news_txt.htmlText = newsText;
    }
  };
  styleObj.load("styles.css");
};

css2_btn.onRelease = function() {
  var styleObj:TextField.StyleSheet = new TextField.StyleSheet();
  styleObj.onLoad = function(success:Boolean) {
    if (success) {
      news_txt.styleSheet = styleObj;
      news_txt.htmlText = newsText;
    }
  };
  styleObj.load("styles2.css");
};

clearCss_btn.onRelease = function() {
  news_txt.styleSheet = undefined;
  news_txt.htmlText = newsText;
};
```

The following styles are applied to the text field. Save the following two CSS files in the same directory as the FLA or AS file you created previously:

```
/* in styles.css */
.important {
  color: #FF0000;
}
.bold {
  font-weight: bold;
}
.headline {
  color: #000000;
  font-family: Arial,Helvetica,sans-serif;
  font-size: 18px;
  font-weight: bold;
  display: block;
}


/* in styles2.css */
.important {
  color: #FF00FF;
}
.bold {
  font-weight: bold;
}
.headline {
  color: #00FF00;
  font-family: Arial,Helvetica,sans-serif;
  font-size: 18px;
  font-weight: bold;
  display: block;
}
```

TextField.tabEnabled

Availability

Flash Player 6.

Usage

my_txt.tabEnabled:*Boolean*

Description

Property; specifies whether *my_txt* is included in automatic tab ordering. It is undefined by default.

If the tabEnabled property is undefined or true, the object is included in automatic tab ordering. If the tabIndex property is also set to a value, the object is included in custom tab ordering as well. If tabEnabled is false, the object is not included in automatic or custom tab ordering, even if the tabIndex property is set.

Example

The following example creates several text fields, called one_txt, two_txt, three_txt and four_txt. The three_txt text field has the tabEnabled property set to false, so it is excluded from the automatic tab ordering.

```
this.createTextField("one_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
one_txt.border = true;
one_txt.type = "input";
this.createTextField("two_txt", this.getNextHighestDepth(), 10, 40, 100, 22);
two_txt.border = true;
two_txt.type = "input";
this.createTextField("three_txt", this.getNextHighestDepth(), 10, 70, 100,
  22);
three_txt.border = true;
three_txt.type = "input";
this.createTextField("four_txt", this.getNextHighestDepth(), 10, 100, 100,
  22);
four_txt.border = true;
four_txt.type = "input";

three_txt.tabEnabled = false;
three_txt.text = "tabEnabled = false;";
```

See also

Button.tabEnabled, MovieClip.tabEnabled

TextField.tabIndex

Availability

Flash Player 6.

Usage

my_txt.tabIndex:*Number*

Parameters

None.

Returns

Nothing.

Description

Property; lets you customize the tab ordering of objects in a SWF file. You can set the `tabIndex` property on a button, movie clip, or text field instance; it is `undefined` by default.

If any currently displayed object in the SWF file contains a `tabIndex` property, automatic tab ordering is disabled, and the tab ordering is calculated from the `tabIndex` properties of objects in the SWF file. The custom tab ordering only includes objects that have `tabIndex` properties.

The `tabIndex` property must be a positive integer. The objects are ordered according to their `tabIndex` properties, in ascending order. An object with a `tabIndex` value of `1` precedes an object with a `tabIndex` value of 2. If two objects have the same `tabIndex` value, the one that precedes the other in the tab ordering is `undefined`.

The custom tab ordering defined by the `tabIndex` property is *flat.* This means that no attention is paid to the hierarchical relationships of objects in the SWF file. All objects in the SWF file with `tabIndex` properties are placed in the tab order, and the tab order is determined by the order of the `tabIndex` values. If two objects have the same `tabIndex` value, the one that goes first is `undefined`. You shouldn't use the same `tabIndex` value for multiple objects.

Example

The following ActionScript dynamically creates four text fields and assigns them to a custom tab order. Add the following ActionScript to your FLA or AS file:

```
this.createTextField("one_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
one_txt.border = true;
one_txt.type = "input";
this.createTextField("two_txt", this.getNextHighestDepth(), 10, 40, 100, 22);
two_txt.border = true;
two_txt.type = "input";
this.createTextField("three_txt", this.getNextHighestDepth(), 10, 70, 100,
  22);
three_txt.border = true;
three_txt.type = "input";
this.createTextField("four_txt", this.getNextHighestDepth(), 10, 100, 100,
  22);
four_txt.border = true;
four_txt.type = "input";
```

```
one_txt.tabIndex = 3;
two_txt.tabIndex = 1;
three_txt.tabIndex = 2;
four_txt.tabIndex = 4;
```

See also

`Button.tabIndex`, `MovieClip.tabIndex`

TextField._target

Availability

Flash Player 6.

Usage

my_txt._target:*String*

Description

Read-only property; the target path of the text field instance specified by `my_txt`. The `_self` target specifies the current frame in the current window, `_blank` specifies a new window, `_parent` specifies the parent of the current frame, and `_top` specifies the top-level frame in the current window.

Example

The following ActionScript creates a text field called `my_txt` and outputs the target path of the new field, in both slash and dot notation.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 100, 22);
trace(my_txt._target); // output: /my_txt
trace(eval(my_txt._target)); // output: _level0.my_txt
```

TextField.text

Availability

Flash Player 6.

Usage

my_txt.text:*String*

Description

Property; indicates the current text in the text field. Lines are separated by the carriage return character ('\r', ASCII 13). This property contains the normal, unformatted text in the text field, without HTML tags, even if the text field is HTML.

Example

The following example creates an HTML text field called `my_txt`, and assigns an HTML-formatted string of text to the field. When you trace the `htmlText` property, the Output panel displays the HTML-formatted string. When you trace the value of the `text` property, the unformatted string with HTML tags displays in the Output panel.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 10, 400, 22);
my_txt.html = true;
my_txt.htmlText = "<b>Remember to always update the help panel.</b>";

trace("htmlText: "+my_txt.htmlText);
trace("text: "+my_txt.text);

/* output:
htmlText: <P ALIGN="LEFT"><FONT FACE="Times New Roman" SIZE="12"
  COLOR="#000000"><B>Remember to always update your help panel.</B></FONT></P>
text: Remember to always update your help panel.
*/
```

See also

`TextField.htmlText`

TextField.textColor

Availability

Flash Player 6.

Usage

my_txt.textColor:*Number*

Description

Property; indicates the color of the text in a text field. The hexadecimal color system uses six digits to represent color values. Each digit has sixteen possible values or characters. The characters range from 0 to 9 and then A to F. Black is represented by (#000000) and white, at the opposite end of the color system, is (#FFFFFF).

Example

The following ActionScript creates a text field and changes its color property to red.

```
this.createTextField("my_txt", 99, 10, 10, 100, 300);
my_txt.text = "this will be red text";
my_txt.textColor = 0xFF0000;
```

TextField.textHeight

Availability

Flash Player 6.

Usage

```
my_txt.textHeight:Number
```

Description

Property; indicates the height of the text.

Example

The following example creates a text field, and assigns a string of text to the field. A trace statement is used to display the text height and width in the Output panel. The `autoSize` property is then used to resize the text field, and the new height and width will also be displayed in the Output panel.

```
this.createTextField("my_txt", 99, 10, 10, 100, 300);
my_txt.text = "Sample text";
trace("textHeight: "+my_txt.textHeight+", textWidth: "+my_txt.textWidth);
trace("_height: "+my_txt._height+", _width: "+my_txt._width+"\n");
my_txt.autoSize = true;
trace("after my_txt.autoSize = true;");
trace("_height: "+my_txt._height+", _width: "+my_txt._width);
```

Which outputs the following information:

```
textHeight: 15, textWidth: 56
_height: 300, _width: 100

after my_txt.autoSize = true;
_height: 19, _width: 60
```

See Also

`TextField.textWidth`

TextField.textWidth

Availability

Flash Player 6.

Usage

```
my_txt.textWidth:Number
```

Description

Property; indicates the width of the text.

Example

See the example for TextField.textHeight.

See Also

`TextField.textHeight`

TextField.type

Availability

Flash Player 6.

Usage

my_txt.type:*String*

Description

Property; Specifies the type of text field. There are two values: "dynamic", which specifies a dynamic text field that cannot be edited by the user, and "input", which specifies an input text field.

Example

The following example creates two text fields: username_txt and password_txt. Text is entered into both text fields; however, password_txt has the password property set to true. Therefore, the characters display as asterisks instead of as characters in the password_txt field.

```
this.createTextField("username_txt", this.getNextHighestDepth(), 10, 10, 100,
  22);
username_txt.border = true;
username_txt.type = "input";
username_txt.maxChars = 16;
username_txt.text = "hello";

this.createTextField("password_txt", this.getNextHighestDepth(), 10, 40, 100,
  22);
password_txt.border = true;
password_txt.type = "input";
password_txt.maxChars = 16;
password_txt.password = true;
password_txt.text = "world";
```

TextField._url

Availability

Flash Player 6.

Usage

my_txt._url*:String*

Description

Read-only property; retrieves the URL of the SWF file that created the text field.

Example

The following example retrieves the URL of the SWF file that created the text field, and a SWF file that loads into it.

```
this.createTextField("my_txt", 1, 10, 10, 100, 22);
trace(my_txt._url);

var mclListener:Object = new Object();
mclListener.onLoadInit = function(target_mc:MovieClip) {
  trace(target_mc._url);
};
var holder_mcl:MovieClipLoader = new MovieClipLoader();
holder_mcl.addListener(mclListener);
holder_mcl.loadClip("best_flash_ever.swf",
  this.createEmptyMovieClip("holder_mc", 2));
```

When you test this example, the URL of the SWF file you're testing, and the file called best_flash_ever.swf are displayed in the Output panel.

TextField.variable

Availability

Flash Player 6.

Usage

my_txt.variable*:String*

Description

Property; The name of the variable that the text field is associated with. The type of this property is String.

Example

The following example creates a text field called `my_txt` and associates the variable `today_date` with the text field. When you change the variable `today_date`, then the text that displays in `my_txt` updates.

```
this.createTextField("my_txt", 1, 10, 10, 200, 22);
my_txt.variable = "today_date";
var today_date:Date = new Date();

var date_interval:Number = setInterval(updateDate, 500);
function updateDate():Void {
  today_date = new Date();
}
```

TextField._visible

Availability

Flash Player 6.

Usage

my_txt._visible:*Boolean*

Description

Property; a Boolean value that indicates whether the text field *my_txt* is visible. Text fields that are not visible (_visible property set to false) are disabled.

Example

The following example creates a text field called my_txt. A button called visible_btn toggles the visibility of my_txt.

```
this.createTextField("my_txt", 1, 10, 10, 200, 22);
my_txt.background = true;
my_txt.backgroundColor = 0xDFDFDF;
my_txt.border = true;
my_txt.type = "input";

visible_btn.onRelease = function() {
  my_txt._visible = !my_txt._visible;
};
```

See also

Button._visible, MovieClip._visible

TextField._width

Availability

Flash Player 6.

Usage

my_txt._width:*Number*

Description

Property; the width of the text field, in pixels.

Example

The following example creates two text fields that you can use to change the width and height of a third text field on the Stage. Add the following ActionScript to a FLA or AS file.

```
this.createTextField("my_txt", this.getNextHighestDepth(), 10, 40, 160, 120);
my_txt.background = true;
my_txt.backgroundColor = 0xFF0000;
my_txt.border = true;
my_txt.multiline = true;
my_txt.type = "input";
my_txt.wordWrap = true;

this.createTextField("width_txt", this.getNextHighestDepth(), 10, 10, 30, 20);
width_txt.border = true;
width_txt.maxChars = 3;
width_txt.restrict = "0-9";
width_txt.type = "input";
width_txt.text = my_txt._width;
width_txt.onChanged = function() {
  my_txt._width = this.text;
}

this.createTextField("height_txt", this.getNextHighestDepth(), 70, 10, 30,
  20);
height_txt.border = true;
height_txt.maxChars = 3;
height_txt.restrict = "0-9";
height_txt.type = "input";
height_txt.text = my_txt._height;
height_txt.onChanged = function() {
  my_txt._height = this.text;
}
```

When you test the example, try entering new values into `width_txt` and `height_txt` to change the dimensions of `my_txt`.

See also

`TextField._height`

TextField.wordWrap

Availability

Flash Player 6.

Usage

```
my_txt.wordWrap:Boolean
```

Description

Property; a Boolean value that indicates if the text field has word wrap. If the value of `wordWrap` is `true`, the text field has word wrap; if the value is `false`, the text field does not have word wrap.

Example

The following example demonstrates how `wordWrap` affects long text in a text field that's created at runtime.

```
this.createTextField("my_txt", 99, 10, 10, 100, 200);
my_txt.text = "This is very long text that will certainly extend beyond the
  width of this text field";
my_txt.border = true;
```

Test the SWF file in Flash Player by selecting Control > Test Movie. Then return to your ActionScript and add the following line to the code and test the SWF file again:

```
my_txt.wordWrap = true;
```

TextField._x

Availability

Flash Player 6.

Usage

*my_txt.*_x:*Number*

Description

Property; an integer that sets the *x* coordinate of a text field relative to the local coordinates of the parent movie clip. If a text field is on the main Timeline, then its coordinate system refers to the upper left corner of the Stage as (0, 0). If the text field is inside a movie clip that has transformations, the text field is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90° counterclockwise, the enclosed text field inherits a coordinate system that is rotated 90° counterclockwise. The text field's coordinates refer to the registration point position.

Example

The following example creates a text field wherever you click the mouse. When it creates a text field, that field displays the current *x* and *y* coordinates of the text field.

```
this.createTextField("coords_txt", this.getNextHighestDepth(), 0, 0, 60, 22);
coords_txt.autoSize = true;
coords_txt.selectable = false;
coords_txt.border = true;

var mouseListener:Object = new Object();
mouseListener.onMouseDown = function() {
  coords_txt.text = "X:"+Math.round(_xmouse)+", Y:"+Math.round(_ymouse);
  coords_txt._x = _xmouse;
  coords_txt._y = _ymouse;
};
Mouse.addListener(mouseListener);
```

See also

TextField._xscale, TextField._y, TextField._yscale

TextField._xmouse

Availability

Flash Player 6.

Usage

```
my_txt._xmouse:Number
```

Description

Read-only property; returns the *x* coordinate of the mouse position relative to the text field.

Example

The following example creates three text fields on the Stage. The `mouse_txt` instance displays the current position of the mouse in relation to the Stage. The `textfield_txt` instance displays the current position of the mouse pointer in relation to the `my_txt` instance. Add the following ActionScript to a FLA or AS file:

```
this.createTextField("mouse_txt", this.getNextHighestDepth(), 10, 10, 200,
  22);
mouse_txt.border = true;
this.createTextField("textfield_txt", this.getNextHighestDepth(), 220, 10,
  200, 22);
textfield_txt.border = true;
this.createTextField("my_txt", this.getNextHighestDepth(), 100, 100, 160,
  120);
my_txt.border = true;

var mouseListener:Object = new Object();
mouseListener.onMouseMove = function() {
  mouse_txt.text = "MOUSE ...
  X:"+Math.round(_xmouse)+",\tY:"+Math.round(_ymouse);
  textfield_txt.text = "TEXTFIELD ...
  X:"+Math.round(my_txt._xmouse)+",\tY:"+Math.round(my_txt._ymouse);
}

Mouse.addListener(mouseListener);
```

See also

`TextField._ymouse`

TextField._xscale

Availability

Flash Player 6.

Usage

my_txt._xscale:*Number*

Description

Property; determines the horizontal scale of the text field as applied from the registration point of the text field, expressed as a percentage. The default registration point is (0,0).

Example

The following example scales the `my_txt` instance when you click the `scaleUp_btn` and `scaleDown_btn` instances.

```
this.createTextField("my_txt", 99, 10, 40, 100, 22);
my_txt.autoSize = true;
my_txt.border = true;
my_txt.selectable = false;
my_txt.text = "Sample text goes here.";

scaleUp_btn.onRelease = function() {
  my_txt._xscale *= 2;
  my_txt._yscale *= 2;
}
scaleDown_btn.onRelease = function() {
  my_txt._xscale /= 2;
  my_txt._yscale /= 2;
}
```

See also

`TextField._x`, `TextField._y`, `TextField._yscale`

TextField._y

Availability

Flash Player 6.

Usage

`my_txt._y:Number`

Description

Property; the *y* coordinate of a text field relative to the local coordinates of the parent movie clip. If a text field is in the main Timeline, then its coordinate system refers to the upper left corner of the Stage as (0, 0). If the text field is inside another movie clip that has transformations, the text field is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90° counterclockwise, the enclosed text field inherits a coordinate system that is rotated 90° counterclockwise. The text field's coordinates refer to the registration point position.

Example

See the example for TextField._x.

See also

`TextField._x`, `TextField._xscale`, `TextField._yscale`

TextField._ymouse

Availability

Flash Player 6.

Usage

```
my_txt._ymouse:Number
```

Description

Read-only property; indicates the *y* coordinate of the mouse position relative to the text field.

Example

See the example for TextField._xmouse.

See also

`TextField._xmouse`

TextField._yscale

Availability

Flash Player 6.

Usage

my_txt._yscale:*Number*

Description

Property; the vertical scale of the text field as applied from the registration point of the text field, expressed as a percentage. The default registration point is (0,0).

Example

See the example for TextField._xscale.

See also

`TextField._x`, `TextField._xscale`, `TextField._y`

TextFormat class

Availability

Flash Player 6.

Description

The TextFormat class represents character formatting information.

You must use the constructor `new TextFormat()` to create a TextFormat object before calling its methods.

You can set TextFormat parameters to `null` to indicate that they are undefined. When you apply a TextFormat object to a text field using TextField.setTextFormat(), only its defined properties are applied, as in the following example:

```
this.createTextField("my_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
my_txt.autoSize = true;
my_txt.text = "Lorem ipsum dolor sit amet...";

var my_fmt:TextFormat = new TextFormat();
my_fmt.bold = true;
my_txt.setTextFormat(my_fmt);
```

This code first creates an empty TextFormat object with all of its properties `null`, and then sets the `bold` property to a defined value.

The code `my_txt.setTextFormat(my_fmt)` only changes the `bold` property of the text field's default text format, because the `bold` property is the only one defined in `my_fmt`. All other aspects of the text field's default text format remain unchanged.

When TextField.getTextFormat() is invoked, a TextFormat object is returned with all of its properties defined; no property is `null`.

Method summary for the TextFormat class

Method	Description
`TextFormat.getTextExtent()`	Returns text measurement information for a text string.

Property summary for the TextFormat class

Property	Description
`TextFormat.align`	Indicates the alignment of a paragraph.
`TextFormat.blockIndent`	Indicates the block indentation, in points.
`TextFormat.bold`	Indicates whether text is boldface.
`TextFormat.bullet`	Indicates whether text is in a bulleted list.
`TextFormat.color`	Indicates the color of text.
`TextFormat.font`	Indicates the font name of the text with a text format.

Property	Description
`TextFormat.indent`	Indicates the indentation from the left margin to the first character in the paragraph.
`TextFormat.italic`	Indicates whether text is italicized.
`TextFormat.leading`	Indicates the amount of vertical space (called *leading*) between lines.
`TextFormat.leftMargin`	Indicates the left margin of the paragraph, in points.
`TextFormat.rightMargin`	Indicates the right margin of the paragraph, in points.
`TextFormat.size`	Indicates the point size of text.
`TextFormat.tabStops`	Specifies custom tab stops.
`TextFormat.target`	Indicates the window in a browser where a hyperlink is displayed.
`TextFormat.underline`	Indicates whether text is underlined.
`TextFormat.url`	Indicates the URL to which the text links.

Constructor for the TextFormat class

Availability

Flash Player 6.

Usage

```
new TextFormat([font:String, [size:Number, [color:Number, [bold:Boolean,
  [italic:Boolean, [underline:Boolean, [url:String, [target:String,
  [align:String, [leftMargin:Number, [rightMargin:Number, [indent:Number,
  [leading:Number]]]]]]]]]]]]]) : TextFormat
```

Parameters

`font` The name of a font for text as a string.

`size` An integer that indicates the point size.

`color` The color of text using this text format. A number containing three 8-bit RGB components; for example, 0xFF0000 is red, and 0x00FF00 is green.

`bold` A Boolean value that indicates whether the text is boldface.

`italic` A Boolean value that indicates whether the text is italicized.

`underline` A Boolean value that indicates whether the text is underlined.

`url` The URL to which the text in this text format hyperlinks. If *`url`* is an empty string, the text does not have a hyperlink.

`target` The target window where the hyperlink is displayed. If the target window is an empty string, the text is displayed in the default target window `_self`. If the *`url`* parameter is set to an empty string or to the value `null`, you can get or set this property, but the property will have no effect.

`align` The alignment of the paragraph, represented as a string. If `"left"`, the paragraph is left-aligned. If `"center"`, the paragraph is centered. If `"right"`, the paragraph is right-aligned.

`leftMargin` Indicates the left margin of the paragraph, in points.

`rightMargin` Indicates the right margin of the paragraph, in points.

`indent` An integer that indicates the indentation from the left margin to the first character in the paragraph.

`leading` A number that indicates the amount of leading vertical space between lines.

Returns

A reference to a TextFormat object.

Description

Constructor; creates a TextFormat object with the specified properties. You can then change the properties of the TextFormat object to change the formatting of text fields.

Any parameter may be set to `null` to indicate that it is not defined. All of the parameters are optional; any omitted parameters are treated as `null`.

Example

The following example creates a TextFormat object, formats the `stats_txt` text field, and creates a new text field to display the text in:

```
// Define a TextFormat which is used to format the stats_txt text field.
var my_fmt:TextFormat = new TextFormat();
my_fmt.bold = true;
my_fmt.font = "Arial";
my_fmt.size = 12;
my_fmt.color = 0xFF0000;
// Create a text field to display the player's statistics.
this.createTextField("stats_txt", 5000, 10, 0, 530, 22);
// Apply the TextFormat to the text field.
stats_txt.setNewTextFormat(my_fmt);
stats_txt.selectable = false;
stats_txt.text = "Lorem ipsum dolor sit amet...";
```

To view another example, see the *animations.fla* file in the HelpExamples Folder. The following list provides typical paths to the HelpExamples Folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/

TextFormat.align

Availability

Flash Player 6.

Usage

my_fmt.align:*String*

Description

Property; indicates the alignment of the paragraph, represented as a string. The alignment of the paragraph, represented as a string. If `"left"`, the paragraph is left-aligned. If `"center"`, the paragraph is centered. If `"right"`, the paragraph is right-aligned. The default value is `null` which indicates that the property is undefined.

Example

The following example creates a text field with a border and uses `TextFormat.align` to center the text.

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.align = "center";

this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "this is my first test field object text";
my_txt.setTextFormat(my_fmt);
```

TextFormat.blockIndent

Availability

Flash Player 6.

Usage

my_fmt.blockIndent:*Number*

Description

Property; a number that indicates the block indentation in points. Block indentation is applied to an entire block of text; that is, to all lines of the text. In contrast, normal indentation (TextFormat.indent) affects only the first line of each paragraph. If this property is null, the TextFormat object does not specify block indentation.

Example

This example creates a text field with a border and sets the blockIndent to 20.

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.blockIndent = 20;

mytext.text = "this is my first test field object text";
mytext.setTextFormat(myformat);
```

TextFormat.bold

Availability

Flash Player 6.

Usage

my_fmt.bold:*Boolean*

Description

Property; a Boolean value that specifies whether the text is boldface. The default value is `null`, which indicates that the property is undefined. If the value is `true`, then the text is boldface.

Example

The following example creates a text field that includes characters in boldface.

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.bold = true;

this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "this is my test field object text";
my_txt.setTextFormat(my_fmt);
```

TextFormat.bullet

Availability

Flash Player 6.

Usage

my_fmt.bullet*:Boolean*

Description

Property; a Boolean value that indicates that the text is part of a bulleted list. In a bulleted list, each paragraph of text is indented. To the left of the first line of each paragraph, a bullet symbol is displayed. The default value is `null`.

Example

The following example creates a new text field at runtime, and enters a string with a line break into the field. The TextFormat class is used to format the characters by adding bullets to each line in the text field. This is demonstrated in the following ActionScript:

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.bullet = true;

this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "this is my text"+newline;
my_txt.text += "this is more text"+newline;
my_txt.setTextFormat(my_fmt);
```

TextFormat.color

Availability

Flash Player 6.

Usage

my_fmt.color:*Number*

Description

Property; indicates the color of text. A number containing three 8-bit RGB components; for example, 0xFF0000 is red, and 0x00FF00 is green.

Example

The following example creates a text field and sets the text color to red.

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.blockIndent = 20;

this.createTextField("my_txt", 1, 100, 100, 300, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "this is my first test field object text";
my_txt.setTextFormat(my_fmt);
```

TextFormat.font

Availability

Flash Player 6.

Usage

```
my_fmt.font:String
```

Description

Property; the name of the font for text in this text format, as a string. The default value is `null`, which indicates that the property is undefined.

Example

The following example creates a text field and sets the font to Courier.

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.font = "Courier";

mytext.text = "this is my first test field object text";
mytext.setTextFormat(myformat);
```

TextFormat.getTextExtent()

Availability

Flash Player 6. The optional *width* parameter is supported in Flash Player 7.

Usage

```
my_fmt.getTextExtent(text:String, [width:Number]) : Object
```

Parameters

text A string.

width An optional number that represents the width, in pixels, at which the specified text should wrap.

Returns

An object with the properties `width`, `height`, `ascent`, `descent`, `textFieldHeight`, `textFieldWidth`.

Description

Method; returns text measurement information for the text string *text* in the format specified by *my_fmt*. The text string is treated as plain text (not HTML).

The method returns an object with six properties: `ascent`, `descent`, `width`, `height`, `textFieldHeight`, and `textFieldWidth`. All measurements are in pixels.

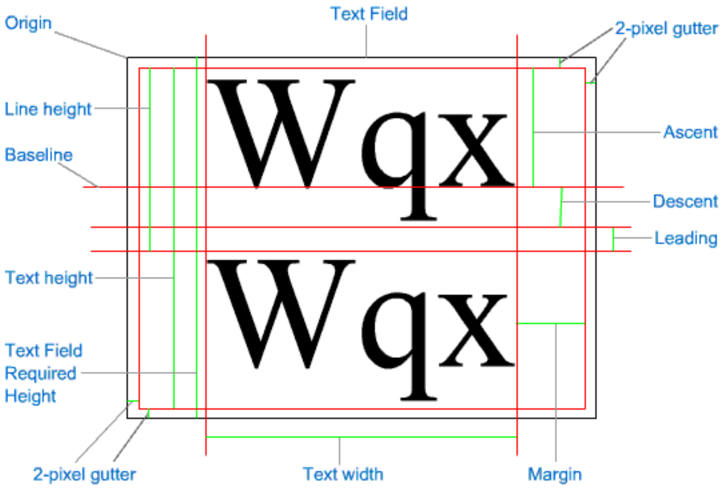
If a *width* parameter is specified, word wrapping is applied to the specified text. This lets you determine the height at which a text box shows all of the specified text.

The `ascent` and `descent` measurements provide, respectively, the distance above and below the baseline for a line of text. The baseline for the first line of text is positioned at the text field's origin plus its `ascent` measurement.

The `width` and `height` measurements provide the width and height of the text string. The `textFieldHeight` and `textFieldWidth` measurements provide the height and width required for a text field object to display the entire text string. Text fields have a 2-pixel-wide "gutter" around them, so the value of `textFieldHeight` is equal the value of `height` + 4; likewise, the value of `textFieldWidth` is always equal to the value of `width` + 4.

If you are creating a text field based on the text metrics, use `textFieldHeight` rather than `height` and `textFieldWidth` rather than `width`.

The following figure illustrates these measurements.



When setting up your TextFormat object, set all the attributes exactly as they will be set for the creation of the text field, including font name, font size, and leading. The default value for leading is 2.

Example

This example creates a single-line text field that's just big enough to display a text string using the specified formatting.

```
var my_str:String = "Small string";

// Create a TextFormat object,
// and apply its properties.
var my_fmt:TextFormat = new TextFormat();
with (my_fmt) {
  font = "Arial";
  bold = true;
}

// Obtain metrics information for the text string
// with the specified formatting.
var metrics:Object = my_fmt.getTextExtent(my_str);

// Create a text field just large enough to display the text.
this.createTextField("my_txt", this.getNextHighestDepth(), 100, 100,
  metrics.textFieldWidth, metrics.textFieldHeight);
my_txt.border = true;
my_txt.wordWrap = true;
// Assign the same text string and TextFormat object to the my_txt object.
my_txt.text = my_str;
my_txt.setTextFormat(my_fmt);
```

The following example creates a multiline, 100-pixel-wide text field that's high enough to display a string with the specified formatting.

```
// Create a TextFormat object.
var my_fmt:TextFormat = new TextFormat();
// Specify formatting properties for the TextFormat object:
my_fmt.font = "Arial";
my_fmt.bold = true;
my_fmt.leading = 4;

// The string of text to be displayed
var textToDisplay:String = "Macromedia Flash Player 7, now with improved text
  metrics.";

// Obtain text measurement information for the string,
// wrapped at 100 pixels.
var metrics:Object = my_fmt.getTextExtent(textToDisplay, 100);

// Create a new TextField object using the metric
// information just obtained.
this.createTextField("my_txt", this.getNextHighestDepth(), 50, 50-
  metrics.ascent, 100, metrics.textFieldHeight);
my_txt.wordWrap = true;
my_txt.border = true;
// Assign the text and the TextFormat object to the TextObject:
my_txt.text = textToDisplay;
my_txt.setTextFormat(my_fmt);
```

TextFormat.indent

Availability

Flash Player 6.

Usage

my_fmt.indent:*Number*

Description

Property; an integer that indicates the indentation from the left margin to the first character in the paragraph. The default value is `null`, which indicates that the property is undefined.

Example

The following example creates a text field and sets the indentation to 10.

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.indent = 10;

mytext.text = "this is my first test field object text";
mytext.setTextFormat(myformat);
```

See also

`TextFormat.blockIndent`

TextFormat.italic

Availability

Flash Player 6.

Usage

my_fmt.italic:*Boolean*

Description

Property; a Boolean value that indicates whether text in this text format is italicized. The default value is `null`, which indicates that the property is undefined.

Example

The following example creates a text field and sets the text style to italic.

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.italic = true;

mytext.text = "this is my first test field object text";
mytext.setTextFormat(myformat);
```

TextFormat.leading

Availability

Flash Player 6.

Usage

```
my_fmt.leading:Number
```

Description

Property; the amount of vertical space (called *leading*) between lines. The default value is `null`, which indicates that the property is undefined.

Example

The following example creates a text field and sets the leading to 10.

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.leading = 10;

this.createTextField("my_txt", 1, 100, 100, 100, 100);
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.border = true;
my_txt.text = "this is my first test field object text";
my_txt.setTextFormat(my_fmt);
```

TextFormat.leftMargin

Availability

Flash Player 6.

Usage

my_fmt.leftMargin:*Number*

Description

Property; the left margin of the paragraph, in points. The default value is `null`, which indicates that the property is undefined.

Example

The following example creates a text field and sets the left margin to 20 points.

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.leftMargin = 20;

mytext.text = "this is my first test field object text";
mytext.setTextFormat(myformat);
```

TextFormat.rightMargin

Availability

Flash Player 6.

Usage

my_fmt.rightMargin:*Number*

Description

Property; the right margin of the paragraph, in points. The default value is `null`, which indicates that the property is undefined.

Example

The following example creates a text field and sets the right margin to 20 points.

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.rightMargin = 20;

mytext.text = "this is my first test field object text";
mytext.setTextFormat(myformat);
```

TextFormat.size

Availability

Flash Player 6.

Usage

my_fmt.size:*Number*

Description

Property; the point size of text in this text format. The default value is `null`, which indicates that the property is undefined.

Example

The following example creates a text field and sets the text size to 20 points.

```
this.createTextField("mytext",1,100,100,100,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.size = 20;

mytext.text = "this is my first test field object text";
mytext.setTextFormat(myformat);
```

TextFormat.tabStops

Availability

Flash Player 6.

Usage

my_fmt.tabStops:*Array*

Description

Property; specifies custom tab stops as an array of non-negative integers. Each tab stop is specified in pixels. If custom tab stops are not specified (`null`), the default tab stop is 4 (average character width).

Example

The following example creates two text fields, one with tab stops every 20 pixels, and the other with tab stops every 40 pixels.

```
this.createTextField("mytext",1,100,100,200,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;
var myformat:TextFormat = new TextFormat();
myformat.tabStops = [20,40,60,80];
mytext.text = "ABCD";
mytext.setTextFormat(myformat);

this.createTextField("mytext2",2,100,220,200,100);
mytext2.multiline = true;
mytext2.wordWrap = true;
mytext2.border = true;
var myformat2:TextFormat = new TextFormat();
myformat2.tabStops = [40,80,120,160];
mytext2.text ="ABCD";
mytext2.setTextFormat(myformat2);
```

TextFormat.target

Availability

Flash Player 6.

Usage

my_fmt.target:*String*

Description

Property; indicates the target window where the hyperlink is displayed. If the target window is an empty string, the text is displayed in the default target window `_self`. You can choose a custom name or one of the following four names: `_self` specifies the current frame in the current window, `_blank` specifies a new window, `_parent` specifies the parent of the current frame, and `_top` specifies the top-level frame in the current window. If the TextFormat.url property is an empty string or `null`, you can get or set this property, but the property will have no effect.

Example

The following example creates a text field with a hyperlink to the Macromedia website. The example uses `TextFormat.target` to display the Macromedia website in a new browser window.

```
var myformat:TextFormat = new TextFormat();
myformat.url = "http://www.macromedia.com";
myformat.target = "_blank";

this.createTextField("mytext",1,100,100,200,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;
mytext.html = true;
mytext.text = "Go to Macromedia.com";
mytext.setTextFormat(myformat);
```

TextFormat.underline

Availability

Flash Player 6.

Usage

`my_fmt.underline:Boolean`

Description

Property; a Boolean value that indicates whether the text that uses this text format is underlined (`true`) or not (`false`). This underlining is similar to that produced by the `<U>` tag, but the latter is not "true" underlining, because it does not skip descenders correctly. The default value is `null`, which indicates that the property is undefined.

Example

The following example creates a text field and sets the text style to underline.

```
this.createTextField("mytext",1,100,100,200,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;

var myformat:TextFormat = new TextFormat();
myformat.underline = true;
mytext.text = "this is my first test field object text";
mytext.setTextFormat(myformat);
```

TextFormat.url

Availability

Flash Player 6.

Usage

my_fmt.url*:String*

Description

Property; indicates the URL that text in this text format hyperlinks to. If the `url` property is an empty string, the text does not have a hyperlink. The default value is `null`, which indicates that the property is undefined.

Example

This example creates a text field that is a hyperlink to the Macromedia website.

```
var myformat:TextFormat = new TextFormat();
myformat.url = "http://www.macromedia.com";

this.createTextField("mytext",1,100,100,200,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = true;
mytext.html = true;
mytext.text = "Go to Macromedia.com";
mytext.setTextFormat(myformat);
```

TextSnapshot object

Availability

Authoring: Flash MX 2004.

Playback: SWF files published for Flash Player 6 or later, playing in Flash Player 7 or later.

Description

TextSnapshot objects let you work with static text in a movie clip. You can use them, for example, to lay out text with greater precision than that allowed by dynamic text, but still access the text in a read-only way.

You don't use a constructor to create a TextSnapshot object; it is returned by MovieClip.getTextSnapshot().

Method summary for the TextSnapshot object

Method	Description
`TextSnapshot.findText()`	Returns the position of the first occurrence of specified text.
`TextSnapshot.getCount()`	Returns the number of characters.
`TextSnapshot.getSelected()`	Specifies whether any of the text in the specified range has been selected by TextSnapshot.setSelected().
`TextSnapshot.getSelectedText()`	Returns a string that contains all the characters specified by TextSnapshot.setSelected().
`TextSnapshot.getText()`	Returns a string containing the characters in the specified range.
`TextSnapshot.hitTestTextNearPos()`	Lets you determine which character within the object is on or near specified coordinates.
`TextSnapshot.setSelectColor()`	Specifies the color to use when highlighting characters that have been selected with the TextSnapshot.setSelected() command.
`TextSnapshot.setSelected()`	Specifies a range of characters to be selected or deselected.

TextSnapshot.findText()

Availability

Authoring: Flash MX 2004.

Playback: SWF files published for Flash Player 6 or later, playing in Flash Player 7 or later.

Usage

```
my_snap.findText( startIndex:Number, textToFind:String, caseSensitive:Boolean
  ) : Number
```

Parameters

startIndex An integer specifying the starting point in *my_snap* to search for the specified text.

textToFind A string specifying the text to search for. If you specify a string literal instead of a variable of type String, enclose the string in quotation marks.

caseSensitive A Boolean value specifying whether the text in *my_snap* must match the case of the string in *textToFind*.

Returns

The zero-based index position of the first occurrence of the specified text, or -1.

Description

Method; searches the specified TextSnapshot object and returns the position of the first occurrence of *textToFind* found at or after *startIndex*. If *textToFind* is not found, the method returns -1.

Example

The following example illustrates how to use this method. To use this code, place a static text field containing the text "TextSnapshot Example" on the Stage.

```
var my_mc:MovieClip = this;
var my_snap:TextSnapshot = my_mc.getTextSnapshot();
var index1:Number = my_snap.findText(0, "Snap", true);
var index2:Number = my_snap.findText(0, "snap", true);
var index3:Number = my_snap.findText(0, "snap", false);
trace(index1); // output: 4
trace(index2); // output: -1
trace(index3); // output: 4
```

See also

`TextSnapshot.getText()`

TextSnapshot.getCount()

Availability

Authoring: Flash MX 2004.

Playback: SWF files published for Flash Player 6 or later, playing in Flash Player 7 or later.

Usage

my_snap.getCount() *: Number*

Parameters

None.

Returns

An integer representing the number of characters in the specified TextSnapshot object.

Description

Method; returns the number of characters in a TextSnapshot object.

Example

The following example illustrates how you can output the number of characters in a specified TextSnapshot object. To use this code, place a static text field containing the text "TextSnapshot Example" on the Stage.

```
// this example assumes that the movie clip contains
// the static text "TextSnapshot Example"
var my_mc:MovieClip = this;
var my_snap:TextSnapshot = my_mc.getTextSnapshot();
var count:Number = my_snap.getCount();
var theText:String = my_snap.getText(0, count, false);
trace(count); // output: 20
trace(theText); // output: TextSnapshot Example
```

See also

TextSnapshot.getText()

TextSnapshot.getSelected()

Availability

Authoring: Flash MX 2004.

Playback: SWF files published for Flash Player 6 or later, playing in Flash Player 7 or later.

Usage

```
my_snap.getSelected(from:Number, to:Number) : Boolean
```

Parameters

from An integer that indicates the position of the first character of *my_snap* to be examined. Valid values for *from* are 0 through TextSnapshot.getCount() - 1. If *from* is a negative value, 0 is used.

to An integer that is 1+ the index of the last character in *my_snap* to be examined. Valid values for *to* are 0 through `TextSnapshot.getCount()`. The character indexed by the *to* parameter is not included in the extracted string. If this parameter is omitted, `TextSnapshot.getCount()` is used. If this value is less than or equal to the value of `from`, `from+1` is used.

Returns

A Boolean value of `true`, if at least one character in the given range has been selected by the corresponding TextSnapshot.setSelected() command, `false` otherwise.

Description

Method; returns a Boolean value that specifies whether a TextSnapshot object contains selected text in the specified range.

To search all characters, pass a value of 0 for *from* and `TextSnapshot.getCount()` (or any very large number) for *to*. To search a single character, pass a value of `from+1` for `to`.

Example

The following example illustrates how to use this method. To use this code, place a static text field containing the text “TextSnapshot Example” on the Stage.

```
// This example assumes that the movie clip contains
// the static text "TextSnapshot Example"
var my_mc:MovieClip = this;
var my_snap:TextSnapshot = my_mc.getTextSnapshot();
var count:Number = my_snap.getCount();
var theText:String = my_snap.getText(0, count, false);
trace(count); // output: 20
trace(theText); // output: TextSnapshot Example
```

See also

`TextSnapshot.getSelectedText(), TextSnapshot.getText()`

TextSnapshot.getSelectedText()

Availability

Authoring: Flash MX 2004.

Playback: SWF files published for Flash Player 6 or later, playing in Flash Player 7 or later.

Usage

my_snap.getSelectedText([*includeLineEndings:Boolean*]) : *String*

Parameters

includeLineEndings An optional Boolean value that specifies whether newline characters are inserted into the returned string where appropriate. The default value is *false*.

Returns

A string that contains all the characters specified by the corresponding TextSnapshot.setSelected() command.

Description

Method; returns a string that contains all the characters specified by the corresponding TextSnapshot.setSelected() command. If no characters are selected, an empty string is returned.

If you pass a value of `true` for *includeLineEndings*, newline characters are inserted in the string returned where deemed appropriate. In this case, the return string might be longer than the input range. If *includeLineEndings* is `false` or omitted, the selected text is returned without any characters added.

Example

The following example illustrates how to use this method. To use this code, place a static text field containing the text "TextSnapshot Example" on the Stage. Create a dynamic text field on the Stage, select it, and click the Character button in the Property inspector. Select Specify ranges and enter any character into the Include These Characters text field. For more information on this process, see TextSnapshot.setSelected(). Add the following ActionScript to Frame 1 of the Timeline.

```
// This example assumes that the movie clip contains
// the static text "TextSnapshot Example"
var my_snap:TextSnapshot = this.getTextSnapshot();
var count:Number = my_snap.getCount();
my_snap.setSelectColor(0xFF0000); // set the selection color to red
my_snap.setSelected(0, 4, true); // select the first four characters
my_snap.setSelected(1, 2, false); // deselect the second character (leaving
  the rest selected)

var firstCharIsSelected:Boolean = my_snap.getSelected(0, 1);
var secondCharIsSelected:Boolean = my_snap.getSelected(1, 2);
var theText:String = my_snap.getSelectedText(false); // get the selected text
trace(theText); // output: Txt
trace(firstCharIsSelected); // output: true
trace(secondCharIsSelected); // output: false
```

When you test the SWF file, you see a colored rectangle around the specified characters.

See also

`TextSnapshot.getSelected()`

TextSnapshot.getText()

Availability

Authoring: Flash MX 2004.

Playback: SWF files published for Flash Player 6 or later, playing in Flash Player 7 or later.

Usage

```
my_snap.getText(from:Number, to:Number [, includeLineEndings:Boolean ] ) :
  String
```

Parameters

from An integer that indicates the position of the first character of *my_snap* to be included in the returned string. Valid values for *from* are 0 through TextSnapshot.getCount() - 1. If *from* is a negative value, 0 is used.

to An integer that is 1+ the index of the last character in *my_snap* to be examined. Valid values for *to* are 0 through `TextSnapshot.getCount()`. The character indexed by the *to* parameter is not included in the extracted string. If this parameter is omitted, `TextSnapshot.getCount()` is used. If this value is less than or equal to the value of `from`, `from+1` is used.

includeLineEndings An optional Boolean value that specifies whether newline characters are inserted into the returned string where appropriate. The default value is *false*.

Returns

A string containing the characters in the specified range, or an empty string if no characters are found in the specified range.

Description

Method; returns a string that contains all the characters specified by the *from* and *to* parameters. If no characters are selected, an empty string is returned.

To return all characters, pass a value of 0 for *from* and `TextSnapshot.getCount()` (or any very large number) for *to*. To return a single character, pass a value of `from+1` for `to`.

If you pass a value of `true` for *includeLineEndings*, newline characters are inserted in the string returned where deemed appropriate. In this case, the return string might be longer than the input range. If *includeLineEndings* is `false` or omitted, the selected text is returned without any characters added.

Example

The following example illustrates how you can output the number of characters in a specified TextSnapshot object. To use this code, place a static text field containing the text "TextSnapshot Example" on the Stage.

```
// This example assumes that the movie clip contains
// the static text "TextSnapshot Example"
var my_mc:MovieClip = this;
var my_snap:TextSnapshot = my_mc.getTextSnapshot();
var count:Number = my_snap.getCount();
var theText:String = my_snap.getText(0, count, false);
```

```
trace(count); // output: 20
trace(theText); // output: TextSnapshot Example
```

See also

`TextSnapshot.getSelectedText()`

TextSnapshot.hitTestTextNearPos()

Availability

Authoring: Flash MX 2004.

Playback: SWF files published for Flash Player 6 or later, playing in Flash Player 7 or later.

Usage

```
my_snap.hitTestTextNearPos(x:Number, y:Number [, maxDistance:Number] ) :
  Number
```

Parameters

x A number that represents the x coordinate of the movie clip containing the text in *my_snap*.

y A number that represents the x coordinate of the movie clip containing the text in *my_snap*.

maxDistance An optional number that represents the maximum distance from *x*, *y* that can be searched for text. The distance is measured from the centerpoint of each character. The default value is 0.

Returns

An integer representing the index value of the character in *my_snap* that is nearest to the specified *x*, *y* coordinate. Returns -1 if no character is found, or if the font doesn't contain character metric information (see "Description").

Description

Method; lets you determine which character within a TextSnapshot object is on or near specified *x*, *y* coordinates of the movie clip containing the text in *my_snap*.

If you omit or pass a value of 0 for *maxDistance*, the location specified by the *x*, *y* coordinates must lie inside the bounding box of *my_snap*.

This method functions correctly only for fonts that include character metric information; by default, Flash does not include this information for static text fields. In some cases, this behavior means that the method returns -1 instead of an index value. To ensure that an index value is returned, you can force the Flash authoring tool to include the character metric information for a font. To do this, add a dynamic text field that uses that font, select Character Options for that dynamic text field, and then specify that font outlines should be embedded for at least one character. It doesn't matter which character(s) you specify, nor even if they are the characters used in the static text fields in question.

Example

The following example illustrates how to use this method. To use this code, place a static text field containing the text "TextSnapshot Example" on the Stage. To test the code, move the pointer over the static text field.

Note: If characters don't appear to be selected when you run the code, you must also place a dynamic text field on the Stage. See "Description" in this entry.

```
// this example assumes that the movie clip contains
// the static text "TextSnapshot Example"
```

```
var my_mc:MovieClip = this;
var my_ts:TextSnapshot = my_mc.getTextSnapshot();
my_mc.onMouseMove = function() {
  // find which character the mouse pointer is over (if any)
  var hitIndex:Number = my_ts.hitTestTextNearPos(_xmouse, _ymouse, 0);
  // deselect everything
  my_ts.setSelected(0, my_ts.getCount(), false);
  if (hitIndex>=0) {
    // select the single character the mouse pointer is over
    my_ts.setSelected(hitIndex, hitIndex+1, true);
  }
};
```

See also

`MovieClip.getTextSnapshot()`, `MovieClip._x`, `MovieClip._y`

TextSnapshot.setSelectColor()

Availability

Authoring: Flash MX 2004.

Playback: SWF files published for Flash Player 6 or later, playing in Flash Player 7 or later.

Usage

```
my_snap.setSelectColor(hexColor:Number);
```

Parameters

`hexColor` The color used for the border placed around characters that have been selected by the corresponding TextSnapshot.setSelected() command, expressed in `0xRRGGBB` format.

Returns

Nothing.

Description

Method; specifies the color to use when highlighting characters that have been selected with the TextSnapshot.setSelected() command. The color is always opaque; you can't specify a transparency value.

This method functions correctly only for fonts that include character metric information; by default, Flash does not include this information for static text fields. In some cases, this behavior means that the method returns -1 instead of an index value. To ensure that an index value is returned, you can force the Flash authoring tool to include the character metric information for a font. To do this, add a dynamic text field that uses that font, select Character Options for that dynamic text field, and then specify that font outlines should be embedded for at least one character. It doesn't matter which character(s) you specify, nor even if they are the characters used in the static text fields in question.

Example

The following example illustrates how to use this method. To use this code, place a static text field containing the text "TextSnapshot Example" on the Stage. Add the following ActionScript to your AS or FLA file.

Note: If characters don't appear to be selected when you run the code, you must also place a dynamic text field on the Stage. See "Description" in this entry.

```
// This example assumes that the movie clip contains
// the static text "TextSnapshot Example"
var my_snap:TextSnapshot = this.getTextSnapshot();
var count:Number = my_snap.getCount();
my_snap.setSelectColor(0xFF0000); // Set the selection color to red.
my_snap.setSelected(0, 4, true); // Select the first four characters.
my_snap.setSelected(1, 2, false); // deselect the second character (leaving
  the rest selected)

var firstCharIsSelected:Boolean = my_snap.getSelected(0, 1);
var secondCharIsSelected:Boolean = my_snap.getSelected(1, 2);
var theText:String = my_snap.getSelectedText(false); // get the selected text
```

```
trace(theText); // output: Txt
trace(firstCharIsSelected); // output: true
trace(secondCharIsSelected); // output: false
```

When you test the SWF file, you see a colored rectangle around the specified characters.

TextSnapshot.setSelected()

Availability

Authoring: Flash MX 2004.

Playback: SWF files published for Flash Player 6 or later, playing in Flash Player 7 or later.

Usage

```
my_snap.setSelected(from:Number, to:Number, select:Boolean) : Void
```

Parameters

`from` An integer that indicates the position of the first character of `my_snap` to select. Valid values for `from` are 0 through TextSnapshot.getCount() - 1. If `from` is a negative value, 0 is used.

`to` An integer that is 1+ the index of the last character in `my_snap` to be examined. Valid values for `to` are 0 through `TextSnapshot.getCount()`. The character indexed by the `to` parameter is not included in the extracted string. If this parameter is omitted, `TextSnapshot.getCount()` is used. If this value is less than or equal to the value of `from`, `from+1` is used.

`select` A Boolean value that specifies whether the text should be selected (`true`) or deselected (`false`).

Returns

Nothing.

Description

Method; specifies a range of characters in a TextSnapshot object to be selected or deselected. Characters that are selected are drawn with a colored rectangle behind them, matching the bounding box of the character. The color of the bounding box is defined by TextSnapshot.setSelectColor().

To select or deselect all characters, pass a value of 0 for `from` and `TextSnapshot.getCount()` (or any very large number) for `to`. To specify a single character, pass a value of `from+1` for `to`.

Because characters are individually marked as selected, you can issue this command multiple times to select multiple characters; that is, using this command does not deselect other characters that have been set by this command.

The colored rectangle that indicates a selection is displayed only for fonts that include character metric information; by default, Flash does not include this information for static text fields. In some cases, this behavior means that text that is selected won't appear to be selected onscreen. To ensure that all selected text appears to be selected, you can force the Flash authoring tool to include the character metric information for a font. To do this, add a dynamic text field that uses that font, select Character Options for that dynamic text field, and then specify that font outlines should be embedded for at least one character. It doesn't matter which characters you specify, nor even if they are the characters used in the static text fields in question.

Example

The following example illustrates how to use this method. To use this code, place a static text field containing the text "TextSnapshot Example" on the Stage.

Note: If characters don't appear to be selected when you run the code, you must also place a dynamic text field on the Stage. See "Description" in this entry.

```
// This example assumes that the movie clip contains
// the static text "TextSnapshot Example"
var my_snap:TextSnapshot = this.getTextSnapshot();
var count:Number = my_snap.getCount();
my_snap.setSelectColor(0xFF0000); // Set the selection color to red.
my_snap.setSelected(0, 4, true); // Select the first four characters.
my_snap.setSelected(1, 2, false); // Deselect the second character (leaving
  the rest selected).

var firstCharIsSelected:Boolean = my_snap.getSelected(0, 1);
var secondCharIsSelected:Boolean = my_snap.getSelected(1, 2);
var theText:String = my_snap.getSelectedText(false); // get the selected text
trace(theText); // output: Txt
trace(firstCharIsSelected); // output: true
trace(secondCharIsSelected); // output: false
```

this

Availability

Flash Player 5.

Usage

```
this
```

Description

Identifier; references an object or movie clip instance. When a script executes, `this` references the movie clip instance that contains the script. When a method is called, `this` contains a reference to the object that contains the called method.

Inside an `on()` event handler attached to a button, `this` refers to the Timeline that contains the button. Inside an `onClipEvent()` event handler attached to a movie clip, `this` refers to the Timeline of the movie clip itself.

Because `this` is evaluated in the context of the script that contains it, you can't use `this` in a script to refer to a variable defined in a class file. Create ApplyThis.as, and enter the following code:

```
class ApplyThis {
  var str:String = "Defined in ApplyThis.as";
  function conctStr(x:String):String {
    return x+x;
  }
  function addStr():String {
    return str;
  }
}
```

Then, in a FLA or AS file, add the following ActionScript:

```
var obj:ApplyThis = new ApplyThis();
var abj:ApplyThis = new ApplyThis();
abj.str = "defined in FLA or AS";
trace(obj.addStr.call(abj, null));  //output: defined in FLA or AS
trace(obj.addStr.call(this, null));  //output: undefined
trace(obj.addStr.call(obj, null));  //output: Defined in applyThis.as
```

Similarly, to call a function defined in a dynamic class, you must use `this` to invoke the function in the proper scope:

```
// incorrect version of Simple.as
/*
dynamic class Simple {
  function callfunc() {
    trace(func());
  }
}
*/
// correct version of Simple.as
dynamic class simple {
  function callfunc() {
    trace(this.func());
  }
}
```

Inside the FLA or AS file, add the following ActionScript:

```
var obj:Simple = new Simple();
obj.num = 0;
obj.func = function() {
  return true;
};
obj.callfunc();
// output: true
```

You get a syntax error when you use the incorrect version of Simple.as.

Example

In the following example, the keyword `this` references the Circle object:

```
function Circle(radius:Number):Void {
  this.radius = radius;
  this.area = Math.PI*Math.pow(radius, 2);
}
var myCircle = new Circle(4);
trace(myCircle.area);
```

In the following statement assigned to a frame inside a movie clip, the keyword `this` references the current movie clip.

```
// sets the alpha property of the current movie clip to 20
this._alpha = 20;
```

In the following statement inside a MovieClip.onPress handler, the keyword `this` references the current movie clip:

```
this.square_mc.onPress = function() {
  startDrag(this);
};
this.square_mc.onRelease = function() {
  stopDrag();
};
```

See also

`on()`, `onClipEvent()`

throw

Availability

Flash Player 7.

Usage

```
throw expression
```

Description

Statement; generates, or *throws*, an error that can be handled, or *caught*, by a `catch{}` code block. If an exception is not caught by a `catch` block, the string representation of the thrown value is sent to the Output panel.

Typically, you throw instances of the Error class or its subclasses (see the Example section).

Parameters

`expression` An ActionScript expression or object.

Example

In this example, a function named `checkEmail()` checks whether the string that is passed to it is a properly formatted e-mail address. If the string does not contain an @ symbol, the function throws an error.

```
function checkEmail(email:String) {
  if (email.indexOf("@") == -1) {
    throw new Error("Invalid email address");
  }
}
checkEmail(“someuser_theirdomain.com”);
```

The following code then calls the `checkEmail()` function within a `try` code block. If the `email_txt` string does not contain a valid e-mail address, the error message appears in a text field (`error_txt`).

```
try {
  checkEmail("Joe Smith”);
} catch (e) {
  error_txt.text = e.toString();
}
```

In the following example, a subclass of the Error class is thrown. The `checkEmail()` function is modified to throw an instance of that subclass. (For more information, see “Creating subclasses” in *Using ActionScript in Flash*.)

```
// Define Error subclass InvalidEmailError
// In InvalidEmailError.as:
class InvalidEmailAddress extends Error {
  var message = "Invalid email address.";
}
```

In a FLA or AS file, enter the following ActionScript in Frame 1 of the Timeline:

```
import InvalidEmailAddress;
function checkEmail(email:String) {
  if (email.indexOf("@") == -1) {
    throw new InvalidEmailAddress();
  }
}
try {
  checkEmail("Joe Smith");
} catch (e) {
  this.createTextField("error_txt", this.getNextHighestDepth(), 0, 0, 100,
  22);
  error_txt.autoSize = true;
  error_txt.text = e.toString();
}
```

See also

Error class, `try..catch..finally`

trace()

Availability

Flash Player 4.

Usage

`trace(expression)`

Parameters

expression An expression to evaluate. When a SWF file is opened in the Flash authoring tool (using the Test Movie command), the value of the *expression* parameter is displayed in the Output panel.

Returns

Nothing.

Description

Statement; evaluates the expression and displays the result in the Output panel in test mode.

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

You can use the Omit Trace Actions command in the Publish Settings dialog box to remove `trace()` actions from the exported SWF file.

Example

The following example uses a `trace()` statement to display in the Output panel the methods and properties of the dynamically created text field called `error_txt`:

```
this.createTextField("error_txt", this.getNextHighestDepth(), 0, 0, 100, 22);
for (var i in error_txt) {
  trace("error_txt."+i+" = "+error_txt[i]);
}
/* output:
  error_txt.styleSheet = undefined
  error_txt.mouseWheelEnabled = true
  error_txt.condenseWhite = false
  ...
  error_txt.maxscroll = 1
  error_txt.scroll = 1
*/
```

true

Availability

Flash Player 5.

Usage

```
true
```

Description

Constant; a unique Boolean value that represents the opposite of `false`. When automatic data typing converts `true` to a number, it becomes 1; when it converts `true` to a string, it becomes `"true"`.

Example

The following example shows the use of `true` in an `if` statement:

```
var shouldExecute:Boolean;
/* ...
code that sets shouldExecute to either true or false goes here
shouldExecute is set to true for this example
*/
shouldExecute = true;

if (shouldExecute == true) {
  trace("your statements here");
}
/*
true is also implied, so the if statement could also be written:
if (shouldExecute) {
  trace("your statements here");
}
*/
```

The following example shows how automatic data typing converts `true` to the number 1:

```
var myNum:Number;
myNum = 1 + true;
trace(myNum); // output: 2
```

See also

`false`, `Boolean class`

try..catch..finally

Availability

Flash Player 7.

Usage

```
try {
  // ... try block ...
} finally {
  // ... finally block ...
}
try {
  // ... try block ...
} catch(error[:ErrorType1]) {
  // ... catch block ...
} [catch(error[:ErrorTypeN]) {
  // ... catch block ...
}] [finally {
  // ... finally block ...
}]
```

Parameters

error The expression thrown from a *throw* statement, typically an instance of the Error class or one of its subclasses.

ErrorType An optional type specifier for the *error* identifier. The `catch` clause catches only errors of the specified type.

Description

Keywords; enclose a block of code in which an error can occur, and then respond to the error. If any code within the `try` code block throws an error (using the `throw` statement), control passes to the `catch` block, if one exists, and then to the `finally` code block, if one exists. The `finally` block always executes, regardless of whether an error was thrown. If code within the `try` block doesn't throw an error (that is, if the `try` block completes normally), then the code in the `finally` block is still executed. The `finally` block executes even if the `try` block exits using a `return` statement.

A `try` block must be followed by a `catch` block, a `finally` block, or both. A single `try` block can have multiple `catch` blocks but only one `finally` block. You can nest `try` blocks as many levels deep as desired.

The *error* parameter specified in a `catch` handler must be a simple identifier such as `e` or `theException` or `x`. The variable in a `catch` handler can also be *typed*. When used with multiple `catch` blocks, typed errors let you catch multiple types of errors thrown from a single `try` block.

If the exception thrown is an object, the type will match if the thrown object is a subclass of the specified type. If an error of a specific type is thrown, the `catch` block that handles the corresponding error is executed. If an exception that is not of the specified type is thrown, the `catch` block does not execute and the exception is automatically thrown out of the `try` block to a `catch` handler that matches it.

If an error is thrown within a function, and the function does not include a `catch` handler, then the ActionScript interpreter exits that function, as well as any caller functions, until a `catch` block is found. During this process, `finally` handlers are called at all levels.

Example

The following example shows how to create a `try..finally` statement. Because code in the `finally` block is guaranteed to execute, it is typically used to perform any necessary clean-up after a `try` block executes. In the following example, `setInterval()` calls a function every 1000 millisecond (1 second). If an error occurs, an error is thrown and is caught by the `catch` block. The finally block is always executed whether or not an error occurs. Because `setInterval()` is used, `clearInterval()` must be placed in the `finally` block to ensure that the interval is cleared from memory.

```
myFunction = function () {
  trace("this is myFunction");
};
try {
  myInterval = setInterval(this, "myFunction", 1000);
  throw new Error("my error");
} catch (myError:Error) {
  trace("error caught: "+myError);
} finally {
  clearInterval(myInterval);
  trace("error is cleared");
}
```

In the following example, the `finally` block is used to delete an ActionScript object, regardless of whether an error occurred. Create a new AS file called Account.as.

```
class Account {
  var balance:Number = 1000;
  function getAccountInfo():Number {
    return (Math.round(Math.random()*10)%2);
  }
}
```

In the same directory as Account.as, create a new AS or FLA document and enter the following ActionScript in Frame 1 of the Timeline:

```
import Account;
var account:Account = new Account();
try {
  var returnVal = account.getAccountInfo();
  if (returnVal != 0) {
    throw new Error("Error getting account information.");
  }
} finally {
  if (account != null) {
    delete account;
  }
}
```

The following example demonstrates a `try..catch` statement. The code within the `try` block is executed. If an exception is thrown by any code within the `try` block, control passes to the `catch` block, which shows the error message in a text field using the `Error.toString()` method.

In the same directory as Account.as, create a new FLA document and enter the following ActionScript in Frame 1 of the Timeline:

```
import Account;
var account:Account = new Account();
try {
  var returnVal = account.getAccountInfo();
  if (returnVal != 0) {
    throw new Error("Error getting account information.");
  }
  trace("success");
} catch (e) {
  this.createTextField("status_txt", this.getNextHighestDepth(), 0, 0, 100,
  22);
  status_txt.autoSize = true;
  status_txt.text = e.toString();
}
```

The following example shows a `try` code block with multiple, typed `catch` code blocks. Depending on the type of error that occurred, the `try` code block throws a different type of object. In this case, `myRecordSet` is an instance of a (hypothetical) class named RecordSet whose `sortRows()` method can throw two types of errors, RecordSetException and MalformedRecord.

In the following example, the RecordSetException and MalformedRecord objects are subclasses of the Error class. Each is defined in its own AS class file. (For more information, see "Creating Custom Classes with ActionScript 2.0" in *Using ActionScript in Flash*.)

```
// In RecordSetException.as:
class RecordSetException extends Error {
  var message = "Record set exception occurred.";
}
// In MalformedRecord.as:
class MalformedRecord extends Error {
  var message = "Malformed record exception occurred.";
}
```

Within the RecordSet class's `sortRows()` method, one of these previously defined error objects is thrown, depending on the type of exception that occurred. The following example shows how this code might look:

```
class RecordSet {
  function sortRows() {
    var returnVal:Number = randomNum();
    if (returnVal == 1) {
      throw new RecordSetException();
    } else if (returnVal == 2) {
      throw new MalformedRecord();
    }
  }
  function randomNum():Number {
    return Math.round(Math.random()*10)%3;
    }
}
```

Finally, in another AS file or FLA script, the following code invokes the `sortRows()` method on an instance of the RecordSet class. It defines `catch` blocks for each type of error that is thrown by `sortRows()`.

```
import RecordSet;
var myRecordSet:RecordSet = new RecordSet();
try {
  myRecordSet.sortRows();
  trace("everything is fine");
} catch (e:RecordSetException) {
  trace(e.toString());
} catch (e:MalformedRecord) {
  trace(e.toString());
}
```

See also

Error class, `throw`, `class`, `extends`

typeof

Availability

Flash Player 5.

Usage

```
typeof(expression) : String
```

Parameters

`expression` A string, movie clip, button, object, or function.

Description

Operator; a unary operator placed before a single parameter. The `typeof` operator causes the Flash interpreter to evaluate *`expression`*; the result is a string specifying whether the expression is a string, movie clip, object, function, number, or Boolean value. The following table shows the results of the `typeof` operator on each type of expression:

Parameter	Output
String	`string`
Movie clip	`movieclip`
Button	`object`
Text field	`object`
Number	`number`
Boolean	`boolean`
Object	`object`
Function	`function`
null	`null`
undefined	`undefined`

Example

In the following example, all instances in a SWF file and their types are traced and displayed in the Output panel.

```
for (i in _root) {
  trace("_root."+i+" ("+typeof (_root[i])+")");
}
```

undefined

Availability

Flash Player 5.

Usage

`undefined`

Parameters

None.

Returns

Nothing.

Description

A special value, usually used to indicate that a variable has not yet been assigned a value. A reference to an undefined value returns the special value `undefined`. The ActionScript code `typeof(undefined)` returns the string `"undefined"`. The only value of type `undefined` is `undefined`.

In files published for Flash Player 6 or earlier, the value of `undefined.toString()` is "" (an empty string). In files published for Flash Player 7 or later, the value of `undefined.toString()` is `undefined`.

The value `undefined` is similar to the special value `null`. When `null` and `undefined` are compared with the equality (==) operator, they compare as equal. However, when `null` and `undefined` are compared with the strict equality (===) operator, they compare as not equal.

Example

In the following example, the variable `x` has not been declared and therefore has the value `undefined`. In the first section of code, the equality operator (==) compares the value of `x` to the value `undefined`, and the appropriate result is sent to the Output panel.In the second section of code, the equality (==) operator compares the values `null` and `undefined`.

```
// x has not been declared
trace("The value of x is "+x);
if (x == undefined) {
  trace("x is undefined");
} else {
  trace("x is not undefined");
}
trace("typeof (x) is "+typeof (x));
if (null == undefined) {
  trace("null and undefined are equal");
} else {
  trace("null and undefined are not equal");
}
```

The following result is displayed in the Output panel.

```
The value of x is undefined
x is undefined
typeof (x) is undefined
null and undefined are equal
```

unescape()

Availability

Flash Player 5.

Usage

```
unescape(x:String) : String
```

Parameters

x A string with hexadecimal sequences to escape.

Returns

A string decoded from a URL-encoded parameter.

Description

Function; evaluates the parameter *x* as a string, decodes the string from URL-encoded format (converting all hexadecimal sequences to ASCII characters), and returns the string.

Example

The following example shows the escape-to-unescape conversion process:

```
var email:String = "user@somedomain.com";
trace(email);
var escapedEmail:String = escape(email);
trace(escapedEmail);
var unescapedEmail:String = unescape(escapedEmail);
trace(unescapedEmail);
```

The following result is displayed in the Output panel.

```
user@somedomain.com
user%40somedomain%2Ecom
user@somedomain.com
```

unloadMovie()

Availability

Flash Player 3.

Usage

```
unloadMovie(target:MovieClip) : Void
unloadMovie(target:String) : Void
```

Parameters

`target`	The target path of a movie clip.

Returns

None.

Description

Function; removes a movie clip that was loaded by means of `loadMovie()` from Flash Player. To unload a movie clip that was loaded by means of `loadMovieNum()`, use `unloadMovieNum()` instead of `unloadMovie()`.

Example

The following example creates a new movie clip called `pic_mc` and loads an image into that clip. It is loaded using the MovieClipLoader class. When you click the image, the movie clip unloads from the SWF file:

```
var pic_mcl:MovieClipLoader = new MovieClipLoader();
pic_mcl.loadClip("http://www.macromedia.com/devnet/mx/blueprint/articles/
  performance/spotlight_speterson.jpg", this.createEmptyMovieClip("pic_mc",
  this.getNextHighestDepth()));
var listenerObject:Object = new Object();
listenerObject.onLoadInit = function(target_mc) {
  target_mc.onRelease = function() {
    unloadMovie(pic_mc);
    /* or you could use the following, which refers to the movie clip
  referenced by 'target_mc'. */
    //unloadMovie(this);
  };
};
pic_mcl.addListener(listenerObject);
```

See also

`MovieClip.loadMovie()`, `MovieClipLoader.unloadClip()`

unloadMovieNum()

Availability

Flash Player 3.

Usage

`unloadMovieNum(`*`level:Number`*`) : `*`Void`*

Parameters

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

Returns

Nothing.

Description

Function; removes a SWF or image that was loaded by means of loadMovieNum() from Flash Player. To unload a SWF or image that was loaded with MovieClip.loadMovie(), use `unloadMovie()` instead of `unloadMovieNum()`.

Example

The following example loads an image into a SWF file. When you click `unload_btn`, the loaded content is removed.

```
loadMovieNum("yourimage.jpg", 1);
unload_btn.onRelease = function() {
  unloadMovieNum(1);
}
```

See also

`MovieClip.loadMovie()`, `loadMovieNum()`, `unloadMovie()`

updateAfterEvent()

Availability

Flash Player 5.

Usage

```
updateAfterEvent() : Void
```

Parameters

None.

Returns

Nothing.

Description

Function; updates the display (independent of the frames per second set for the movie) when you call it within an `onClipEvent()` handler or as part of a function or method that you pass to setInterval(). Flash ignores calls to `updateAfterEvent` that are not within an `onClipEvent()` handler or part of a function or method passed to setInterval(). This function works only with certain Mouse and MovieClip handlers: the mouseDown, mouseUp, mouseMove, keyDown and keyUp handlers for the Mouse class; the onMouseMove, onMouseDown, onMouseUp, onKeyDown, and onKeyUp handlers for the MovieClip class. It does not work with the Key class.

Example

The following example show how to create a custom cursor called `cursor_mc`. ActionScript is used to replace the mouse cursor with `cursor_mc`. Then `updateAfterEvent()` is used to continually refresh the Stage to make the cursor's movement appear smooth.

```
Mouse.hide();
cursor_mc.onMouseMove = function() {
  this._x = this._parent._xmouse;
  this._y = this._parent._ymouse;
  updateAfterEvent();
};
```

See also

`onClipEvent()`, `setInterval()`

var

Availability

Flash Player 5.

Usage

```
var variableName [= value1] [...,variableNameN [=valueN]]
```

Parameters

`variableName` An identifier.

`value` The value assigned to the variable.

Returns

Nothing.

Description

Statement; used to declare local or Timeline variables.

If you declare variables inside a function, the variables are local. They are defined for the function and expire at the end of the function call. More specifically, a variable defined using `var` is local to the code block containing it. Code blocks are demarcated by curly braces ({}).

You cannot declare a variable scoped to another object as a local variable (for more information, see "Scoping and declaring variables" in *Using Actionscript in Flash*):

```
my_array.length = 25;     // ok
var my_array.length = 25; // syntax error
```

When you use `var`, you can strictly type the variable. For more information, see "Strict data typing" in *Using ActionScript in Flash*.

Note: You must also use `var` when declaring properties inside class definitions in external scripts. Class files also support public, private, and static variable scopes. See "Creating Custom Classes with ActionScript 2.0" in *Using ActionScript in Flash,* and see private, public, and static.

Example

The following ActionScript creates a new array of product names. `Array.push` adds an element onto the end of the array. If you want to use strict typing, it is essential that you use the `var` keyword. Without `var` before `product_array`, you get errors when you try to use strict typing.

```
var product_array:Array = new Array("MX 2004", "Studio", "Dreamweaver",
  "Flash", "ColdFusion", "Contribute", "Breeze");
product_array.push("Flex");
trace(product_array);
// output: MX 2004,Studio,Dreamweaver,Flash,ColdFusion,Contribute,Breeze,Flex
```

Video class

Availability

Flash Player 6; the ability to play Flash Video (FLV) files was added in Flash Player 7.

Description

The Video class lets you display live streaming video on the Stage without embedding it in your SWF file. You capture the video by using `Camera.get()`. In files published for Flash Player 7 and later, you can also use the Video class to play back Flash Video (FLV) files over HTTP or from the local file system. For more information, see the NetConnection class and NetStream class entries.For more information, see "Playing back external FLV files dynamically" on page 299 and the NetConnection class and NetStream class entries.

A Video object can be used like a movie clip. As with other objects you place on the Stage, you can control various properties of Video objects. For example, you can move the Video object around on the Stage by using its `_x` and `_y` properties, you can change its size using its `_height` and `_width` properties, and so on.

To display the video stream, first place a Video object on the Stage. Then use `Video.attachVideo()` to attach the video stream to the Video object.

To place a Video object on the Stage:

1. If the Library panel isn't visible, select Window > Library to display it.
2. Add an embedded Video object to the library by clicking the Options menu on the right side of the Library panel title bar and selecting New Video.
3. Drag the Video object to the Stage and use the Property inspector to give it a unique instance name, such as `my_video`. (Do not name it Video.)

Method summary for the Video class

Method	Description
`Video.attachVideo()`	Specifies a video stream to be displayed within the boundaries of the Video object on the Stage.
`Video.clear()`	Clears the image currently displayed in the Video object.

Property summary for the Video class

Property	Description
`Video.deblocking`	Specifies the behavior for the deblocking filter that the video compressor applies as needed when streaming the video.
`Video.height`	Read-only; the height of the video stream, in pixels.
`Video.smoothing`	Specifies whether the video should be smoothed (interpolated) when it is scaled.
`Video.width`	Read-only; the width of the video stream, in pixels.

Video.attachVideo()

Availability

Flash Player 6; the ability to work with Flash Video (FLV) files was added in Flash Player 7.

Usage

```
my_video.attachVideo(source:Object) : Void
```

Parameters

`source` A Camera object that is capturing video data or a NetStream object. To drop the connection to the Video object, pass `null` for *`source`*.

Returns

Nothing.

Description

Method; specifies a video stream (*`source`*) to be displayed within the boundaries of the Video object on the Stage. The video stream is either an FLV file being displayed by means of the `NetStream.play()` command, a Camera object, or `null`. If *`source`* is `null`, video is no longer played within the Video object.

You don't have to use this method if the FLV file contains only audio; the audio portion of an FLV files is played automatically when the `NetStream.play()` command is issued.

If you want to control the audio associated with an FLV file, you can use `MovieClip.attachAudio()` to route the audio to a movie clip; you can then create a Sound object to control some aspects of the audio. For more information, see `MovieClip.attachAudio()`.

Example

The following example plays live video locally:

```
var my_video:Video; //my_video is a Video object on the Stage
var active_cam:Camera = Camera.get();
my_video.attachVideo(active_cam);
```

The following example plays a previously recorded file named myVideo.flv that is stored in the same directory as the SWF file.

```
var my_video:Video; // my_video is a Video object on the Stage
var my_nc:NetConnection = new NetConnection();
my_nc.connect(null);
var my_ns:NetStream = new NetStream(my_nc);
my_video.attachVideo(my_ns);
my_ns.play("video1.flv");
```

See also

Camera class, NetStream class

Video.clear()

Availability

Flash Player 6.

Usage

`*my_video*.clear() : *Void*`

Parameters

None.

Returns

Nothing.

Description

Method; clears the image currently displayed in the Video object. This is useful when, for example, you want to display standby information without having to hide the Video object.

Example

The following example pauses and clears video1.flv that is playing in a Video object (called `my_video`) when the user clicks the `pause_btn` instance.

```
var pause_btn:Button;
var my_video:Video; // my_video is a Video object on the Stage
var my_nc:NetConnection = new NetConnection();
my_nc.connect(null);
var my_ns:NetStream = new NetStream(my_nc);
my_video.attachVideo(my_ns);
my_ns.play("video1.flv");
pause_btn.onRelease = function() {
  my_ns.pause();
  my_video.clear();
};
```

See also

`Video.attachVideo()`

Video.deblocking

Availability

Flash Player 6.

Usage

my_video.deblocking:*Number*

Description

Property; a number that specifies the behavior for the deblocking filter that the video compressor applies as needed when streaming the video. The following are acceptable values:

- 0 (the default): Let the video compressor apply the deblocking filter as needed.
- 1: Never use the deblocking filter.
- 2: Always use the deblocking filter.

The deblocking filter has an effect on overall playback performance, and it is usually not necessary for high-bandwidth video. If your system is not powerful enough, you might experience difficulties playing back video with this filter enabled.

Example

The following example plays video1.flv in the `my_video` video object, and lets the user change the deblocking filter behavior on video1.flv. Add a video object called `my_video` and a ComboBox instance called `deblocking_cb` to your file, and then add the following ActionScript to your FLA or AS file.

```
var deblocking_cb:mx.controls.ComboBox;
var my_video:Video; // my_video is a Video object on the Stage
var my_nc:NetConnection = new NetConnection();
my_nc.connect(null);
var my_ns:NetStream = new NetStream(my_nc);
my_video.attachVideo(my_ns);
my_ns.play("video1.flv");

deblocking_cb.addItem({data:0, label:'Auto'});
deblocking_cb.addItem({data:1, label:'No'});
deblocking_cb.addItem({data:2, label:'Yes'});

var cbListener:Object = new Object();
cbListener.change = function(evt:Object) {
  my_video.deblocking = evt.target.selectedItem.data;
};
deblocking_cb.addEventListener("change", cbListener);
```

Use the ComboBox instance to change the deblocking filter behavior on video1.flv.

Video.height

Availability

Flash Player 6.

Usage

```
my_video.height:Number
```

Description

Property (read-only); an integer specifying the height of the video stream, in pixels. For live streams, this value is the same as the `Camera.height` property of the Camera object that is capturing the video stream. For FLV files, this value is the height of the file that was exported as FLV.

You may want to use this property, for example, to ensure that the user is seeing the video at the same size at which it was captured, regardless of the actual size of the Video object on the Stage.

Example

Usage 1: The following example sets the height and width values of the Video object to match the values of an FLV file. You should call this code after `NetStream.onStatus` is invoked with a `code` property of `NetStream.Buffer.Full`. If you call it when the `code` property is `NetStream.Play.Start`, the height and width values will be 0, because the Video object doesn't yet have the height and width of the loaded FLV file.

```
var my_nc:NetConnection = new NetConnection();
my_nc.connect(null);
var my_ns:NetStream = new NetStream(my_nc);
my_mc.my_video.attachVideo(my_ns);
my_ns.play("video1.flv");

my_ns.onStatus = function(infoObject:Object) {
  switch (infoObject.code) {
  case 'NetStream.Buffer.Full' :
    my_mc._width = my_mc.my_video.width;
    my_mc._height = my_mc.my_video.height;
    break;
  }
};
```

Usage 2: The following example lets the user press a button to set the height and width of a video stream being displayed in the Flash Player to be the same as the height and width at which the video stream was captured.

```
var my_nc:NetConnection = new NetConnection();
my_nc.connect(null);
var my_ns:NetStream = new NetStream(my_nc);
my_mc.my_video.attachVideo(my_ns);
my_ns.play("video1.flv");

resize_btn.onRelease = function() {
  my_mc._width = my_mc.my_video.width;
```

```
    my_mc._height = my_mc.my_video.height;
};
```

See also

`MovieClip._height`, `Video.width`

Video.smoothing

Availability

Flash Player 6.

Usage

my_video.smoothing*:Boolean*

Description

Property; a Boolean value that specifies whether the video should be smoothed (interpolated) when it is scaled. For smoothing to work, the player must be in high-quality mode. The default value is `false` (no smoothing).

Example

The following example uses a button (called `smoothing_btn`) to toggle the smoothing property that is applied to the video `my_video` when it plays in a SWF file. Create a button called `smoothing_btn` and add the following ActionScript to your FLA or AS file:

```
this.createTextField("smoothing_txt", this.getNextHighestDepth(), 0, 0, 100,
  22);
smoothing_txt.autoSize = true;

var my_nc:NetConnection = new NetConnection();
my_nc.connect(null);
var my_ns:NetStream = new NetStream(my_nc);
my_video.attachVideo(my_ns);
my_ns.play("video1.flv");
my_ns.onStatus = function(infoObject:Object) {
  updateSmoothing();
};
smoothing_btn.onRelease = function() {
  my_video.smoothing = !my_video.smoothing;
  updateSmoothing();
};
function updateSmoothing():Void {
  smoothing_txt.text = "smoothing = "+my_video.smoothing;
}
```

Video.width

Availability

Flash Player 6.

Usage

```
my_video.width:Number
```

Description

Property (read-only); an integer specifying the width of the video stream, in pixels. For live streams, this value is the same as the `Camera.width` property of the Camera object that is capturing the video stream. For FLV files, this value is the width of the file that was exported as an FLV file.

You may want to use this property, for example, to ensure that the user is seeing the video at the same size at which it was captured, regardless of the actual size of the Video object on the Stage.

Example

See the examples for Video.height.

void

Availability

Flash Player 5.

Usage

```
void (expression)
function functionName():Void {}
```

Description

Usage 1: Operator; a unary operator that discards the `expression` value and returns an undefined value. The `void` operator is often used in comparisons using the equality (==) operator to test for undefined values.

Usage 2: Data type; used in function definitions to indicate that a function will not return a value.

Example

Usage 1: In the following ActionScript, the `void` operator is used to test for undefined values:

```
if (someUndefinedVariable == void (0)) {
  trace("someUndefinedVariable is undefined");
}
```

The previous code can also be written in the following way:

```
if (someUndefinedVariable == undefined) {
  trace("someUndefinedVariable is undefined");
}
```

Usage 2: In the following example, a function that returns a value is defined using the `Void` return type, which results in a compile-time error:

```
function myFunction():Void {
  return "This will cause a compile-time error.";
}

/* the following function call will generate a compile-time error:
**Error** Scene=Scene 1, layer=Layer 1, frame=1:Line 2: A function with return
  type Void may not return a value.
     return "This will cause a compile-time error.";
*/
myFunction();
```

while

Availability

Flash Player 4.

Usage

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

Parameters

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

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

Returns

Nothing.

Description

Statement; evaluates a condition and if the condition evaluates to `true`, runs a statement or series of statements before looping back to evaluate the condition again. After the condition evaluates to `false`, the statement or series of statements is skipped and the loop ends.

The `while` statement performs the following series of steps. Each repetition of steps 1 through 4 is called an *iteration* of the loop. The *condition* is retested at the beginning of each iteration, as shown in the following steps:

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

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

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

The curly braces (`{}`) used to enclose the block of statements to be executed by the `while` statement are not necessary if only one statement will execute.

Example

In the following example, the `while` statement is used to test an expression. When the value of `i` is less than 20, the value of `i` is traced. When the condition is no longer `true`, the loop exits.

```
var i:Number = 0;
while (i<20) {
  trace(i);
```

```
    i += 3;
}
```

The following result is displayed in the Output panel.

```
0
3
6
9
12
15
18
```

See also

`do while`, `continue`, `for`, `for..in`

with

Availability

Flash Player 5.

Usage

```
with (object:Object) {
  statement(s);
}
```

Parameters

object An instance of an ActionScript object or movie clip.

statement(s) An action or group of actions enclosed in curly braces ({}).

Returns

Nothing.

Description

Statement; lets you specify an object (such as a movie clip) with the *object* parameter and evaluate expressions and actions inside that object with the *statement(s)* parameter. This prevents you from having to repeatedly write the object's name or the path to the object.

The *object* parameter becomes the context in which the properties, variables, and functions in the *statement(s)* parameter are read. For example, if *object* is `my_array`, and two of the properties specified are `length` and `concat`, those properties are automatically read as `my_array.length` and `my_array.concat`. In another example, if *object* is `state.california`, any actions or statements inside the `with` statement are called from inside the `california` instance.

To find the value of an identifier in the *statement(s)* parameter, ActionScript starts at the beginning of the scope chain specified by the *object* and searches for the identifier at each level of the scope chain, in a specific order.

The scope chain used by the `with` statement to resolve identifiers starts with the first item in the following list and continues to the last item:

- The object specified in the *object* parameter in the innermost `with` statement.
- The object specified in the *object* parameter in the outermost `with` statement.
- The Activation object. (A temporary object that is automatically created when a function is called that holds the local variables called in the function.)
- The movie clip that contains the currently executing script.
- The Global object (built-in objects such as Math and String).

To set a variable inside a `with` statement, you must have declared the variable outside the `with` statement, or you must enter the full path to the Timeline on which you want the variable to live. If you set a variable in a `with` statement without declaring it, the `with` statement will look for the value according to the scope chain. If the variable doesn't already exist, the new value will be set on the Timeline from which the `with` statement was called.

Instead of using `with()`, you can use direct paths. If you find that paths are long and cumbersome to type, you can create a local variable and store the path in the variable, which you can then reuse in your code, as shown in the following ActionScript:

```
var shortcut = this._parent._parent.name_txt;
shortcut.text = "Hank";
shortcut.autoSize = true;
```

For more information on best practices in writing code and code readability, see Chapter 3, "Avoiding the with statement," on page 87 of *Using ActionScript in Flash*.

Example

The following example sets the `_x` and `_y` properties of the `someOther_mc` instance, and then instructs `someOther_mc` to go to Frame 3 and stop.

```
with (someOther_mc) {
  _x = 50;
  _y = 100;
  gotoAndStop(3);
}
```

The following code snippet shows how to write the preceding code without using a `with` statement.

```
someOther_mc._x = 50;
someOther_mc._y = 100;
someOther_mc.gotoAndStop(3);
```

The `with` statement is useful for accessing multiple items in a scope chain list simultaneously. In the following example, the built-in `Math` object is placed at the front of the scope chain. Setting `Math` as a default object resolves the identifiers `cos`, `sin`, and `PI` to `Math.cos`, `Math.sin`, and `Math.PI`, respectively. The identifiers `a`, `x`, `y`, and `r` are not methods or properties of the `Math` object, but because they exist in the object activation scope of the function `polar()`, they resolve to the corresponding local variables.

```
function polar(r:Number):Void {
  var a:Number, x:Number, y:Number;
  with (Math) {
    a = PI*pow(r, 2);
    x = r*cos(PI);
    y = r*sin(PI/2);
  }
  trace("area = "+a);
  trace("x = "+x);
  trace("y = "+y);
}
polar(3);
```

The following result is displayed in the Output panel.

```
area = 28.2743338823081
x = -3
y = 3
```

See also

`tellTarget`

XML class

Availability

Flash Player 5 (became a native object in Flash Player 6, which improved performance significantly).

Description

Use the methods and properties of the XML class to load, parse, send, build, and manipulate XML document trees.

You must use the constructor `new XML()` to create an XML object before calling any method of the XML class.

An XML document is represented in Flash by the XML class. Each element of the hierarchical document is represented by an XMLNode object.

Note: The XML and XMLNode objects are modeled after the W3C DOM Level 1 recommendation: www.w3.org/TR/1998/REC-DOM-Level-1-19981001/level-one-core.html. That recommendation specifies a Node interface and a Document interface. The Document interface inherits from the Node interface, and adds methods such as `createElement()` and `createTextNode()`. In ActionScript, the XML and XMLNode objects are designed to divide functionality along similar lines.

Method summary for the XML class

Method	Description
`XML.addRequestHeader()`	Adds or changes HTTP headers for `POST` operations.
`XML.appendChild()`	Appends a node to the end of the specified object's child list.
`XML.cloneNode()`	Clones the specified node and, optionally, recursively clones all children.
`XML.createElement()`	Creates a new XML element.
`XML.createTextNode()`	Creates a new XML text node.
`XML.getBytesLoaded()`	Returns the number of bytes loaded for the specified XML document.
`XML.getBytesTotal()`	Returns the size of the XML document, in bytes.
`XML.hasChildNodes()`	Returns `true` if the specified node has child nodes; otherwise, returns `false`.
`XML.insertBefore()`	Inserts a node in front of an existing node in the specified node's child list.
`XML.load()`	Loads a document (specified by the XML object) from a URL.
`XML.parseXML()`	Parses an XML document into the specified XML object tree.
`XML.removeNode()`	Removes the specified node from its parent.
`XML.send()`	Sends the specified XML object to a URL.
`XML.sendAndLoad()`	Sends the specified XML object to a URL, and loads the server response into another XML object.
`XML.toString()`	Converts the specified node and any children to XML text.

Property summary for the XML class

Property	Description
`XML.contentType`	The MIME type transmitted to the server.
`XML.docTypeDecl`	Sets and returns information about an XML document's `DOCTYPE` declaration.
`XML.firstChild`	Read-only; references the first child in the list for the specified node.
`XML.ignoreWhite`	When set to `true`, discards, during the parsing process, text nodes that contain only white space.
`XML.lastChild`	Read-only; references the last child in the list for the specified node.
`XML.loaded`	Read-only; checks whether the specified XML object has loaded.
`XML.nextSibling`	Read-only; references the next sibling in the parent node's child list.
`XML.nodeName`	The node name of an XML object.
`XML.nodeType`	The type of the specified node (XML element or text node).
`XML.nodeValue`	The text of the specified node if the node is a text node.
`XML.parentNode`	Read-only; references the parent node of the specified node.
`XML.previousSibling`	Read-only; references the previous sibling in the parent node's child list.
`XML.status`	A numeric status code that indicates the success or failure of an XML document parsing operation.
`XML.xmlDecl`	Specifies information about a document's XML declaration.

Collections summary for the XML class

Method	Description
`XML.attributes`	Returns an associative array that contains all the attributes of the specified node.
`XML.childNodes`	Read-only; returns an array that contains references to the child nodes of the specified node.

Event handler summary for the XML class

Event handler	Description
`XML.onData`	Invoked when XML text has been completely downloaded from the server, or when an error occurs downloading XML text from a server.
`XML.onLoad`	Returns a Boolean value indicating whether the XML object was successfully loaded with `XML.load()` or `XML.sendAndLoad()`.

Constructor for the XML class

Availability

Flash Player 5.

Usage

```
new XML([source:String]) : XML
```

Parameters

`source`	A string; the XML text parsed to create the new XML object.

Returns

A reference to an XML object.

Description

Constructor; creates a new XML object. You must use the constructor to create an XML object before you call any of the methods of the XML class.

Note: Use the `createElement()` and `createTextNode()` methods to add elements and text nodes to an XML document tree.

Example

The following example creates a new, empty XML object:

```
var my_xml:XML = new XML();
```

The following example creates an XML object by parsing the XML text specified in the *`source`* parameter, and populates the newly created XML object with the resulting XML document tree:

```
var other_xml:XML = new XML("<state name=\"California\">
  <city>San Francisco</city></state>");
```

See also

`XML.createElement()`, `XML.createTextNode()`

XML.addRequestHeader()

Availability

Flash Player 6.

Usage

```
my_xml.addRequestHeader(headerName:String, headerValue:String) : Void
my_xml.addRequestHeader(["headerName_1":String, "headerValue_1":String ...
  "headerName_n":String, "headerValue_n":String]) : Void
```

Parameters

`headerName` A string that represents an HTTP request header name.

`headerValue` A string that represents the value associated with `headerName`.

Returns

Nothing.

Description

Method; adds or changes HTTP request headers (such as `Content-Type` or `SOAPAction`) sent with `POST` actions. In the first usage, you pass two strings to the method: `headerName` and `headerValue`. In the second usage, you pass an array of strings, alternating header names and header values.

If multiple calls are made to set the same header name, each successive value replaces the value set in the previous call.

You cannot add or change the following standard HTTP headers using this method: `Accept-Ranges`, `Age`, `Allow`, `Allowed`, `Connection`, `Content-Length`, `Content-Location`, `Content-Range`, `ETag`, `Host`, `Last-Modified`, `Locations`, `Max-Forwards`, `Proxy-Authenticate`, `Proxy-Authorization`, `Public`, `Range`, `Retry-After`, `Server`, `TE`, `Trailer`, `Transfer-Encoding`, `Upgrade`, `URI`, `Vary`, `Via`, `Warning`, and `WWW-Authenticate`.

Example

The following example adds a custom HTTP header named `SOAPAction` with a value of `Foo` to an XML object named `my_xml`:

```
my_xml.addRequestHeader("SOAPAction", "'Foo'");
```

The following example creates an array named `headers` that contains two alternating HTTP headers and their associated values. The array is passed as a parameter to the `addRequestHeader()` method.

```
var headers:Array = new Array("Content-Type", "text/plain",
  "X-ClientAppVersion", "2.0");
my_xml.addRequestHeader(headers);
```

See also

`LoadVars.addRequestHeader()`

XML.appendChild()

Availability

Flash Player 5.

Usage

my_xml.appendChild(*childNode:XMLNode*) *: Void*

Parameters

childNode An XMLNode that represents the node to be moved from its current location to the child list of the *my_xml* object.

Returns

Nothing.

Description

Method; appends the specified node to the XML object's child list. This method operates directly on the node referenced by the *childNode* parameter; it does not append a copy of the node. If the node to be appended already exists in another tree structure, appending the node to the new location will remove it from its current location. If the *childNode* parameter refers to a node that already exists in another XML tree structure, the appended child node is placed in the new tree structure after it is removed from its existing parent node.

Example

This example does the following things in the order shown:

- Creates two empty XML documents, doc1 and doc2.
- Creates a new node using the createElement() method, and appends it, using the appendChild() method, to the XML document named doc1.
- Shows how to move a node using the appendChild() method, by moving the root node from doc1 to doc2.
- Clones the root node from doc2 and appends it to doc1.
- Creates a new node and appends it to the root node of the XML document doc1.

```
var doc1:XML = new XML();
var doc2:XML = new XML();

// create a root node and add it to doc1
var rootnode:XMLNode = doc1.createElement("root");
doc1.appendChild(rootnode);
trace ("doc1: " + doc1); // output: doc1: <root />
trace ("doc2: " + doc2); // output: doc2:

// move the root node to doc2
doc2.appendChild(rootnode);
trace ("doc1: " + doc1); // output: doc1:
trace ("doc2: " + doc2); // output: doc2: <root />

// clone the root node and append it to doc1
```

```
var clone:XMLNode = doc2.firstChild.cloneNode(true);
doc1.appendChild(clone);
trace ("doc1: " + doc1); // output: doc1: <root />
trace ("doc2: " + doc2); // output: doc2: <root />

// create a new node to append to root node (named clone) of doc1
var newNode:XMLNode = doc1.createElement("newbie");
clone.appendChild(newNode);
trace ("doc1: " + doc1); // output: doc1: <root><newbie /></root>
```

XML.attributes

Availability

Flash Player 5.

Usage

`my_xml.attributes:Array`

Description

Property; an associative array that contains all the attributes of the specified XML object. Associative arrays use *keys* as indexes, instead of the simple ordinal integer indexes used by regular arrays. In the `XML.attributes` associative array, the key index is a string representing the name of the attribute. The value associated with that key index is the string value associated with that attribute. For example, if you have an attribute named *color*, you would retrieve that attribute's value by using the color as the key index, as the following code shows:

```
var myColor:String = doc.firstChild.attributes.color
```

Example

The following example shows the XML attribute names:

```
// create a tag called 'mytag' with
// an attribute called 'name' with value 'Val'
var doc:XML = new XML("<mytag name=\"Val\"> item </mytag>");

// assign the value of the 'name' attribute to variable y
var y:String = doc.firstChild.attributes.name;
trace (y);  // output: Val

// create a new attribute named 'order' with value 'first'
doc.firstChild.attributes.order = "first";

// assign the value of the 'order' attribute to variable z
var z:String = doc.firstChild.attributes.order
trace(z);  // output: first
```

The following is displayed in the Output panel:

```
Val
first
```

XML.childNodes

Availability

Flash Player 5.

Usage

my_xml.childNodes:*Array*

Description

Read-only property; an array of the specified XML object's children. Each element in the array is a reference to an XML object that represents a child node. This is a read-only property and cannot be used to manipulate child nodes. Use the XML.appendChild(), XML.insertBefore(), and XML.removeNode() methods to manipulate child nodes.

This property is undefined for text nodes (nodeType == 3).

Example

The following example shows how to use the XML.childNodes property to return an array of child nodes:

```
// create a new XML document
var doc:XML = new XML();

// create a root node
var rootNode:XMLNode = doc.createElement("rootNode");

// create three child nodes
var oldest:XMLNode = doc.createElement("oldest");
var middle:XMLNode = doc.createElement("middle");
var youngest:XMLNode = doc.createElement("youngest");

// add the rootNode as the root of the XML document tree
doc.appendChild(rootNode);

// add each of the child nodes as children of rootNode
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// create an array and use rootNode to populate it
var firstArray:Array = doc.childNodes;
trace (firstArray);
// output: <rootNode><oldest /><middle /><youngest /></rootNode>

// create another array and use the child nodes to populate it
var secondArray:Array = rootNode.childNodes;
trace(secondArray);
// output: <oldest />,<middle />,<youngest />
```

See also

XML.nodeType

XML.cloneNode()

Availability

Flash Player 5.

Usage

`my_xml``.cloneNode(`*`deep:Boolean`*`) :` *`XMLNode`*

Parameters

`deep` A Boolean value; if set to `true`, the children of the specified XML object will be recursively cloned.

Returns

An XMLNode object.

Description

Method; constructs and returns a new XML node of the same type, name, value, and attributes as the specified XML object. If *`deep`* is set to `true`, all child nodes are recursively cloned, resulting in an exact copy of the original object's document tree.

The clone of the node that is returned is no longer associated with the tree of the cloned item. Consequently, `nextSibling`, `parentNode`, and `previousSibling` all have a value of `null`. If the *`deep`* parameter is set to `false`, or the *`my_xml`* node has no child nodes, `firstChild` and `lastChild` are also `null`.

Example

The following example shows how to use the `XML.cloneNode()` method to create a copy of a node:

```
// create a new XML document
var doc:XML = new XML();

// create a root node
var rootNode:XMLNode = doc.createElement("rootNode");

// create three child nodes
var oldest:XMLNode = doc.createElement("oldest");
var middle:XMLNode = doc.createElement("middle");
var youngest:XMLNode = doc.createElement("youngest");

// add the rootNode as the root of the XML document tree
doc.appendChild(rootNode);

// add each of the child nodes as children of rootNode
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// create a copy of the middle node using cloneNode()
var middle2:XMLNode = middle.cloneNode(false);
```

```
// insert the clone node into rootNode between the middle and youngest nodes
rootNode.insertBefore(middle2, youngest);
trace(rootNode);
// output (with line breaks added):
// <rootNode>
//    <oldest />
//    <middle />
//    <middle />
//    <youngest />
// </rootNode>

// create a copy of rootNode using cloneNode() to demonstrate a deep copy
var rootClone:XMLNode = rootNode.cloneNode(true);

// insert the clone, which contains all child nodes,  to rootNode
rootNode.appendChild(rootClone);
trace(rootNode);
// output (with line breaks added):
// <rootNode>
//   <oldest />
//   <middle />
//   <middle />
//   <youngest />
//   <rootNode>
//      <oldest />
//      <middle />
//      <middle />
//      <youngest />
//   </rootNode>
// </rootNode>
```

XML.contentType

Availability

Flash Player 6.

Usage

my_xml.contentType:*String*

Description

Property; the MIME content type that is sent to the server when you call the XML.send() or XML.sendAndLoad() method. The default is `application/x-www-form-urlencoded`, which is the standard MIME content type used for most HTML forms.

Example

The following example creates a new XML document and checks its default content type:

```
// create a new XML document
var doc:XML = new XML();

// trace the default content type
trace(doc.contentType);  // output: application/x-www-form-urlencoded
```

The following example defines an XML packet, and sets the content type for the XML object. The data is then sent to a server and shows a result in a browser window.

```
var my_xml:XML = new XML("<highscore><name>Ernie</name><score>13045</score>
  </highscore>");
my_xml.contentType = "text/xml";
my_xml.send("http://www.flash-mx.com/mm/highscore.cfm", "_blank");
```

Press F12 to test this example in a browser.

See also

`XML.send()`, `XML.sendAndLoad()`

XML.createElement()

Availability

Flash Player 5.

Usage

my_xml.createElement(*name*:*String*) : *XMLNode*

Parameters

name The tag name of the XML element being created.

Returns

An XMLNode; an XML element.

Description

Method; creates a new XML element with the name specified in the parameter. The new element initially has no parent, no children, and no siblings. The method returns a reference to the newly created XML object that represents the element. This method and the XML.createTextNode() method are the constructor methods for creating nodes for an XML object.

Example

The following example creates three XML nodes using the `createElement()` method:

```
// create an XML document
var doc:XML = new XML();

// create three XML nodes using createElement()
var element1:XMLNode = doc.createElement("element1");
var element2:XMLNode = doc.createElement("element2");
var element3:XMLNode = doc.createElement("element3");

// place the new nodes into the XML tree
doc.appendChild(element1);
element1.appendChild(element2);
element1.appendChild(element3);

trace(doc);
// output: <element1><element2 /><element3 /></element1>
```

See also

`XML.createTextNode()`

XML.createTextNode()

Availability

Flash Player 5.

Usage

```
my_xml.createTextNode(text:String) : XMLNode
```

Parameters

`text` A string; the text used to create the new text node.

Returns

An XMLNode.

Description

Method; creates a new XML text node with the specified text. The new node initially has no parent, and text nodes cannot have children or siblings. This method returns a reference to the XML object that represents the new text node. This method and the XML.createElement() method are the constructor methods for creating nodes for an XML object.

Example

The following example creates two XML text nodes using the `createTextNode()` method, and places them into existing XML nodes:

```
// create an XML document
var doc:XML = new XML();

// create three XML nodes using createElement()
var element1:XMLNode = doc.createElement("element1");
var element2:XMLNode = doc.createElement("element2");
var element3:XMLNode = doc.createElement("element3");

// place the new nodes into the XML tree
doc.appendChild(element1);
element1.appendChild(element2);
element1.appendChild(element3);

// create two XML text nodes using createTextNode()
var textNode1:XMLNode = doc.createTextNode("textNode1");
var textNode2:XMLNode = doc.createTextNode("textNode2");

// place the new nodes into the XML tree
element2.appendChild(textNode1);
element3.appendChild(textNode2);

trace(doc);
// output (with line breaks added between tags):
// <element1>
//    <element2>textNode1</element2>
//    <element3>textNode2</element3>
// </element1>
```

See also

`XML.createElement()`

XML.docTypeDecl

Availability

Flash Player 5.

Usage

```
my_xml.docTypeDecl:String
```

Description

Property; specifies information about the XML document's `DOCTYPE` declaration. After the XML text has been parsed into an XML object, the `XML.docTypeDecl` property of the XML object is set to the text of the XML document's `DOCTYPE` declaration (for example, `<!DOCTYPE greeting SYSTEM "hello.dtd">`). This property is set using a string representation of the `DOCTYPE` declaration, not an XML node object.

The ActionScript XML parser is not a validating parser. The `DOCTYPE` declaration is read by the parser and stored in the `XML.docTypeDecl` property, but no DTD validation is performed.

If no `DOCTYPE` declaration was encountered during a parse operation, the `XML.docTypeDecl` property is set to `undefined`. The XML.toString() method outputs the contents of `XML.docTypeDecl` immediately after the XML declaration stored in `XML.xmlDecl`, and before any other text in the XML object. If `XML.docTypeDecl` is undefined, no `DOCTYPE` declaration is output.

Example

The following example uses the `XML.docTypeDecl` property to set the `DOCTYPE` declaration for an XML object:

```
my_xml.docTypeDecl = "<!DOCTYPE greeting SYSTEM \"hello.dtd\">";
```

See also

`XML.toString()`, `XML.xmlDecl`

XML.firstChild

Availability

Flash Player 5.

Usage

`my_xml.firstChild:XMLNode`

Description

Read-only property; evaluates the specified XML object and references the first child in the parent node's child list. This property is `null` if the node does not have children. This property is `undefined` if the node is a text node. This is a read-only property and cannot be used to manipulate child nodes; use the `appendChild()`, `insertBefore()`, and `removeNode()` methods to manipulate child nodes.

Example

The following example shows how to use `XML.firstChild` to loop through a node's child nodes:

```
// create a new XML document
var doc:XML = new XML();

// create a root node
var rootNode:XMLNode = doc.createElement("rootNode");

// create three child nodes
var oldest:XMLNode = doc.createElement("oldest");
var middle:XMLNode = doc.createElement("middle");
var youngest:XMLNode = doc.createElement("youngest");

// add the rootNode as the root of the XML document tree
doc.appendChild(rootNode);

// add each of the child nodes as children of rootNode
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// use firstChild to iterate through the child nodes of rootNode
for (var aNode:XMLNode = rootNode.firstChild; aNode != null; aNode =
  aNode.nextSibling) {
  trace(aNode);
}

// output:
// <oldest />
// <middle />
// <youngest />
```

The following example is from the XML_languagePicker FLA file in the Examples directory and can be found in the `languageXML.onLoad` event handler function definition:

```
// loop through the strings in each language node
// adding each string as a new element in the language array
```

```
for (var stringNode:XMLNode = childNode.firstChild; stringNode != null;
  stringNode = stringNode.nextSibling, j++) {
  masterArray[i][j] = stringNode.firstChild.nodeValue;
}
```

To view the entire script, see XML_languagePicker.fla in the HelpExamples folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/

See also

`XML.appendChild()`, `XML.insertBefore()`, `XML.removeNode()`

XML.getBytesLoaded()

Availability

Flash Player 6.

Usage

my_xml.getBytesLoaded() : *Number*

Parameters

None.

Returns

An integer that indicates the number of bytes loaded.

Description

Method; returns the number of bytes loaded (streamed) for the XML document. You can compare the value of `getBytesLoaded()` with the value of `getBytesTotal()` to determine what percentage of an XML document has loaded.

Example

The following example shows how to use the `XML.getBytesLoaded()` method with the XML.getBytesTotal() method to trace the progress of an XML.load() command. You must replace the URL parameter of the `XML.load()` command so that the parameter refers to a valid XML file using HTTP. If you attempt to use this example to load a local file that resides on your hard disk, this example will not work properly because in test movie mode Flash Player loads local files in their entirety.

```
// create a new XML document
var doc:XML = new XML();

var checkProgress = function(xmlObj:XML) {
  var bytesLoaded:Number = xmlObj.getBytesLoaded();
  var bytesTotal:Number = xmlObj.getBytesTotal();
  var percentLoaded:Number = Math.floor((bytesLoaded / bytesTotal ) * 100);
  trace ("milliseconds elapsed: " + getTimer());
  trace ("bytesLoaded: " + bytesLoaded);
  trace ("bytesTotal: " + bytesTotal);
  trace ("percent loaded: " + percentLoaded);
  trace ("---------------------------------");
}

doc.onLoad = function(success:Boolean) {
  clearInterval(intervalID);
  trace("intervalID: " + intervalID);
}
doc.load("[place a valid URL pointing to an XML file here]");
var intervalID:Number = setInterval(checkProgress, 100, doc);
```

See also

`XML.getBytesTotal()`

XML.getBytesTotal()

Availability

Flash Player 6.

Usage

```
my_xml.getBytesTotal() : Number
```

Parameters

None.

Returns

An integer.

Description

Method; returns the size, in bytes, of the XML document.

Example

See example for XML.getBytesLoaded().

See also

`XML.getBytesLoaded()`

XML.hasChildNodes()

Availability

Flash Player 5.

Usage

my_xml.hasChildNodes() *: Boolean*

Parameters

None.

Returns

A Boolean value.

Description

Method; returns `true` if the specified XML object has child nodes; otherwise, returns `false`.

Example

The following example creates a new XML packet. If the root node has child nodes, the code loops over each child node to display the name and value of the node. Add the following ActionScript to your FLA or AS file:

```
var my_xml:XML = new XML("<login><username>hank</username>
  <password>rudolph</password></login>");
if (my_xml.firstChild.hasChildNodes()) {
  // use firstChild to iterate through the child nodes of rootNode
  for (var aNode:XMLNode = my_xml.firstChild.firstChild; aNode != null;
  aNode=aNode.nextSibling) {
    if (aNode.nodeType == 1) {
      trace(aNode.nodeName+":\t"+aNode.firstChild.nodeValue);
    }
  }
}
```

The following is displayed in the Output panel:

```
output:
username:hank
password:rudolph
```

XML.ignoreWhite

Availability

Flash Player 5.

Usage

```
my_xml.ignoreWhite:boolean
XML.prototype.ignoreWhite:boolean
```

Parameters

boolean A Boolean (`true` or `false`) value.

Description

Property; default setting is `false`. When set to `true`, text nodes that contain only white space are discarded during the parsing process. Text nodes with leading or trailing white space are unaffected.

Usage 1: You can set the `ignoreWhite` property for individual XML objects, as the following code shows:

```
my_xml.ignoreWhite = true;
```

Usage 2: You can set the default `ignoreWhite` property for XML objects, as the following code shows:

```
XML.prototype.ignoreWhite = true;
```

Example

The following example loads an XML file with a text node that contains only white space; the `foyer` tag comprises fourteen space characters. To run this example, create a text file named *flooring.xml*, and copy the following tags into it:

```
<house>
   <kitchen>   ceramic tile   </kitchen>
   <bathroom>linoleum</bathroom>
   <foyer>              </foyer>
</house>
```

Create a new Flash document named *flooring.fla* and save it to the same directory as the XML file. Place the following code into the main Timeline:

```
// create a new XML object
var flooring:XML = new XML();

// set the ignoreWhite property to true (default value is false)
flooring.ignoreWhite = true;

// After loading is complete, trace the XML object
flooring.onLoad = function(success:Boolean) {
  trace(flooring);
}

// load the XML into the flooring object
flooring.load("flooring.xml");
```

```
/* output (line breaks added for clarity):
<house>
   <kitchen>   ceramic tile   </kitchen>
   <bathroom>linoleum</bathroom>
   <foyer />
</house>
*/
```

If you then change the setting of `flooring.ignoreWhite` to `false`, or simply remove that line of code entirely, the fourteen space characters in the `foyer` tag will be preserved:

```
...
// set the ignoreWhite property to false (default value)
flooring.ignoreWhite = false;
...
/* output (line breaks added for clarity):
<house>
   <kitchen>   ceramic tile   </kitchen>
   <bathroom>linoleum</bathroom>
   <foyer>              </foyer>
</house>
*/
```

The XML_blogTracker.fla and XML_languagePicker.fla files in the HelpExamples folder also contain a code example. The following are typical paths to this folder:

- Windows: \Program Files\Macromedia\Flash MX 2004\Samples\HelpExamples\
- Macintosh: HD/Applications/Macromedia Flash MX 2004/Samples/HelpExamples/
-

XML.insertBefore()

Availability

Flash Player 5.

Usage

my_xml.insertBefore(*childNode*:*XMLNode*, *beforeNode*:*XMLNode*) : *Void*

Parameters

childNode The XMLNode object to be inserted.

beforeNode The XMLNode object before the insertion point for the *childNode*.

Returns

Nothing.

Description

Method; inserts a new child node into the XML object's child list, before the *beforeNode* node. If the *beforeNode* parameter is undefined or null, the node is added using the appendChild() method. If *beforeNode* is not a child of *my_xml*, the insertion fails.

Example

The following example is an excerpt from the XML.cloneNode() example:

```
// create a copy of the middle node using cloneNode()
var middle2:XMLNode = middle.cloneNode(false);

// insert the clone node into rootNode between the middle and youngest nodes
rootNode.insertBefore(middle2, youngest);
```

See also

XMLNode class

XML.lastChild

Availability

Flash Player 5.

Usage

```
my_xml.lastChild:XMLNode
```

Description

Read-only property; an XMLNode value that references the last child in the node's child list. The `XML.lastChild` property is `null` if the node does not have children. This property cannot be used to manipulate child nodes; use the `appendChild()`, `insertBefore()`, and `removeNode()` methods to manipulate child nodes.

Example

The following example uses the `XML.lastChild` property to iterate through the child nodes of an XML node, beginning with the last item in the node's child list and ending with the first child of the node's child list:

```
// create a new XML document
var doc:XML = new XML();

// create a root node
var rootNode:XMLNode = doc.createElement("rootNode");

// create three child nodes
var oldest:XMLNode = doc.createElement("oldest");
var middle:XMLNode = doc.createElement("middle");
var youngest:XMLNode = doc.createElement("youngest");

// add the rootNode as the root of the XML document tree
doc.appendChild(rootNode);

// add each of the child nodes as children of rootNode
rootNode.appendChild(oldest);
rootNode.appendChild(middle);
rootNode.appendChild(youngest);

// use lastChild to iterate through the child nodes of rootNode
for (var aNode:XMLNode = rootNode.lastChild; aNode != null; aNode =
  aNode.previousSibling) {
  trace(aNode);
}

/*
output:
<youngest />
<middle />
<oldest />
*/
```

The following example creates a new XML packet and uses the `XML.lastChild` property to iterate through the child nodes of the root node:

```
// create a new XML document
var doc:XML = new XML("<rootNode><oldest /><middle /><youngest /></
  rootNode>");

var rootNode:XMLNode = doc.firstChild;

// use lastChild to iterate through the child nodes of rootNode
for (var aNode:XMLNode = rootNode.lastChild; aNode != null;
  aNode=aNode.previousSibling) {
  trace(aNode);
}
/*
output:
<youngest />
<middle />
<oldest />

*/
```

See also

`XML.appendChild()`, `XML.insertBefore()`, `XML.removeNode()`, `XMLNode class`

XML.load()

Availability

Flash Player 5; behavior changed in Flash Player 7.

Usage

my_xml.load(*url:String*) : *Void*

Parameters

url A string that represents the URL where the XML document to be loaded is located. If the SWF file that issues this call is running in a web browser, *url* must be in the same domain as the SWF file; for details, see the Description section.

Returns

Nothing.

Description

Method; loads an XML document from the specified URL, and replaces the contents of the specified XML object with the downloaded XML data. The URL is relative and is called using HTTP. The load process is asynchronous; it does not finish immediately after the `load()` method is executed.

In SWF files running in a version of the player earlier than Flash Player 7, the *url* parameter must be in the same superdomain as the SWF file that issues this call. A *superdomain* is derived by removing the leftmost component of a file's URL. For example, a SWF file at www.someDomain.com can load data from sources at store.someDomain.com, because both files are in the same superdomain of someDomain.com.

In SWF files of any version running in Flash Player 7 or later, the *url* parameter must be in exactly the same domain (see "Flash Player security features" in *Using ActionScript in Flash*). For example, a SWF file at www.someDomain.com can load data only from sources that are also at www.someDomain.com. If you want to load data from a different domain, you can place a *cross-domain policy file* on the server that is hosting the SWF file. For more information, see "About allowing cross-domain data loading" in *Using ActionScript in Flash*.

When the `load()` method is executed, the XML object property `loaded` is set to `false`. When the XML data finishes downloading, the `loaded` property is set to `true`, and the `onLoad` event handler is invoked. The XML data is not parsed until it is completely downloaded. If the XML object previously contained any XML trees, they are discarded.

You can define a custom function that executes when the `onLoad` event handler of the XML object is invoked.

Example

The following simple example uses the `XML.load()` method:

```
// create a new XML object
var flooring:XML = new XML();

// set the ignoreWhite property to true (default value is false)
```

```
flooring.ignoreWhite = true;

// After loading is complete, trace the XML object
flooring.onLoad = function(success) {
  trace(flooring);
};

// load the XML into the flooring object
flooring.load("flooring.xml");
```

For the contents of the flooring.xml file, and the output that this example produces, see the example for XML.ignoreWhite.

See also

`XML.loaded`, `XML.onLoad`

XML.loaded

Availability

Flash Player 5.

Usage

my_xml.loaded:*Boolean*

Description

Read-only property; a Boolean value that is `true` if the document-loading process initiated by the XML.load() call has completed successfully; otherwise, it is `false`.

Example

The following example uses the `XML.loaded` property in a simple script:

```
var my_xml:XML = new XML();
my_xml.ignoreWhite = true;
my_xml.onLoad = function(success:Boolean) {
  trace("success: "+success);
  trace("loaded:  "+my_xml.loaded);
  trace("status:  "+my_xml.status);
};
my_xml.load("http://www.flash-mx.com/mm/problems/products.xml");
```

Information displays in the Output panel when the `onLoad` handler invokes. If the call completes successfully, `true` displays for the `loaded` status in the Output panel.

```
success: true
loaded:  true
status:  0
```

See also

`XML.load()`, `XML.onLoad`

XML.nextSibling

Availability

Flash Player 5.

Usage

`my_xml.nextSibling:XMLNode`

Description

Read-only property; an XMLNode value that references the next sibling in the parent node's child list. This property is `null` if the node does not have a next sibling node. This property cannot be used to manipulate child nodes; use the `appendChild()`, `insertBefore()`, and `removeNode()` methods to manipulate child nodes.

Example

The following example is an excerpt from the example for the XML.firstChild property, and shows how you can use the `XML.nextSibling` property to loop through an XML node's child nodes:

```
for (var aNode:XMLNode = rootNode.firstChild; aNode != null; aNode =
  aNode.nextSibling) {
  trace(aNode);
}
```

See also

`XML.appendChild()`, `XML.insertBefore()`, `XML.removeNode()`, `XMLNode class`

XML.nodeName

Availability

Flash Player 5.

Usage

my_xml.nodeName:*String*

Description

Property; a string representing the node name of the XML object. If the XML object is an XML element (nodeType == 1), nodeName is the name of the tag that represents the node in the XML file. For example, TITLE is the nodeName of an HTML TITLE tag. If the XML object is a text node (nodeType == 3), nodeName is null.

Example

The following example creates an element node and a text node, and checks the node name of each:

```
// create an XML document
var doc:XML = new XML();

// create an XML node using createElement()
var myNode:XMLNode = doc.createElement("rootNode");

// place the new node into the XML tree
doc.appendChild(myNode);

// create an XML text node using createTextNode()
var myTextNode:XMLNode = doc.createTextNode("textNode");

// place the new node into the XML tree
myNode.appendChild(myTextNode);

trace(myNode.nodeName);
trace(myTextNode.nodeName);

/*
output:
rootNode
null
*/
```

The following example creates a new XML packet. If the root node has child nodes, the code loops over each child node to display the name and value of the node. Add the following ActionScript to your FLA or AS file:

```
var my_xml:XML = new XML("<login><username>hank</username>
  <password>rudolph</password></login>");
if (my_xml.firstChild.hasChildNodes()) {
  // use firstChild to iterate through the child nodes of rootNode
  for (var aNode:XMLNode = my_xml.firstChild.firstChild; aNode != null;
  aNode=aNode.nextSibling) {
    if (aNode.nodeType == 1) {
            trace(aNode.nodeName+":\t"+aNode.firstChild.nodeValue);
        }
    }
}
```

The following node names are displayed in the Output panel:

```
output:
username:hank
password:rudolph
```

See also

`XML.nodeType`

XML.nodeType

Availability

Flash Player 5.

Usage

my_xml.nodeType:*Number*

Description

Read-only property; a `nodeType` value, either 1 for an XML element or 3 for a text node.

The `nodeType` is a numeric value from the NodeType enumeration in the W3C DOM Level 1 recommendation: www.w3.org/TR/1998/REC-DOM-Level-1-19981001/level-one-core.html. The following table lists the values:

Integer value	Defined constant
1	`ELEMENT_NODE`
2	`ATTRIBUTE_NODE`
3	`TEXT_NODE`
4	`CDATA_SECTION_NODE`
5	`ENTITY_REFERENCE_NODE`
6	`ENTITY_NODE`
7	`PROCESSING_INSTRUCTION_NODE`
8	`COMMENT_NODE`
9	`DOCUMENT_NODE`
10	`DOCUMENT_TYPE_NODE`
11	`DOCUMENT_FRAGMENT_NODE`
12	`NOTATION_NODE`

In Flash Player, the built-in XML class only supports 1 (`ELEMENT_NODE`) and 3 (`TEXT_NODE`).

Example

The following example creates an element node and a text node, and checks the node type of each:

```
// create an XML document
var doc:XML = new XML();

// create an XML node using createElement()
var myNode:XMLNode = doc.createElement("rootNode");

// place the new node into the XML tree
doc.appendChild(myNode);

// create an XML text node using createTextNode()
```

```
var myTextNode:XMLNode = doc.createTextNode("textNode");

// place the new node into the XML tree
myNode.appendChild(myTextNode);

trace(myNode.nodeType);
trace(myTextNode.nodeType);

/*
output:
1
3
*/
```

See also

`XML.nodeValue`

XML.nodeValue

Availability

Flash Player 5.

Usage

my_xml.nodeValue:*Node*

Description

Property; the node value of the XML object. If the XML object is a text node, the `nodeType` is 3, and the `nodeValue` is the text of the node. If the XML object is an XML element (`nodeType` is 1), `nodeValue` is `null` and read-only.

Example

The following example creates an element node and a text node, and checks the node value of each:

```
// create an XML document
var doc:XML = new XML();

// create an XML node using createElement()
var myNode:XMLNode = doc.createElement("rootNode");

// place the new node into the XML tree
doc.appendChild(myNode);

// create an XML text node using createTextNode()
var myTextNode:XMLNode = doc.createTextNode("myTextNode");

// place the new node into the XML tree
myNode.appendChild(myTextNode);

trace(myNode.nodeValue);
trace(myTextNode.nodeValue);

/*
output:
null
myTextNode
*/
```

The following example creates and parses an XML packet. The code loops through each child node, and displays the node value using the `firstChild` property and `firstChild.nodeValue`. When you use `firstChild` to display contents of the node, it maintains the `&amp;` entity. However, when you explicitly use `nodeValue`, it converts to the ampersand character (&).

```
var my_xml:XML = new XML("<login><username>morton</username>
  <password>good&amp;evil</password></login>");
trace("using firstChild:");
for (var i = 0; i<my_xml.firstChild.childNodes.length; i++) {
  trace("\t"+my_xml.firstChild.childNodes[i].firstChild);
}
trace("");
trace("using firstChild.nodeValue:");
for (var i = 0; i<my_xml.firstChild.childNodes.length; i++) {
  trace("\t"+my_xml.firstChild.childNodes[i].firstChild.nodeValue);
}
```

The following information is displayed in the Output panel:

```
using firstChild:
  morton
  good&amp;evil

using firstChild.nodeValue:
  morton
  good&evil
```

See also

`XML.nodeType`

XML.onData

Availability

Flash Player 5.

Usage

```
my_xml.onData = function(src:String) {
  // your statements here
}
```

Parameters

src A string or `undefined`; the raw data, usually in XML format, that is sent by the server.

Returns

Nothing.

Description

Event handler; invoked when XML text has been completely downloaded from the server, or when an error occurs downloading XML text from a server. This handler is invoked before the XML is parsed, and you can use it to call a custom parsing routine instead of using the Flash XML parser. The `src` parameter is a string that contains XML text downloaded from the server, unless an error occurs during the download, in which case the `src` parameter is `undefined`.

By default, the `XML.onData` event handler invokes XML.onLoad. You can override the `XML.onData` event handler with custom behavior, but `XML.onLoad` is not called unless you call it in your implementation of `XML.onData`.

Example

The following example shows what the `XML.onData` event handler looks like by default:

```
XML.prototype.onData = function (src:String) {
  if (src == undefined) {
    this.onLoad(false);
  } else {
    this.parseXML(src);
    this.loaded = true;
    this.onLoad(true);
  }
}
```

You can override the `XML.onData` event handler to intercept the XML text without parsing it.

XML.onLoad

Availability

Flash Player 5.

Usage

```
my_xml.onLoad = function (success:Boolean) {
  //your statements here
}
```

Parameters

success A Boolean value that evaluates to `true` if the XML object is successfully loaded with a XML.load() or XML.sendAndLoad() operation; otherwise, it is `false`.

Returns

Nothing.

Description

Event handler; invoked by Flash Player when an XML document is received from the server. If the XML document is received successfully, the *success* parameter is `true`. If the document was not received, or if an error occurred in receiving the response from the server, the *success* parameter is `false`. The default, implementation of this method is not active. To override the default implementation, you must assign a function that contains custom actions.

Example

The following example includes ActionScript for a simple e-commerce storefront application. The `sendAndLoad()` method transmits an XML element that contains the user's name and password, and uses an `XML.onLoad` handler to process the reply from the server.

```
var login_str:String = "<login username=\""+username_txt.text+"\"
  password=\""+password_txt.text+"\" />";
var my_xml:XML = new XML(login_str);
var myLoginReply_xml:XML = new XML();
myLoginReply_xml.ignoreWhite = true;
myLoginReply_xml.onLoad = function(success:Boolean){
  if (success) {
    if ((myLoginReply_xml.firstChild.nodeName == "packet") &&
       (myLoginReply_xml.firstChild.attributes.success == "true")) {
      gotoAndStop("loggedIn");
    } else {
      gotoAndStop("loginFailed");
    }
  } else {
    gotoAndStop("connectionFailed");
  }
};
my_xml.sendAndLoad("http://www.flash-mx.com/mm/login_xml.cfm",
  myLoginReply_xml);
```

See also

`function`, `XML.load()`, `XML.sendAndLoad()`

XML.parentNode

Availability

Flash Player 5.

Usage

my_xml.parentNode:*XMLNode*

Description

Read-only property; an XMLNode value that references the parent node of the specified XML object, or returns `null` if the node has no parent. This is a read-only property and cannot be used to manipulate child nodes; use the `appendChild()`, `insertBefore()`, and `removeNode()` methods to manipulate child nodes.

Example

The following example creates an XML packet and displays the parent node of the `username` node in the Output panel:

```
var my_xml:XML = new XML("<login><username>morton</username>
  <password>good&amp;evil</password></login>");

// first child is the <login /> node
var rootNode:XMLNode = my_xml.firstChild;

// first child of the root is the <username /> node
var targetNode:XMLNode = rootNode.firstChild;
trace("the parent node of '"+targetNode.nodeName+"' is:
  "+targetNode.parentNode.nodeName);
trace("contents of the parent node are:\n"+targetNode.parentNode);

/* output (line breaks added for clarity):

the parent node of 'username' is: login
contents of the parent node are:
<login>
  <username>morton</username>
  <password>good&amp;evil</password>
</login>

*/
```

See also

`XML.appendChild()`, `XML.insertBefore()`, `XML.removeNode()`, `XMLNode class`

XML.parseXML()

Availability

Flash Player 5.

Usage

`my_xml.parseXML(source:String) : Void`

Parameters

`source` A string that represents the XML text to be parsed and passed to the specified XML object.

Returns

Nothing.

Description

Method; parses the XML text specified in the `source` parameter, and populates the specified XML object with the resulting XML tree. Any existing trees in the XML object are discarded.

Example

The following example creates and parses an XML packet:

```
var xml_str:String = "<state name=\"California\">
  <city>San Francisco</city></state>"

// defining the XML source within the XML constructor:
var my1_xml:XML = new XML(xml_str);
trace(my1_xml.firstChild.attributes.name); // output: California

// defining the XML source using the XML.parseXML method:
var my2_xml:XML = new XML();
my2_xml.parseXML(xml_str);
trace(my2_xml.firstChild.attributes.name); // output: California
```

XML.previousSibling

Availability

Flash Player 5.

Usage

my_xml.previousSibling:*XMLNode*

Description

Read-only property; an XMLNode value that references the previous sibling in the parent node's child list. The property has a value of `null` if the node does not have a previous sibling node. This property cannot be used to manipulate child nodes; use the XML.appendChild(), XML.insertBefore(), and XML.removeNode() methods to manipulate child nodes.

Example

The following example is an excerpt from the example for the XML.lastChild property, and shows how you can use the `XML.previousSibling` property to loop through an XML node's child nodes:

```
for (var aNode:XMLNode = rootNode.lastChild; aNode != null; aNode =
  aNode.previousSibling) {
  trace(aNode);
}
```

See also

`XML.appendChild()`, `XML.insertBefore()`, `XML.removeNode()`, `XMLNode class`

XML.removeNode()

Availability

Flash Player 5.

Usage

my_xml.removeNode() *: Void*

Parameters

None.

Returns

Nothing.

Description

Method; removes the specified XML object from its parent. Also deletes all descendants of the node.

Example

The following example creates an XML packet, and then deletes the specified XML object and its descendant nodes:

```
var xml_str:String = "<state name=\"California\">
  <city>San Francisco</city></state>";

var my_xml:XML = new XML(xml_str);
var cityNode:XMLNode = my_xml.firstChild.firstChild;
trace("before XML.removeNode():\n"+my_xml);
cityNode.removeNode();
trace("");
trace("after XML.removeNode():\n"+my_xml);

/* output (line breaks added for clarity):

before XML.removeNode():
<state name="California">
  <city>San Francisco</city>
</state>

after XML.removeNode():
<state name="California" />

*/
```

XML.send()

Availability

Flash Player 5.

Usage

my_xml.send(*url:String*, [*window:String*]) : *Void*

Parameters

url String; the destination URL for the specified XML object.

window String; the browser window to show data that the server returns:

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

  This parameter is optional; if you do not specify a *window* parameter, it is the same as specifying _self.

Returns

Nothing.

Description

Method; encodes the specified XML object into an XML document, and sends it to the specified URL using the POST method in a browser. The Flash test environment only uses the GET method.

Example

The following example defines an XML packet and sets the content type for the XML object. The data is then sent to a server and shows a result in a browser window.

```
var my_xml:XML = new XML("<highscore><name>Ernie</name>
  <score>13045</score></highscore>");
my_xml.contentType = "text/xml";
my_xml.send("http://www.flash-mx.com/mm/highscore.cfm", "_blank");
```

Press F12 to test this example in a browser.

See also

XML.sendAndLoad()

XML.sendAndLoad()

Availability

Flash Player 5; behavior changed in Flash Player 7.

Usage

```
my_xml.sendAndLoad(url:String, targetXMLobject:XML) : Void
```

Parameters

`url` A string; the destination URL for the specified XML object. If the SWF file issuing this call is running in a web browser, `url` must be in the same domain as the SWF file; for details, see the Description section.

`targetXMLobject` An XML object created with the XML constructor method that will receive the return information from the server.

Returns

Nothing.

Description

Method; encodes the specified XML object into an XML document, sends it to the specified URL using the `POST` method, downloads the server's response, and loads it into the `targetXMLobject` specified in the parameters. The server response loads in the same manner used by the `XML.load()` method.

In SWF files running in a version of the player earlier than Flash Player 7, the `url` parameter must be in the same superdomain as the SWF file that is issuing this call. A *superdomain* is derived by removing the left-most component of a file's URL. For example, a SWF file at www.someDomain.com can load data from sources at store.someDomain.com, because both files are in the same superdomain of someDomain.com.

In SWF files of any version running in Flash Player 7 or later, the `url` parameter must be in exactly the same domain (see "Flash Player security features" in *Using ActionScript in Flash*). For example, a SWF file at www.someDomain.com can load data only from sources that are also at www.someDomain.com. If you want to load data from a different domain, you can place a *cross-domain policy file* on the server hosting the SWF file. For more information, see "About allowing cross-domain data loading" in *Using ActionScript in Flash*.

When `sendAndLoad()` is executed, the XML object property `loaded` is set to `false`. When the XML data finishes downloading, the `loaded` property is set to `true` if the data successfully loaded, and the `onLoad` event handler is invoked. The XML data is not parsed until it is completely downloaded. If the XML object previously contained any XML trees, they are discarded.

Example

The following example includes ActionScript for a simple e-commerce storefront application. The `XML.sendAndLoad()` method transmits an XML element that contains the user's name and password, and uses an `onLoad` handler to process the reply from the server.

```
var login_str:String = "<login username=\""+username_txt.text+"\"
  password=\""+password_txt.text+"\" />";
var my_xml:XML = new XML(login_str);
var myLoginReply_xml:XML = new XML();
myLoginReply_xml.ignoreWhite = true;
myLoginReply_xml.onLoad = myOnLoad;
my_xml.sendAndLoad("http://www.flash-mx.com/mm/login_xml.cfm",
  myLoginReply_xml);
function myOnLoad(success:Boolean) {
  if (success) {
    if ((myLoginReply_xml.firstChild.nodeName == "packet") &&
       (myLoginReply_xml.firstChild.attributes.success == "true")) {
      gotoAndStop("loggedIn");
    } else {
      gotoAndStop("loginFailed");
    }
  } else {
    gotoAndStop("connectionFailed");
  }
}
```

See also

`XML.send()`, `XML.load()`, `XML.loaded`, `XML.onLoad`

XML.status

Availability

Flash Player 5.

Usage

```
my_xml.status:Number
```

Description

Property; automatically sets and returns a numeric value that indicates whether an XML document was successfully parsed into an XML object. The following are the numeric status codes, with descriptions:

- 0 No error; parse was 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 encountered without a matching start-tag.

Example

The following example loads an XML packet into a SWF file. A status message displays, depending on whether the XML loads and parses successfully. Add the following ActionScript to your FLA or AS file:

```
var my_xml:XML = new XML();
my_xml.onLoad = function(success:Boolean) {
  if (success) {
    if (my_xml.status == 0) {
      trace("XML was loaded and parsed successfully");
    } else {
      trace("XML was loaded successfully, but was unable to be parsed.");
    }
    var errorMessage:String;
    switch (my_xml.status) {
    case 0 :
      errorMessage = "No error; parse was completed successfully.";
      break;
    case -2 :
      errorMessage = "A CDATA section was not properly terminated.";
      break;
    case -3 :
      errorMessage = "The XML declaration was not properly terminated.";
      break;
    case -4 :
      errorMessage = "The DOCTYPE declaration was not properly terminated.";
      break;
    case -5 :
      errorMessage = "A comment was not properly terminated.";
      break;
    case -6 :
      errorMessage = "An XML element was malformed.";
      break;
    case -7 :
      errorMessage = "Out of memory.";
      break;
    case -8 :
      errorMessage = "An attribute value was not properly terminated.";
      break;
    case -9 :
      errorMessage = "A start-tag was not matched with an end-tag.";
      break;
    case -10 :
      errorMessage = "An end-tag was encountered without a matching
        start-tag.";
      break;
    default :
      errorMessage = "An unknown error has occurred.";
      break;
    }
    trace("status: "+my_xml.status+" ("+errorMessage+")");
  } else {
    trace("Unable to load/parse XML. (status: "+my_xml.status+")");
  }
};
my_xml.load("http://www.flash-mx.com/mm/badxml.xml");
```

XML.toString()

Availability

Flash Player 5.

Usage

my_xml.toString() *: String*

Parameters

None.

Returns

A string.

Description

Method; evaluates the specified XML object, constructs a textual representation of the XML structure, including the node, children, and attributes, and returns the result as a string.

For top-level XML objects (those created with the constructor), the `XML.toString()` method outputs the document's XML declaration (stored in the `XML.xmlDecl` property), followed by the document's `DOCTYPE` declaration (stored in the `XML.docTypeDecl` property), followed by the text representation of all XML nodes in the object. The XML declaration is not output if the `XML.xmlDecl` property is `undefined`. The `DOCTYPE` declaration is not output if the `XML.docTypeDecl` property is `undefined`.

Example

The following example of the `XML.toString()` method displays `<h1>test</h1>` in the Output panel:

```
var node:XML = new XML("<h1>test</h1>");
trace(node.toString());
```

See also

`XML.docTypeDecl`, `XML.xmlDecl`

XML.xmlDecl

Availability

Flash Player 5.

Usage

my_xml.xmlDecl*:String*

Description

Property; a string that specifies information about a document's XML declaration. After the XML document is parsed into an XML object, this property is set to the text of the document's XML declaration. This property is set using a string representation of the XML declaration, not an XML node object. If no XML declaration is encountered during a parse operation, the property is set to undefined.XML. The XML.toString() method outputs the contents of the XML.xmlDecl property before any other text in the XML object. If the XML.xmlDecl property contains the undefined type, no XML declaration is output.

Example

The following example creates a text field called my_txt that has the same dimensions as the Stage. The text field displays properties of the XML packet that loads into the SWF file. The doc type declaration displays in my_txt. Add the following ActionScript to your FLA or AS file:

```
var my_fmt:TextFormat = new TextFormat();
my_fmt.font = "_typewriter";
my_fmt.size = 12;
my_fmt.leftMargin = 10;

this.createTextField("my_txt", this.getNextHighestDepth(), 0, 0, Stage.width,
  Stage.height);
my_txt.border = true;
my_txt.multiline = true;
my_txt.wordWrap = true;
my_txt.setNewTextFormat(my_fmt);

var my_xml:XML = new XML();
my_xml.ignoreWhite = true;
my_xml.onLoad = function(success:Boolean) {
  var endTime:Number = getTimer();
  var elapsedTime:Number = endTime-startTime;
  if (success) {
    my_txt.text = "xmlDecl:"+newline+my_xml.xmlDecl+newline+newline;
    my_txt.text += "contentType:"+newline+my_xml.contentType+newline+newline;
    my_txt.text += "docTypeDecl:"+newline+my_xml.docTypeDecl+newline+newline;
    my_txt.text += "packet:"+newline+my_xml.toString()+newline+newline;
  } else {
    my_txt.text = "Unable to load remote XML."+newline+newline;
  }
  my_txt.text += "loaded in: "+elapsedTime+" ms.";
};
my_xml.load("http://www.flash-mx.com/crossdomain.xml");
var startTime:Number = getTimer();
```

See also

`XML.docTypeDecl`, `XML.toString()`

XMLNode class

Availability

Flash Player 5.

Description

An XML document is represented in Flash by the XML class. Each element of the hierarchical document is represented by an XMLNode object.

The XMLNode class supports the following properties, methods, and collections; for information on their usage, see the corresponding XML class entries.

Property, method, or collection	Corresponding XML class entry
`appendChild()`	`XML.appendChild()`
`attributes`	`XML.attributes`
`childNodes`	`XML.childNodes`
`cloneNode()`	`XML.cloneNode()`
`firstChild`	`XML.firstChild`
`hasChildNodes()`	`XML.hasChildNodes()`
`insertBefore()`	`XML.insertBefore()`
`lastChild`	`XML.lastChild`
`nextSibling`	`XML.nextSibling`
`nodeName`	`XML.nodeName`
`nodeType`	`XML.nodeType`
`nodeValue`	`XML.nodeValue`
`parentNode`	`XML.parentNode`
`previousSibling`	`XML.previousSibling`
`removeNode()`	`XML.removeNode()`
`toString()`	`XML.toString()`

See also

XML class

XMLSocket class

Availability

Flash Player 5.

Description

The XMLSocket class implements client sockets that let the computer running Flash Player communicate with a server computer identified by an IP address or domain name. The XMLSocket class is useful for client-server applications that require low latency, such as real-time chat systems. A traditional HTTP-based chat solution frequently polls the server and downloads new messages using an HTTP request. In contrast, an XMLSocket chat solution maintains an open connection to the server, which lets the server immediately send incoming messages without a request from the client.

To use the XMLSocket class, the server computer must run a daemon that understands the protocol used by the XMLSocket class. The protocol is described in the following list:

- XML messages are sent over a full-duplex TCP/IP stream socket connection.
- Each XML message is a complete XML document, terminated by a zero (0) byte.
- An unlimited number of XML messages can be sent and received over a single XMLSocket connection.

The following restrictions apply to how and where an XMLSocket object can connect to the server:

- The `XMLSocket.connect()` method can connect only to TCP port numbers greater than or equal to 1024. One consequence of this restriction is that the server daemons that communicate with the XMLSocket object must also be assigned to port numbers greater than or equal to 1024. Port numbers below 1024 are often used by system services such as FTP, Telnet, and HTTP, so XMLSocket objects are barred from these ports for security reasons. The port number restriction limits the possibility that these resources will be inappropriately accessed and abused.
- The `XMLSocket.connect()` method can connect only to computers in the same domain where the SWF file resides. This restriction does not apply to SWF files running off a local disk. (This restriction is identical to the security rules for `loadVariables()`, `XML.sendAndLoad()`, and `XML.load()`.) To connect to a server daemon running in a domain other than the one where the SWF resides, you can create a security policy file on the server that allows access from specific domains. For more information on creating policy files for XMLSocket connections, see "About allowing cross-domain data loading" in *Using ActionScript in Flash*.

Setting up a server to communicate with the XMLSocket object can be challenging. If your application does not require real-time interactivity, use the `loadVariables()` function, or Flash HTTP-based XML server connectivity (`XML.load()`, `XML.sendAndLoad()`, `XML.send()`), instead of the XMLSocket class.

To use the methods of the XMLSocket class, you must first use the constructor, `new XMLSocket`, to create an XMLSocket object.

Method summary for the XMLSocket class

Method	Description
`XMLSocket.close()`	Closes an open socket connection.
`XMLSocket.connect()`	Establishes a connection to the specified server.
`XMLSocket.send()`	Sends an XML object to the server.

Event handler summary for the XMLSocket class

Event handler	Description
`XMLSocket.onClose`	Invoked when an XMLSocket connection is closed.
`XMLSocket.onConnect`	Invoked by Flash Player when a connection request initiated through `XMLSocket.connect()` has succeeded or failed.
`XMLSocket.onData`	Invoked when an XML message has been downloaded from the server.
`XMLSocket.onXML`	Invoked when an XML object arrives from the server.

Constructor for the XMLSocket class

Availability

Flash Player 5.

Usage

```
new XMLSocket() : XMLSocket
```

Parameters

None.

Returns

A reference to an XMLSocket object.

Description

Constructor; creates a new XMLSocket object. The XMLSocket object is not initially connected to any server. You must call `XMLSocket.connect()` to connect the object to a server.

Example

The following example creates an XMLSocket object:

```
var socket:XMLSocket = new XMLSocket();
```

XMLSocket.close()

Availability

Flash Player 5.

Usage

```
myXMLSocket.close() : Void
```

Parameters

None.

Returns

Nothing.

Description

Method; closes the connection specified by XMLSocket object.

Example

The following simple example creates an XMLSocket object, attempts to connect to the server, and then closes the connection.

```
var socket:XMLSocket = new XMLSocket();
socket.connect(null, 2000);
socket.close();
```

See also

`XMLSocket.connect()`

XMLSocket.connect()

Availability

Flash Player 5; behavior changed in Flash Player 7.

Usage

```
myXMLSocket.connect(host:String, port:Number) : Boolean
```

Parameters

`host` String; a fully qualified DNS domain name or an IP address in the form *aaa.bbb.ccc.ddd*. You can also specify `null` to connect to the host server on which the SWF file resides. If the SWF file issuing this call is running in a web browser, `host` must be in the same domain as the SWF file; for details, see "Description".

`port` A number; the TCP port number on the host used to establish a connection. The port number must be 1024 or greater.

Returns

A Boolean value.

Description

Method; establishes a connection to the specified Internet host using the specified TCP port (must be 1024 or higher), and returns `true` or `false`, depending on whether a connection is successfully established. If you don't know the port number of your Internet host computer, contact your network administrator.

If you specify `null` for the `host` parameter, the host contacted is the one where the SWF file calling `XMLSocket.connect()` resides. For example, if the SWF file was downloaded from www.yoursite.com, specifying `null` for the host parameter is the same as entering the IP address for www.yoursite.com.

In SWF files running in a version of the player earlier than Flash Player 7, `host` must be in the same superdomain as the SWF file that is issuing this call. For example, a SWF file at www.someDomain.com can load variables from a SWF file at store.someDomain.com because both files are in the same superdomain of someDomain.com.

In SWF files of any version running in Flash Player 7 or later, `host` must be in exactly the same domain (see "Flash Player security features" in *Using ActionScript Help*). For example, a SWF file at www.someDomain.com that is published for Flash Player 5, but is running in Flash Player 7 or later can load variables only from SWF files that are also at www.someDomain.com. If you want to load variables from a different domain, you can place a *cross-domain policy file* on the server hosting the SWF file that is being accessed. For more information, see "About allowing cross-domain data loading" in *Using ActionScript Help*.

When `load()` is executed, the XML object property `loaded` is set to `false`. When the XML data finishes downloading, the `loaded` property is set to `true`, and the `onLoad` event handler is invoked. The XML data is not parsed until it is completely downloaded. If the XML object previously contained any XML trees, they are discarded.

If `XMLSocket.connect()` returns a value of `true`, the initial stage of the connection process is successful; later, the `XMLSocket.onConnect` method is invoked to determine whether the final connection succeeded or failed. If `XMLSocket.connect()` returns `false`, a connection could not be established.

Example

The following example uses `XMLSocket.connect()` to connect to the host where the SWF file resides and uses `trace` to display the return value indicating the success or failure of the connection:

```
var socket:XMLSocket = new XMLSocket()
socket.onConnect = function (success:Boolean) {
  if (success) {
    trace ("Connection succeeded!")
  } else {
    trace ("Connection failed!")
  }
}
if (!socket.connect(null, 2000)) {
  trace ("Connection failed!")
}
```

See also

`function`, `XMLSocket.onConnect`

XMLSocket.onClose

Availability

Flash Player 5.

Usage

```
myXMLSocket.onClose = function() {
  // your statements here
}
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked only when an open connection is closed by the server. The default implementation of this method performs no actions. To override the default implementation, you must assign a function containing custom actions.

Example

The following example executes a trace statement if an open connection is closed by the server:

```
var socket:XMLSocket = new XMLSocket();
socket.connect(null, 2000);
socket.onClose = function () {
  trace("Connection to server lost.");
}
```

See also

`function`, `XMLSocket.onConnect`

XMLSocket.onConnect

Availability

Flash Player 5.

Usage

```
myXMLSocket.onConnect = function(success:Boolean) {
  // your statements here
}
```

Parameters

`success` A Boolean value indicating whether a socket connection is successfully established (`true` or `false`).

Returns

Nothing.

Description

Event handler; invoked by Flash Player when a connection request initiated through XMLSocket.connect() has succeeded or failed. If the connection succeeded, the `success` parameter is `true`; otherwise the `success` parameter is `false`.

The default implementation of this method performs no actions. To override the default implementation, you must assign a function containing custom actions.

Example

The following example illustrates the process of specifying a replacement function for the `onConnect` method in a simple chat application.

After creating the XMLSocket object using the constructor method, the script defines the custom function to be executed when the onConnect event handler is invoked. The function controls the screen to which users are taken, depending on whether a connection is successfully established. If the connection is successfully made, users are taken to the main chat screen on the frame labeled `startChat`. If the connection is not successful, users go to a screen with troubleshooting information on the frame labeled `connectionFailed`.

```
var socket:XMLSocket = new XMLSocket();
socket.onConnect = function (success) {
  if (success) {
    gotoAndPlay("startChat");
  } else {
    gotoAndStop("connectionFailed");
  }
}
```

Finally, the connection is initiated. If `connect()` returns `false`, the SWF file is sent directly to the frame labeled `connectionFailed`, and `onConnect` is never invoked. If `connect()` returns `true`, the SWF file jumps to a frame labeled `waitForConnection`, which is the “Please wait” screen. The SWF file remains on the `waitForConnection` frame until the `onConnect` handler is invoked, which happens at some point in the future depending on network latency.

```
if (!socket.connect(null, 2000)) {
  gotoAndStop("connectionFailed");
} else {
  gotoAndStop("waitForConnection");
}
```

See also

`function`, `XMLSocket.connect()`

XMLSocket.onData

Availability

Flash Player 5.

Usage

```
XMLSocket.onData = function(src:String) {
  // your statements here
}
```

Parameters

src A string containing the data sent by the server.

Returns

Nothing.

Description

Event handler; invoked when a message has been downloaded from the server, terminated by a zero (0) byte. You can override `XMLSocket.onData` to intercept the data sent by the server without parsing it as XML. This is a useful if you're transmitting arbitrarily formatted data packets, and you'd prefer to manipulate the data directly when it arrives, rather than have Flash Player parse the data as XML.

By default, the `XMLSocket.onData` method invokes the `XMLSocket.onXML` method. If you override `XMLSocket.onData` with custom behavior, `XMLSocket.onXML` is not called unless you call it in your implementation of `XMLSocket.onData`.

Example

In this example, the *src* parameter is a string containing XML text downloaded from the server. The zero (0) byte terminator is not included in the string.

```
XMLSocket.prototype.onData = function (src) {
  this.onXML(new XML(src));
}
```

XMLSocket.onXML

Availability

Flash Player 5.

Usage

```
myXMLSocket.onXML = function(object:XML) {
  // your statements here
}
```

Parameter

object An XML object that contains a parsed XML document received from a server.

Returns

Nothing.

Description

Event handler; invoked by Flash Player when the specified XML object containing an XML document arrives over an open XMLSocket connection. An XMLSocket connection can be used to transfer an unlimited number of XML documents between the client and the server. Each document is terminated with a zero (0) byte. When Flash Player receives the zero byte, it parses all the XML received since the previous zero byte or since the connection was established if this is the first message received. Each batch of parsed XML is treated as a single XML document and passed to the `onXML` method.

The default implementation of this method performs no actions. To override the default implementation, you must assign a function containing actions that you define.

Example

The following function overrides the default implementation of the `onXML` method in a simple chat application. The function `myOnXML` instructs the chat application to recognize a single XML element, `MESSAGE`, in the following format.

```
<MESSAGE USER="John" TEXT="Hello, my name is John!" />.
```

```
var socket:XMLSocket = new XMLSocket();
```

The following function `displayMessage()` is assumed to be a user-defined function that displays the message received by the user:

```
socket.onXML = function (doc) {
  var e = doc.firstChild;
  if (e != null && e.nodeName == "MESSAGE") {
    displayMessage(e.attributes.user, e.attributes.text);
  }
}
```

See also

`function`

XMLSocket.send()

Availability

Flash Player 5.

Usage

myXMLSocket`.send(`*object*`:`*XML*`) : `*Void*

Parameters

object An XML object or other data to transmit to the server.

Returns

Nothing.

Description

Method; converts the XML object or data specified in the *object* parameter to a string and transmits it to the server, followed by a zero (0) byte. If *object* is an XML object, the string is the XML textual representation of the XML object. The send operation is asynchronous; it returns immediately, but the data may be transmitted at a later time. The `XMLSocket.send()` method does not return a value indicating whether the data was successfully transmitted.

If the *myXMLSocket* object is not connected to the server (using XMLSocket.connect()), the `XMLSocket.send()` operation will fail.

Example

The following example shows how you could specify a user name and password to send the XML object `my_xml` to the server:

```
var myXMLSocket:XMLSocket = new XMLSocket();
var my_xml:XML = new XML();
var myLogin:XMLNode = my_xml.createElement("login");
myLogin.attributes.username = usernameTextField;
myLogin.attributes.password = passwordTextField;
my_xml.appendChild(myLogin);
myXMLSocket.send(my_xml);
```

See also

`XMLSocket.connect()`

APPENDIX
Deprecated Language Elements

The evolution of ActionScript has deprecated many elements of the language. This appendix lists the deprecated items in alphabetical order and suggests alternatives.

add

Availability

Flash Player 4. This function was deprecated in Flash 5; Macromedia recommends you use the add (+) operator when creating content for Flash Player 5 or later.

Usage

`string1` `add` *`string2`*

Parameters

`string1, string2` A string.

Returns

Nothing.

Description

Operator; concatenates two or more strings. The add (+) operator replaces the Flash 4 `&` operator; Flash Player 4 files that use the `&` operator are automatically converted to use the add (+) operator for string concatenation when brought into the Flash 5 or later authoring environment. Use the add (+) operator to concatenate strings if you are creating content for Flash Player 4 or earlier versions of the Player.

See also

`+ (addition)`

and

Availability

Flash Player 4. This function was deprecated in Flash 5; Macromedia recommends that you use the logical AND (`&&`) operator.

Usage

`condition1` `and` *`condition2`*

Parameters

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

Returns

Nothing.

Description

Operator; performs a logical AND (`&&`) operation in Flash Player 4. If both expressions evaluate to `true`, the entire expression is `true`.

See also

`&& (logical AND)`

Button._highquality

Availability

Flash Player 6. The global version of this function was deprecated in Flash 5 in favor of `_quality`.

Usage

```
my_btn._highquality:Number
```

Description

Property (global); specifies the level of anti-aliasing applied to the current SWF file. Specify 2 (best quality) to apply high quality with bitmap smoothing always on. Specify 1 (high quality) to apply anti-aliasing; this smooths bitmaps if the SWF file does not contain animation and is the default value. Specify 0 (low quality) to prevent anti-aliasing.

Example

Add a button instance on the Stage and name it myBtn_btn. Draw an oval on the Stage using the Oval tool that has a stroke and fill color. Select Frame 1 and add the following ActionScript using the Actions panel:

```
myBtn_btn.onRelease = function(){
  myBtn_btn._highquality = 0;
};
```

When you click `myBtn_btn`, the circle's stroke will look jagged. You could add the following ActionScript instead to affect the SWF globally:

```
_highquality = 0;
```

See also

`_quality`

call()

Availability

Flash Player 4. This action was deprecated in Flash 5 in favor of the `function` statement.

Usage

```
call(frame)
```

Parameters

`frame` The label or number of a frame in the Timeline.

Returns

Nothing.

Description

Action; executes the script in the called frame without moving the playhead to that frame. Local variables do not exist after the script executes.

- If variables are not declared inside a block (`{}`) but the action list was executed with a `call()` action, the variables are local and expire at the end of the current list.
- If variables are not declared inside a block and the current action list was not executed with the `call()` action, the variables are interpreted as Timeline variables.

See also

`function`, `Function.call()`

chr

Availability

Flash Player 4. This function was deprecated in Flash 5 in favor of `String.fromCharCode()`.

Usage

`chr(number)`

Parameters

number	An ASCII code number.

Returns

Nothing.

Description

String function; converts ASCII code numbers to characters.

Example

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

```
myVar = chr(65);
```

See also

`String.fromCharCode()`

eq (equal – string specific)

Availability

Flash Player 4. This operator was deprecated in Flash 5 in favor of the `== (equality)` operator.

Usage

```
expression1 eq expression2
```

Parameters

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

Returns

Nothing.

Description

Operator (comparison); compares two expressions for equality and returns a value of `true` if the string representation of *`expression1`* is equal to the string representation of *`expression2`*, `false` otherwise.

See also

`== (equality)`

ge (greater than or equal to — string specific)

Availability

Flash Player 4. This operator was deprecated in Flash 5 in favor of the >= (greater than or equal to) operator.

Usage

```
expression1 ge expression2
```

Parameters

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

Returns

Nothing.

Description

Operator (comparison); compares the string representation of *expression1* with the string representation of *expression2* and returns `true` if *expression1* is greater than or equal to *expression2*, `false` otherwise.

See also

`>= (greater than or equal to)`

gt (greater than – string specific)

Availability

Flash Player 4. This operator was deprecated in Flash 5 in favor of the > (greater than) operator.

Usage

```
expression1 gt expression2
```

Parameters

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

Description

Operator (comparison); compares the string representation of *`expression1`* with the string representation of *`expression2`* and returns `true` if *`expression1`* is greater than *`expression2`*, `false` otherwise.

See also

`> (greater than)`

_highquality

Availability

Flash Player 4. This function was deprecated in Flash 5 in favor of `_quality`.

Usage

```
_highquality
```

Description

Property (global); specifies the level of anti-aliasing applied to the current SWF file. Specify 2 (best quality) to apply high quality with bitmap smoothing always on. Specify 1 (high quality) to apply anti-aliasing; this will smooth bitmaps if the SWF file does not contain animation. Specify 0 (low quality) to prevent anti-aliasing.

Example

The following ActionScript is placed on the main Timeline, and sets the global quality property to always apply bitmap smoothing in non-animated files.

```
_highquality = 1;
```

See also

`_quality`, `TextField._highquality`

<> (inequality)

Availability

Flash 2 . This operator was deprecated in Flash 5; Macromedia recommends that you use the `!= (inequality)` operator.

Usage

```
expression1 <> expression2
```

Parameters

`expression1,expression2` A number, string, Boolean value, variable, object, array, or function.

Returns

A Boolean value.

Description

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

- Numbers, strings, and Boolean values are compared by value.
- Objects, arrays, and functions are compared by reference.
- Variables are compared by value or by reference depending on their type.

For more information, see "Deprecated Flash 4 operators" in Using ActionScript Help.

See also

`!= (inequality)`

ifFrameLoaded

Availability

Flash Player 3. This action was deprecated in Flash 5; Macromedia recommends that you use the `MovieClip._framesloaded` property.

Usage

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

Parameters

scene An optional string that specifies the name of the scene that must be loaded.

frame The frame number or frame label that must be loaded before the next statement is executed.

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

Returns

Nothing.

Description

Action; checks whether the contents of a specific frame are available locally. Use `ifFrameLoaded` to start playing a simple animation while the rest of the SWF file downloads to the local computer. The difference between using `_framesloaded` and `ifFrameLoaded` is that `_framesloaded` lets you add custom `if` or `else` statements.

See also

`MovieClip._framesloaded`, `MovieClipLoader.addListener()`.

int

Availability

Flash Player 4. This function was deprecated in Flash 5 in favor of `Math.round()`.

Usage

```
int(value)
```

Parameters

`value` A number to be rounded to an integer.

Returns

Nothing.

Description

Function; converts a decimal number to an integer value by truncating the decimal value. This function is equivalent to `Math.floor()` if the *`value`* parameter is positive and `Math.ceil()` if the *`value`* parameter is negative.

See also

`Math.floor()`, `Math.ceil()`

le (less than or equal to — string specific)

Availability

Flash Player 4. This operator was deprecated in Flash 5 in favor of the <= (less than or equal to) operator.

Usage

```
expression1 le expression2
```

Parameters

expression1,expression2 Numbers, strings, or variables.

Returns

Nothing.

Description

Operator (comparison); compares *expression1* to *expression2* and returns a value of `true` if *expression1* is less than or equal to *expression2*, `false` otherwise.

See also

`<= (less than or equal to)`

length

Availability

Flash Player 4. This function, along with all the string functions, was deprecated in Flash 5. Macromedia recommends that you use the methods of the String class and the `String.length` property to perform the same operations.

Usage

```
length(expression)
```

```
length(variable)
```

Parameters

`expression`	A string.
`variable`	The name of a variable.

Returns

The length of the specified string or variable.

Description

String function; returns the length of the specified string or variable.

Example

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

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

The result is 5.

See also

`" " (string delimiter)`, `String class`, `String.length`

lt (less than — string specific)

Availability

Flash Player 4. This operator was deprecated in Flash 5 in favor of the < (less than) operator.

Usage

```
expression1 lt expression2
```

Parameters

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

Description

Operator (comparison); compares *`expression1`* to *`expression2`* and returns `true` if *`expression1`* is less than *`expression2`*, `false` otherwise.

See also

`< (less than)`

maxscroll

Availability

Flash Player 4. This function was deprecated in favor of the `TextField.maxscroll` property.

Usage

```
variable_name.maxscroll
```

Description

Read-only property; indicates the line number of the top line of visible text in a text field when the bottom line in the field is also visible. The `maxscroll` property works with the `scroll` property to control how information appears in a text field. This property can be retrieved, but not modified.

See also

`TextField.maxscroll`, `TextField.scroll`

mbchr

Availability

Flash Player 4. This function was deprecated in favor of the `String.fromCharCode()` method.

Usage

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

Parameters

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

Returns

A string.

Description

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

See also

`String.fromCharCode()`

mblength

Availability

Flash Player 4. This function was deprecated in favor of the String class.

Usage

```
mblength(string)
```

Parameters

string	A string.

Returns

A number.

Description

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

mbord

Availability

Flash Player 4. This function was deprecated in Flash 5 in favor of `String.charCodeAt()`.

Usage

```
mbord(character)
```

Parameters

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

Returns

A number.

Description

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

See also

`String.fromCharCode()`

mbsubstring

Availability

Flash Player 4. This function was deprecated in Flash 5 in favor of `String.substr()`.

Usage

```
mbsubstring(value, index, count)
```

Parameters

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

index The number of the first character to extract.

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

Returns

A string.

Description

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

See also

`String.substr()`

MovieClip._highquality

Availability

Flash Player 6. This function was deprecated in favor of `_quality`.

Usage

my_mc._highquality:*Number*

Description

Property; specifies the level of anti-aliasing applied to the current SWF file. Specify 2 (best quality) to apply high quality with bitmap smoothing always on. Specify 1 (high quality) to apply anti-aliasing; this will smooth bitmaps if the SWF file does not contain animation. Specify 0 (low quality) to prevent anti-aliasing. This property can overwrite the global `_highquality` property.

Example

The following ActionScript specifies that best quality anti-aliasing should be applied to the SWF file.

```
my_mc._highquality = 2;
```

See also

`TextField._quality`

ne (not equal – string specific)

Availability

Flash Player 4. This operator was deprecated in favor of the `!= (inequality)` operator.

Usage

`expression1 ne expression2`

Parameters

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

Returns

A Boolean value.

Description

Operator (comparison); compares `expression1` to `expression2` and returns `true` if `expression1` is not equal to `expression2`; `false` otherwise.

See also

`!= (inequality)`

not

Availability

Flash Player 4. This operator was deprecated in favor of the `! (logical NOT)` operator.

Usage

```
not expression
```

Parameters

`expression` A variable or other expression that converts to a Boolean value.

Description

Operator; performs a logical NOT (!) operation in Flash Player 4.

See also

`! (logical NOT)`

or

Availability

Flash 4 . This operator was deprecated in favor of the `|| (logical OR)` operator.

Usage

`condition1` `or` *`condition2`*

Parameters

`condition1,2`	An expression that evaluates to `true` or `false`.

Returns

Nothing.

Description

Operator; evaluates *`condition1`* and *`condition2`*, and if either expression is `true`, the whole expression is `true`.

See also

`|| (logical OR)`, `| (bitwise OR)`

ord

Availability

Flash Player 4. This function was deprecated in favor of the methods and properties of the String class.

Usage

```
ord(character)
```

Parameters

character	The character to convert to an ASCII code number.

Returns

Nothing.

Description

String function; converts characters to ASCII code numbers.

See also

String class

random

Availability

Flash Player 4. This function was deprecated in Flash 5 in favor of `Math.random()`.

Usage

```
random(value)
```

Parameters

value An integer.

Returns

An integer.

Description

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

Example

The following use of `random()` returns a value of 0, 1, 2, 3, or 4:

```
random(5);
```

See also

`Math.random()`

scroll

Availability

Flash Player 4. This property was deprecated in favor of the scroll property of the TextField class.

Usage

textFieldVariableName`.scroll = ` *x*

Description

Property; controls the display of information in a text field associated with a variable. The `scroll` property defines where the text field begins displaying content; after you set it, Flash Player updates it as the user scrolls through the text field. The `scroll` property is useful for directing users to a specific paragraph in a long passage or creating scrolling text fields. This property can be retrieved and modified.

Example

The following code is attached to an Up button that scrolls the text field named `myText`:

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

See also

`TextField.maxscroll`, `TextField.scroll`

substring

Availability

Flash Player 4. This function was deprecated in favor of `String.substr()`.

Usage

```
substring("string", index, count)
```

Parameters

`string` The string from which to extract the new string.

`index` The number of the first character to extract.

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

Returns

Nothing.

Description

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

See also

`String.substr()`

tellTarget

Availability

Flash Player 3. This function was deprecated in Flash 5; Macromedia recommends that you use dot (.) notation and the `with` statement.

Usage

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

Parameters

`target` A string that specifies the target path of the Timeline to be controlled.

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

Returns

Nothing.

Description

Action; applies the instructions specified in the *`statements`* parameter to the Timeline specified in the *`target`* parameter. The `tellTarget` action is useful for navigation controls. Assign `tellTarget` to buttons that stop or start movie clips elsewhere on the Stage. You can also make movie clips go to a particular frame in that clip. For example, you might assign `tellTarget` to buttons that stop or start movie clips on the Stage or prompt movie clips to jump to a particular frame.

In Flash 5 or later, you can use dot (.) notation instead of the `tellTarget` action. You can use the `with` action to issue multiple actions to the same Timeline. You can use the `with` action to target any object, whereas the `tellTarget` action can target only movie clips.

Example

This `tellTarget` statement controls the movie clip instance `ball` on the main Timeline. Frame 1 of the `ball` instance is blank and has a `stop()` action so it isn't visible on the Stage. When you click the button with the following action, `tellTarget` tells the playhead in `ball` to go to Frame 2, where the animation starts:

```
on(release) {
  tellTarget("_parent.ball") {
    gotoAndPlay(2);
  }
}
```

The following example uses dot (.) notation to achieve the same results:

```
on(release) {
  _parent.ball.gotoAndPlay(2);
}
```

If you need to issue multiple commands to the `ball` instance, you can use the `with` action, as shown in the following statement:

```
on(release) {
  with(_parent.ball) {
    gotoAndPlay(2);
    _alpha = 15;
    _xscale = 50;
    _yscale = 50;
  }
}
```

See also

`with`

TextField._highquality

Availability

Flash Player 6. This function was deprecated in favor of `_quality`.

Usage

```
my_txt._highquality
```

Description

Property (global); specifies the level of anti-aliasing applied to the current SWF file. Specify 2 (best quality) to apply high quality with bitmap smoothing always on. Specify 1 (high quality) to apply anti-aliasing; this will smooth bitmaps if the SWF file does not contain animation. Specify 0 (low quality) to prevent anti-aliasing.

See also

`_quality`

toggleHighQuality()

Availability

Flash 2. This function was deprecated in Flash 5 in favor of `_quality`.

Usage

```
toggleHighQuality()
```

Parameters

None.

Returns

Nothing.

Description

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

Example

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

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

See also

`_highquality`, `_quality`