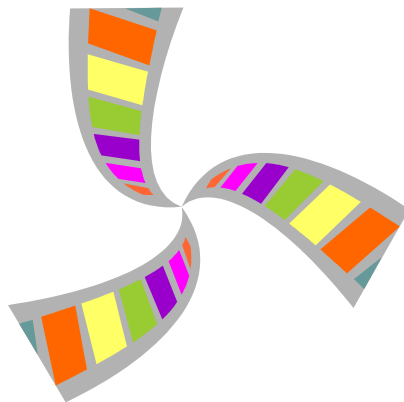




ActionScript Programming



Introduction to ActionScript.....	1
Button scripts.....	3
Loops.....	5
Functions.....	6
Arrays.....	7
Edit fields.....	8
Operators.....	9
Assignment operators.....	11
Internal functions.....	13
MovieClip.....	16
Drawing Sprites.....	21
Array.....	24
Key.....	28
Mouse.....	30
Button.....	31
Math.....	33
Date.....	35
Variable classes.....	37
Sound.....	41
String.....	43
Stage.....	45
System.....	46
TextField.....	49
CSS.....	55
XML.....	57
LoadVars.....	61
Functions not supported by Alligator Flash Designer.....	63

Introduction to ActionScript

ActionScript allows Flash developer to control contents of a Flash document by using commands executed during animation playback.

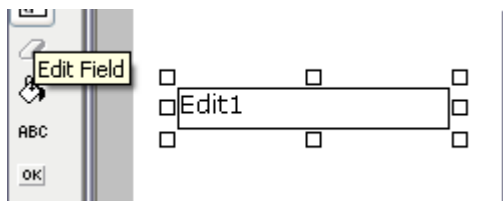
Frame script is executed before the frame appears on the screen. Button script is triggered by events like mouse click or when the mouse cursor enters the area of an object.

Frame Script

To define the script, choose “Frame” > “ActionScript” and enter the script in the dialog box.

The simplest ActionScript

Launch Alligator Flash Designer, draw an edit field using the “Edit field” tool. The field will appear as Edit1, Edit2... etc.).



Choose “Frame” > “ActionScript” and paste the following code:

```
Edit1 = "hello!!!";
```

Press F9 to run the animation. The text “hello!!!” will appear in the edit field.

Language Syntax

ActionScript consists of a series of commands each ending with a semicolon. To maintain script clarity each command shall be entered on a separate line.

Text and Numbers

In order to distinguish text from numbers in ActionScript, text is limited with quotes, and numbers are entered without any delimiters.

In the following example, number 100 will appear instead of “hello!!!”:

```
Edit1 = 100;
```

Variables

Variables can be divided into text variables and number variables. Text variables store a string of characters and number variables store numbers. A variable stores data during playback of the entire Flash animation.

In the following example you can use ActionScript to calculate area of a rectangle.

Launch Alligator Flash Designer, draw and edit field with the “Edit field” tool. The edit field will appear as Edit1, Edit2 etc. Choose “Frame” > “ActionScript” and paste the following code:

```
width = 20;  
height = 30;  
result = width * height;  
Edit1 = result;
```

Press F9 to run the movie. "600" will be displayed in the edit field.

You can use numeric variables in various mathematical equations, for example to calculate surface area of a triangle:

```
result = 0.5 * width * height;
```

or more complex operation

```
result = 1.45 + (width * height + 20) * 100;
```

Text variables

Text variables can be concatenated with + operator:

```
width = 20;  
height = 30;  
result = width * height;  
text = "Area: " + result + " m2";  
Edit1 = text;
```

The result displayed in the edit field is "Area: 600 m2".

Variable name must start with an alphanumeric character: a to z and can include numbers (not as the first character) and underscore sign `_`. Variable names must not include any national characters.

Correct variables:

variable1, my_variable

Incorrect variables

1variable (starts with a number)

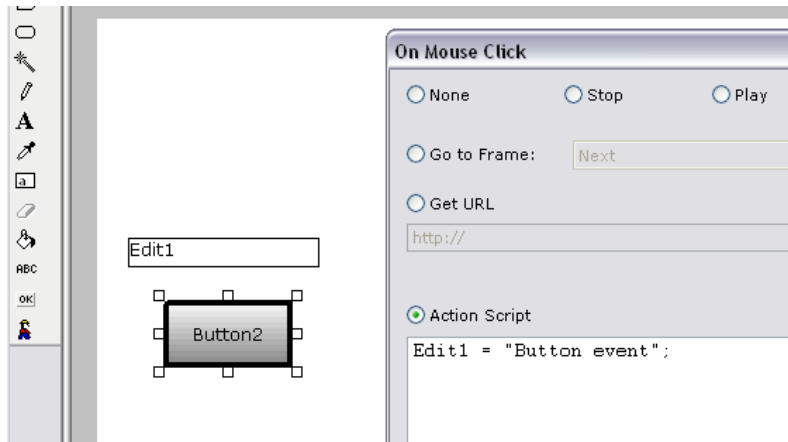
íuvariable (includes diacritics)

Button scripts

ActionScript can be executed in response to mouse events (click, mouse over, mouse out, mouse up). To define button script, select an object and choose "Action", then one of the commands: "On Click", "On Over", "On Out", "On Up". In the dialog window select "ActionScript" and enter the script.

To define a simple event, open a new Flash project and draw 2 objects: edit field Edit1 and Button2. Select Button2 and choose "Action" > "On Click". Enter the following code and click OK

```
Edit1 = "Button event";
```



Press F9 to run preview. Click Button2 to execute the code; "Button event" will be displayed in the edit field.

Mouse over and mouse out event.

choose "Action" > "On Over" and enter the code:

```
Edit1 = "Mouse over";
```

choose "Action" > "On Out" and enter the code:

```
Edit1 = "";
```

this command will erase the content of Edit1.

Conditionals

The following instruction checks the variable value and executes part of the code if the condition is met

Syntax:

```
if (condition)
{
  .. execute code
}
```

Example:

```
width = 20;
height = 30;
result = width * height;
if (result > 500)
{
  text = "Area > 500";
}
Edit1 = text;
```

If the area is larger than 500, "Area > 500" will be displayed in the Edit1 field.

Else instruction

Else command executes the code, if the condition is not met:

```
width = 20;
height = 30;
result = width * height;
if (result > 500)
{
  text = "What a large area";
}
else
{
  text = "What a small area";
}
Edit1 = text;
```

If the result is larger than 500, command text = "What a large area" will be executed, otherwise text = "What a small area" will be executed.

Loops

Loop will execute the same code several times, each time with increased (or decreased) specific variable, enabling the same calculation for several variable values.

For instruction

Syntax

```
for( initial value ; continuation condition ; increasing command )
{
    instructions will be repeated in the loop
}
```

Draw Edit1 text field, choose "Frame" > "ActionScript" and enter the following script:

```
text = "Even numbers: ";
for( i = 2 ; i < 10 ; i = i + 2 )
{
    text = text + i + " ";
}
Edit1 = text;
```

The code will display Even numbers: 2 4 6 8

While instruction

While loop is another type of loop::

```
while( condition )
{
    instructions will be repeated in the loop
}
```

In this case, you have to enter the command for initiating variable and increasing its value so the loop can end.

```
text = "Even numbers: ";
i = 2;
while( i < 10 )
{
    text = text + i + " ";
    i = i + 2;
}
Edit1 = text;
```

Functions

Function is a part of code that can be stored in a memory and executed as one of the ActionScript commands. Function can have parameters and return a value calculated inside the function.

In the following example, a function calculating rectangle area surface will be defined and executed.

```
function area(width,height)
{
    result = width * height;
    return result;
}
```

```
Edit1 = area(20,30);
```

Function definition must be preceded by 'function' word. The name of the function must obey the same rules as variable names, e.g. it can not start with a numerical and it can include letters, numbers and underscore sign.

The code of a function is entered between brackets { }.

The last command is `return` returning calculated value.

Arrays

Array is a variable with several values. Array index is specified in brackets [].
For example, create an array with female names:

```
names = new Array();  
  
names[0] = "Julia";  
names[1] = "Maria";  
names[2] = "Sandra";  
  
Edit1 = names[2];
```

Unlike numeric and text variables, arrays must be created before used.

Array can be initialized directly with the command **new**.

```
names = new Array("Julia", "Maria", "Sandra");  
Edit1 = names[2];
```

Objects

Objects, also referred to as classes, are similar to variables, although their structure allows you to store variables referred to as object attributes and functions referred to as object methods.

For example all Sprite objects are of MovieClip class. They include `_x` and `_y` attributes which is the upper and left position of a sprite. A sprite object can be moved during the playback by modifying these attributes.

Attributes and methods must be referenced with a dot between the object name and the method or between the object name and the attribute name.

Draw Sprite1 object, draw circle inside the Sprite. Exit the sprite, choose "Frame" > "ActionScript" and enter the code:

```
Sprite1._x = 0;  
Sprite1._y = 0;
```

Press F2 to change the sprite name so it is identical to the name used in the code, in this case it must be "Sprite1". Press F9 to execute the code. The sprite will move to the upper left corner of your animation.

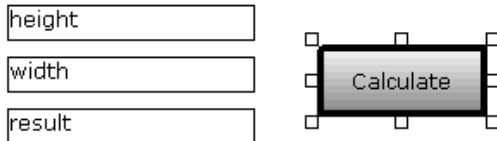
Use **new** command to create objects:

```
variable = new ObjectType( parameters );
```

Edit fields

Edit fields, usually Edit1, Edit2 etc. can be used for displaying variables and entering data.

Create a new Flash project and draw 3 edit fields: Edit1, Edit2 and Edit3, and Button4 button. Select each field and press Enter to open preferences. Name the fields accordingly height, width and result, and name the button Calculate:



Select the Calculate button, choose "Action" > "On Click", enter the following code:

```
result = height * width;
```

Press F9 and try some calculations by entering input data and clicking Calculate button.

Multiline fields

Text fields are a single line as a default. Select the field and press Enter to modify the field parameters. Check Multiline to accept return key in the edit field.

New line mark

Use "\n" string inside the string variable to break the text into lines. Create the text field and extend it vertically to contain several lines, check the Multiline option.

Enter the following frame code:

```
Edit1 = "Line 1\nNew line\nAnother line";
```

Operators

Operators are mathematical or logical operation commands for variables and numbers.

+
add values

-
subtract values

/
divide values

multiply values

%
modulo, remainder of the division
e.g.

```
Edit1 = 10 % 3;
```

the result is 1

Comparison operators

Operators used for conditional commands, returning true or false value.

<
less, returns true, if the first parameter is less than the second parameter

>
greater, returns true, if the first parameter is greater than the second parameter

<=
less or equal to, returns true, if the first parameter is less or equal to the second parameter

>=
greater or equal to, returns true, if the first parameter is greater or equal to the second parameter

==
equality, returns true, if the parameters are identical, and false if they differ

===
exact equality, returns true if the parameters are identical and of the same type, and false if they differ

!=
inequality, returns true if the parameters differ, false if they are identical

!
logical negation, converts true to false or false to true

Logical operators

Operators used in conditional commands to combine true or false values.

&&

logical AND, returns true if both conditions are met, otherwise returns false

||

logical OR, returns true if one of the conditions is met, returns false if both conditions are not met

Bit operators

Operators for binary numbers

Example: decimal and binary numbers

```
1 = 00000001
2 = 00000010
3 = 00000011
4 = 00000100
8 = 00001000
16 = 00010000
32 = 00100000
```

&

bitwise AND operator, if for both parameters in a specific location the bit has a value of 1, the result is also 1.

example

```
1 & 2 = 0
1 & 3 = 1
```

|

bitwise OR operator, if both bits on specific location have a value of 1, the resulting bit is also 1

example

```
1 | 2 = 3
```

^

Xor, if bits in a specific location are equal, the result is a bit 0, if they differ, the result is 1

example

```
1 ^ 3 = 2
```

~

Bit negation, inverse the bit value for each position

Assignment operators

Assignment operator calculates value on the right-hand side of the equal sign and stores it in the variable on the left-hand side of the equal sign.

```
Edit1 = x + 1;
```

Also the following operators are available:

+=

adds value on the right-hand side of the equal sign to the variable

```
x += 4;
```

is equivalent to

```
x = x + 4;
```

or

```
Edit1 += "additional text";
```

is equivalent to

```
Edit1 = Edit1 + "additional text";
```

-=

subtracts value on the right-hand side of the equal sign from the variable

```
x -= a + 2;
```

is equivalent to

```
x = x - (a + 2);
```

***=**

multiplies value on the right-hand side by the variable

```
x *= a + 2;
```

is equivalent to

```
x = x * (a + 2);
```

/=

divides a variable by the value on the right-hand side of the equal sign

```
x /= a + 2;
```

is equivalent to

```
x = x / (a + 2);
```

%=

calculates variable modulo and assigns a result

&=

adds bit value to the current variable and assigns a result

|=

executes OR operation of the value and the current variable and assigns a result

^=

executes XOR operation of the value and the current variable and assigns a result

>>=

moves variable bits to the right-hand side and assigns a result

<<=

moves variable bits to the left-hand side and assigns a result

Internal functions

escape(expression:String) : String

Changes a string to a form that can be transmitted as HTTP call arguments, i.e. all non-alphanumeric characters are changed to % code

unescape(x:String) : String

Changes string from the HTTP call arguments to normal text

getTimer() : Number

Returns milliseconds from the time when the movie clip has started

getURL(url:String)

Opens an internet link

getURL(url:String, window:String)

Opens an internet link with target parameter

Example

```
getURL("http://www.selteco.com", "_blank");
```

opens www.selteco.com address in a new window

Link parameters can be specified after ? sign

```
getURL("http://www.selteco.com?param1=value1&param2=value2", "_blank");
```

gotoAndPlay(scene:String)

jumps to the frame with specified name

```
gotoAndPlay("Frame 2");  
Sprite1.gotoAndPlay("Frame 2");  
_root.gotoAndPlay("Frame 2");
```

gotoAndPlay(frame:Number)

jumps to the frame by physical frame index, number of frames depends on movie clip frequency, usually 20 frames per second.

gotoAndStop(scene:String)

jumps to the frame with specified name and stops

```
gotoAndStop("Frame 2");  
Sprite1.gotoAndStop("Frame 2");  
_root.gotoAndStop("Frame 2");
```

gotoAndStop(frame:Number)

jumps to the frame with frame index and stops

isNaN(expression:Object) : Boolean

Returns true if value is non-numerical, false if it is numerical

Number(text: String)

Converts string value to numeric

Example

```
Edit3 = Number(Edit1) + Number(Edit2);
```

parseFloat(string:String) : Number

Changes string to number

Example

```
Edit1 = parseFloat("3.5e6");
```

3500000 is obtained

parseInt(expression:String [, base:Number]) : Number

Changes an integer in the specific base system: binary or hexadecimal

Examples

```
Edit1 = parseInt("101",2);
```

setInterval(functionName:Function, interval:Number) : Number

Call specific function in specified time interval in milliseconds

Example: draw Edit1 field and paste the frame code

```
Edit1 = 0;
```

```
function myInterval()  
{  
  Edit1 = Edit1 + 1;  
}
```

```
setInterval(myInterval,100);
```

this

Variable used inside the function refers to the current owner of the function

```
function Constructor()  
{  
  this.attribut1 = "some text";  
}
```

```
o = new Constructor();
```

```
Edit1 = o.attribut1;
```


typeof(expression) : String

Return variable type

String: string
Sprite: movieclip
Button: object
Text field: object
Number: number
Boolean: boolean
Object: object
Function: function
Null value: null
Undefined value: undefined

Examples:

```
s = "12345";  
Edit1 = typeof(s);
```

or

```
s = 12345;  
Edit1 = typeof(s);
```

or

```
Edit1 = typeof(_root);
```

or

Draw Sprite1 with a circle inside and Edit1 text field, paste the frame code:

```
Sprite1.onPress = function ()  
{  
    Edit1 = "press";  
}  
Edit1 = typeof(Sprite1.onPress);
```

undefined

Not a number

```
if (someUndefinedVariable == undefined)  
{  
    Edit1 = "variable does not exist";  
}
```

MovieClip

MovieClip is the most commonly used class. All Group or Sprite type objects exist as MovieClips. The main Flash movie is defined as a `_root` object.

MovieClip.alpha : *Number*

Object transparency from 0 to 100 %

MovieClip.currentframe : *Number*

Number of current physical frame during playback.

MovieClip.droptarget : *String*

Name of other Sprite, the current Sprite is dragged and dropped into

MovieClip.enabled : *Boolean*

True if the Sprite can receive mouse events, otherwise the Sprite is blocked

MovieClip.focusEnabled : *Boolean*

True value if the Sprite can receive key events, otherwise Sprite is blocked

MovieClip.focusrect : *Boolean*

If true, Sprite is enclosed with a rectangle, which means that it accepts keyboard events

MovieClip.framesloaded : *Number*

Number of Sprite frames currently downloaded from the internet, if the Sprite is loaded from an external file

MovieClip.height : *Number*

Sprite height in pixels

MovieClip.hitArea : *MovieClip*

Indicator of a different Sprite, if the Sprite has a different object acting as an active button field

MovieClip.lockroot : *Boolean*

If the subclip is loaded from an external file `_lockroot = true`, references from the subclip to the `_root` object are related to the subclip object not the main clip that loads a subclip.

MovieClip.menu : *ContextMenu*

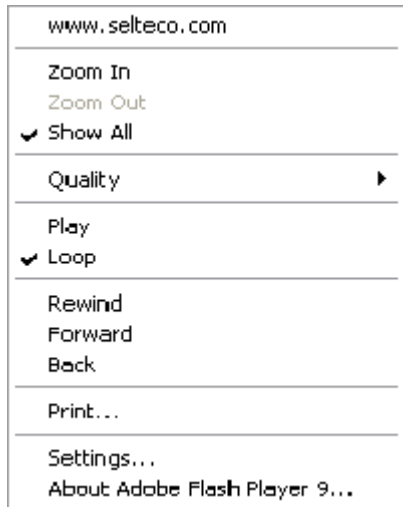
Context menu objects (right mouse button) assigned to a specific Sprite. File must be exported to Flash Player 8 or higher.

In this example, menu item for the main clip is added

```
function goOnTheWeb()
{
    getURL("http://www.selteco.com", "_blank");
}

mymenu = new ContextMenu();
mymenu.customItems.push(new ContextMenuItem("www.selteco.com", goOnTheWeb));

_root.menu = mymenu;
```



MovieClip._name : *String*
Sprite instance name

```
Edit1 = Spritel._name;
```

MovieClip._parent : *MovieClip*
Indicator for parent Sprite including this sprite

MovieClip._quality : *String*
Movie quality:

"LOW" Low quality, fast playback

"MEDIUM" Medium quality, bitmaps and text are not optimized

"HIGH" Default quality

"BEST" High quality, bitmaps are optimized

MovieClip._rotation : *Number*
Sprite angle of rotation

MovieClip._soundbuftime : *Number*
Delay in seconds, before buffered sound is played

MovieClip.tabEnabled : *Boolean*
True if the Sprite belongs to the chain of Tab switching

MovieClip.tabChildren : *Boolean*
True if the Sprite children will be included in the Tab switching cycle

MovieClip.tabIndex : *Number*
Entry number for Tab switching

MovieClip._target : *String*
Absolute sprite path

```
Edit1 = Spritel._target;
```

MovieClip._totalframes : *Number*
Total number of sprite frames

MovieClip.trackAsMenu : *Boolean*

If true, specific Sprite accepts all events of mouse up, even outside the sprite area

MovieClip._url : *String*

Internet address from which the Sprite is loaded

MovieClip.useHandCursor : *Boolean*

If false and when mouse action is defined, the Sprite will have an arrow cursor instead of a link cursor

```
Sprite1.useHandCursor = false;
```

MovieClip._visible : *Boolean*

Defines if Sprite is visible or not

Show sprite:

```
Sprite1._visible = true;
```

Hide sprite:

```
Sprite1._visible = false;
```

MovieClip._width : *Number*

MovieClip._height : *Number*

Sprite width and height in pixels

MovieClip._x : *Number*

MovieClip._y : *Number*

Sprite location inside the parent

MovieClip._xmouse : *Number*

MovieClip._ymouse : *Number*

Cursor location

```
function readmouse()  
{  
    Edit1 = _root._xmouse + ", " + _root._ymouse;  
}
```

```
setInterval(readmouse,10);
```

MovieClip._xscale : *Number*

MovieClip._yscale : *Number*

Sprite x and y scale in percents, default 100

MovieClip.createEmptyMovieClip(instanceName:String, depth:Number) : *MovieClip*

Creates new and empty Sprite object with instanceName and specific depth, higher depth hides the object under other objects

MovieClip.createTextField(instanceName:String, depth:Number, x:Number, y:Number, width:Number, height:Number)

Creates empty text field with instanceName, specific depth and dimensions.

Dimensions are specified in pixels.

```
_root.CreateTextField("EditField1",10,20,20,500,40);  
EditField1.text = "My text field";
```

MovieClip.duplicateMovieClip(newname:String, depth:Number) : MovieClip

Duplicates Sprite and places it on specific depth

```
Sprite1.duplicateMovieClip("Sprite2",100);  
Sprite2._x = Sprite1._x + 10;  
Sprite2._y = Sprite1._y + 10;
```

MovieClip.getBounds(targetCoordinateSpace:Sprite) : Object

Returns rectangle with items visible inside the Sprite in relation to targetCoordinateSpace object or in relation to each other, if the parameter is not specified

```
rect = Sprite1.getBounds();  
Edit1 = rect.yMin + ", " + rect.yMax + ", " + rect.xMin + ", " + rect.xMax;
```

MovieClip.getBytesLoaded() : Number

Returns loaded bytes if the file is downloaded from the internet

MovieClip.getBytesTotal() : Number

Returns total Sprite bytes

MovieClip.getDepth() : Number

Returns sprite depth

MovieClip.getInstanceAtDepth(depth:Number) : MovieClip

Returns pointer to the sprite on specific depth

MovieClip.getNextHighestDepth() : Number

Returns any depth on which the new Sprite can be located Each depth may include a single object.

MovieClip.getSWFVersion() : Number

Returns version number for which the specific sprite is intended, if loaded from an external file

MovieClip.getTextSnapshot() : String

Creates a string from the content of text fields within the Sprite

MovieClip.getURL(URL:String [,window:String, method:String])

Open link

URL: internet address e.g. <http://www.selteco.com>

window: `_blank` opens new browser window, `_self` opens link in the current window

method: string POST or GET, if link has parameters after the `?`, default value is GET

MovieClip.globalToLocal(point:Object)

Changes global point coordinates (x and y) to the coordinates within the Sprite

MovieClip.gotoAndPlay(framename:String)

Jumps to the specific frame (e.g. Frame2) and starts playback

MovieClip.gotoAndStop(framename:String)

Jumps to specific frame (e.g. Frame2) and stops playback

MovieClip.hitTest(target:Object)

Returns true, if the Sprite overlaps (contacts) the Sprite specified as a target parameter

MovieClip.hitTest(x:Number, y:Number, shapeFlag:Boolean)

Returns true, if point (x and y) touches the Sprite, shapeFlag parameter defines, if the visible sprite components or the whole rectangle is taken for the calculations

MovieClip.loadMovie(url:String)

Loads SWF, FLV or JPG file

MovieClip.loadVariables(url:String)

Loads variable from the text file, text file must contain variable in the same form as the URL addresses

.txt file example

```
var1="hello"&var2="goodbye"
```

MovieClip.localToGlobal(point:Object)

Changes point coordinates (x and y) inside the Sprite to global coordinates

MovieClip.nextFrame()

Jumps to the next frame

MovieClip.play()

Starts movie playback

MovieClip.prevFrame()

Jumps to the previous frame

MovieClip.removeMovieClip()

Removes Sprite created using MovieClip.duplicateMovieClip() command

MovieClip.setMask(target:Sprite)

Sets Sprite as a mask for other Sprite

MovieClip.startDrag()

Starts dragging a Sprite with a mouse

MovieClip.startDrag([lock:Boolean, [left:Number, top:Number, right:Number, bottom:Number]])

Starts dragging a Sprite with a mouse limiting available area with left, top, right and bottom values
Lock parameter makes the Sprite center correspond to a mouse cursor

MovieClip.stop()

Stops Sprite playback

MovieClip.stopDrag()

Stops dragging a Sprite with a mouse

MovieClip.swapDepths(depth:Number)**MovieClip.swapDepths(target:String)**

Swaps 2 Sprite depths at the specific depth or with a specific name

MovieClip.unloadMovie()

Removes the sprite, dynamically loaded from an external file, from memory

Drawing Sprites

MovieClip.beginFill(rgb:Number)

MovieClip.beginFill(rgb:Number, alpha:Number)

Specifies fill Color using hexadecimal code and alpha opacity

MovieClip.beginGradientFill(fillType:String, Colors:Array, alphas:Array, ratios:Array, matrix:Object)

Specifies gradient fill between different Colors

fillType: "linear" or "radial"

Colors: Color array, max. 8 items

alphas: opacity array (from 0 transparent to 255 full opacity), max. 8 items

ratios: gradient fill Color position array, values from 0 to 255, max. 8 items

MovieClip.clear()

Clears the drawing made with drawing commands

MovieClip.curveTo(cx:Number, cy:Number, x:Number, y:Number)

Draws Bezier curve from point x,y and control point cx and cy

MovieClip.endFill()

Closes started lines and fills the curve with Color specified with MovieClip.beginFill() or MovieClip.beginGradientFill() commands.

MovieClip.lineStyle(thickness:Number, rgb:Number, alpha:Number)

Sets new line style with specific thickness rgb Color and alpha opacity

MovieClip.lineTo(x:Number, y:Number)

Draws straight line to the x, y point

MovieClip.moveTo()

Sets new drawing start position

Sprite event support

The below events can be assigned to the defined functions

MovieClip.onData : *Function*

Called while downloading data

MovieClip.onDragOut : *Function*

Called when mouse button is pressed within the sprite and moved outside its area

MovieClip.onDragOver : *Function*

Called when the mouse button is pressed outside the sprite and moved into its area

MovieClip.onEnterFrame : *Function*

Called before displaying each physical frame

MovieClip.onKeyDown : *Function*

Called after pressing `Key.getCode()` and `Key.getAscii()` in order to obtain a key code

MovieClip.onKeyUp : *Function*

Called when the key is released

MovieClip.onKillFocus : *Function*

Called when the Sprite cannot accept keyboard events

MovieClip.onLoad : *Function*

Called before the Sprite appears in the clip for the first time

MovieClip.onMouseDown : *Function*

Called when left mouse button is used

MovieClip.onMouseMove : *Function*

Called by mouse movement

MovieClip.onMouseUp : *Function*

Called when the mouse button is released

MovieClip.onPress : *Function*

Called when left mouse button is used

MovieClip.onRelease : *Function*

Called when the mouse button is released

MovieClip.onReleaseOutside : *Function*

Called when mouse button is pressed within the sprite, moved outside and released

MovieClip.onRollOut : *Function*

Called when mouse cursor is moved outside the sprite area

MovieClip.onRollOver : *Function*

Called when mouse cursor is moved into the sprite area

MovieClip.onSetFocus : *Function*

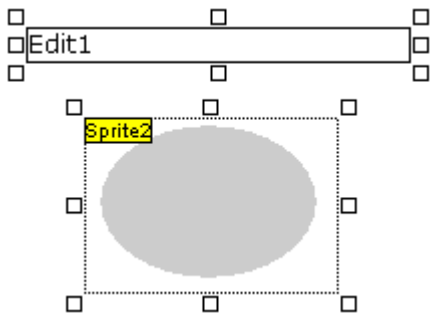
Called when sprite sets focus

MovieClip.onUnload : *Function*

Called when sprite is unloaded from the clip

Example:

Draw Edit1 field and Sprite2, draw circle inside the sprite:



paste the following code in the ActionScript frame:

```
Sprite2.onPress = function ()
{
    Edit1 = "onPress";
}

Sprite2.onRelease = function ()
{
    Edit1 = "onRelease";
}

Sprite2.onRollOut = function ()
{
    Edit1 = "onRollOut";
}

Sprite2.onRollOver = function ()
{
    Edit1 = "onRollOver";
}
```

Array

This class represents array type, arranged variables that can be referred to using index in brackets [].

Array.concat(array1, array2, ...)

Joins arrays

```
arr1 = new Array();
arr1[0] = "Julia";
arr1[1] = "Maria";

arr2 = new Array();
arr2[0] = "Sandra";
arr2[1] = "Pamela";

arr3 = arr1.concat(arr2);

Edit1 = arr3[3];
```

Array.join(separator)

Joins array items as a string

```
arr1 = new Array();
arr1[0] = "Julia";
arr1[1] = "Maria";
arr1[2] = "Pamela";
Edit1 = arr1.join(", ");
```

Array.pop()

Removes last array item and returns its value

```
arr1 = new Array();
arr1[0] = "Julia";
arr1[1] = "Maria";
arr1[2] = "Pamela";
Edit1 = arr1.pop();
```

Array.push()

Adds item at the end of an array and returns new number of items

```
arr1 = new Array();
arr1[0] = "Julia";
arr1[1] = "Maria";
arr1.push("Pamela");
```

Array.reverse()

Reverses the order of array items

```
arr1 = new Array();
arr1[0] = "Julia";
arr1[1] = "Maria";
arr1[2] = "Pamela";
arr1.reverse();
Edit1 = arr1.join(", ");
```

Array.shift()

Removes the first array item and returns its value

```
arr1 = new Array();
arr1[0] = "Julia";
arr1[1] = "Maria";
arr1[2] = "Pamela";
Edit1 = arr1.shift();
```

Array.slice(start, end)

Slices part of an array from the start item to the end item (not included) and returns a new array

```
arr1 = new Array();
arr1[0] = "Julia";
arr1[1] = "Maria";
arr1[2] = "Sandra";
arr1[3] = "Pamela";

arr2 = arr1.slice(1,3);

Edit1 = arr2.join(" , ");
```

Array.sort()

Sorts array items

```
arr1 = new Array();
arr1[0] = "Maria";
arr1[1] = "Sandra";
arr1[2] = "Pamela";
arr1[3] = "Julia";

arr1.sort();

Edit1 = arr1.join(" , ");
```

Array.sort(option)

Available option parameter values

- 1 or Array.CASEINSENSITIVE, case insensitive
- 2 or Array.DESENDING, reversed order (descending)
- 4 or Array.UNIQUESORT, sorting error, identical values
- 8 or Array.RETURNINDEXEDARRAY, returns indexed arrays without sorting original array
- 16 or Array.NUMERIC, numerical values in array, otherwise algorithm will sort 100 before 99, since 1 is before 9

Sorting function syntax

```
function sort(a, b)
{
    ... compare a and b
    return 1 , 0 or -1
}
```

Array.sort(compareFunction)

Sort compareFunction must return 0, when items are identical, -1 when item a is lesser than item b, 1 when item b is lesser than item a.

```
arr1 = new Array();
arr1[0] = 30;
arr1[1] = 4;
arr1[2] = 1;
arr1[3] = 16;

function sort(a,b)
{
    if(a<b) return -1;
    if(a>b) return 1;
    return 0
}

arr1.sort(sort);

Edit1 = arr1.join(", ");
```

Array.sortOn(fieldName)

Sorts items in relation to the array field.

Array fields can be used as a sorting value:

```
arr1 = new Array();
arr1[0] = new Object(); arr1[0].name = "Maria"; arr1[0].age = 24;
arr1[1] = new Object(); arr1[1].name = "Sandra"; arr1[1].age = 15;
arr1[2] = new Object(); arr1[2].name = "Pamela"; arr1[2].age = 31;
arr1[3] = new Object(); arr1[3].name = "Julia"; arr1[3].age = 22;

arr1.sortOn("age");

Edit1 = arr1[0].name + ", " + arr1[1].name + ", " + arr1[2].name + ", " +
arr1[3].name ;
```

Array.splice(start, count)

Removes array items

```
arr1 = new Array();
arr1[0] = "Julia";
arr1[1] = "Maria";
arr1[2] = "Sandra";
arr1[3] = "Pamela";
arr1.splice(1,2);

Edit1 = arr1.join(", ");
```

Array.toString()

Converts array to a string.

```
arr1 = new Array();
arr1[0] = 1;
arr1[1] = 10;
arr1[2] = 100;
arr1[3] = 1000;

Edit1 = arr1.toString();
```

Array.unshift()

Adds new items at the beginning.

```
arr1 = new Array();
arr1[0] = "Julia";
arr1[1] = "Maria";

arr1.unshift("Sandra", "Pamela" );

Edit1 = arr1.join(", ");
```

Array.length

Returns number of items in the table

```
names = new Array();

names[0] = "Julia";
names[1] = "Maria";
names[2] = "Sandra";

Edit1 = names.length;
```

Key

This Class is responsible for keyboard support.

Before keyboard actions are supported by the clip, it must be activated in the browser by clicking a mouse button in the movie clip area.

Key.addListener(newListener:Object)

Adds listener for support of pressing and releasing buttons

Example

```
myListener = new Object();

myListener.onKeyDown = function ()
{
    Edit1 = "Key pressed";
}
myListener.onKeyUp = function ()
{
    Edit1 = "Key released.";
}

Key.addListener(myListener);
```

Key.getAscii() : Number

Returns ASCII of the last pressed button

Key.getCode() : Number

Returns code of the last pressed button

Key.isDown(keycode:Number) : Boolean

Returns true if specific key is pressed

Key.isToggled(keycode:Number) : Boolean

Returns true if Num Lock or Caps Lock is pressed.

Key.removeListener(listener:Object) : Boolean

Removes listener

Key codes

To simplify, Key class includes attributes corresponding to the codes of the most common keys

```
Key.BACKSPACE = 8
Key.CAPSLOCK = 20
Key.CONTROL = 17
Key.DELETEKEY = 46
Key.DOWN = 40
Key.END = 35
Key.ENTER = 13
Key.ESCAPE = 27
Key.HOME = 36
Key.INSERT = 45
Key.LEFT = 37
Key.PGUP = 33
Key.PGDN = 34
```

Example: moving the sprite with keys.

Draw Sprite1, place a circle inside, exit the sprite and enter the frame code:

```
myListener = new Object();

myListener.onKeyDown = function ()
{
    if(Key.isDown(Key.LEFT)) Spritel._x = Spritel._x - 5;
    if(Key.isDown(Key.RIGHT)) Spritel._x = Spritel._x + 5;
    if(Key.isDown(Key.UP)) Spritel._y = Spritel._y - 5;
    if(Key.isDown(Key.DOWN)) Spritel._y = Spritel._y + 5;
}

Key.addListener(myListener);
```

Mouse

Mouse.addListener(newListener:Object)

Adds listener for mouse events support

Mouse.hide() : *Number*

Hides mouse cursor, returns true if cursor is visible

Mouse.removeListener((listener:Object) : *Boolean*

Removes listener added by addListener().

Mouse.show() : *Number*

Shows cursor, returns if cursor is visible before function call

MouseListener.onMouseDown : *Function*

Function called when pressing mouse button

MouseListener.onMouseMove : *Function*

Function called when mouse is moved

MouseListener.onMouseUp : *Function*

Function called when mouse button is released

MouseListener.onMouseWheel : *Function*

Function called when mouse scroll is rotated

Example display of mouse position in Edit1 field

```
myListener = new Object();

myListener.onMouseMove = function ()
{
  Edit1 = _root._xmouse + ", " + _root._ymouse;
}

Mouse.addListener(myListener);
```


Button

Button class corresponds to buttons created with “Button” tool

By default buttons are named: ButtonObject1, ButtonObject2 etc. Select the button and press F2 to display button name.

In order to define button for ActionScript press F2 and check “ActionScript object” option.

Button._alpha : *Number*

Button opacity: 0 to 100 percent

Button.enabled : *Boolean*

Specifies if the button accepts mouse and keyboard events

Example: blocking button before clicking, draw Button1 and paste the frame code:

```
ButtonObject1._alpha = 20;  
ButtonObject1.enabled = false;
```

Button._height : *Number*

Button._width : *Number*

Define button dimensions

Button._name : *String*

Button object name

Button._rotation : *Number*

Button rotation in relation to upper left corner

Button.tabEnabled : *Boolean*

True if the button is within the chain of Tab switching

Button.tabIndex : *Number*

Entry number for Tab switching

Button._target : *String*

Absolute button path

```
Edit1 = ButtonObject1._target;
```

Button.trackAsMenu : *Boolean*

If true, specific button accepts all mouse release events, even outside the sprite area

Button.useHandCursor : *Boolean*

If false, button will have an arrow cursor instead of a link cursor, if the mouse action is defined for the specific button

```
ButtonObject1.useHandCursor = false;
```

Button._x : *Number*

Button._y : *Number*

Shifting button in relation to the current position, default 0,0

Button._xmouse : *Number*
Button._ymouse : *Number*
Mouse cursor position on a button

Button._visible : *Boolean*
Specifies if the button is visible

Button.onDragOut : *Function*
Called when the mouse button is pressed within the button and the cursor is dragged outside the area

Button.onDragOver : *Function*
Called when the mouse button is pressed outside the button and the cursor is dragged over the area

Button.onKeyDown : *Function*
Called after pressing `Key.getCode()` and `Key.getAscii()` in order to obtain a key code

Button.onKeyUp : *Function*
Called when the key is released

Button.onKillFocus : *Function*
Called when the button cannot accept keyboard events

Button.onPress : *Function*
Called when pressing left mouse button on the button

Button.onRelease : *Function*
Called when the mouse button is released

Button.onReleaseOutside : *Function*
Called when the mouse button is pressed within the button, and the cursor is dragged outside the area and the mouse button is released

Button.onRollOut : *Function*
Called when the mouse cursor rolls out of the button

Button.onRollOver : *Function*
Called when the mouse pointer rolls over the button

Button.onSetFocus : *Function*
Called when the button accepts keyboard events

Math

Math class provides mathematical functions and fixed values.

Math.abs(x:Number) : *Number*
Absolute number value

```
Edit1 = Math.abs(-1.45);
```

Math.acos(x:Number) : *Number*
Calculates acosine.

Math.asin(x:Number) : *Number*
Calculates asine

Math.atan(x:Number) : *Number*
Calculates atangent.

Math.atan2(y:Number, x:Number) : *Number*
Calculates angle from x,y point to the x axis in radians (from -pi to pi)

Math.ceil(x:Number) : *Number*
Rounds up a number to the next integer

Math.cos(x:Number) : *Number*
Calculates cosine

Math.exp(x:Number) : *Number*
exp function

Math.floor(x:Number) : *Number*
Rounds down a number to the next integer

Math.log(x:Number) : *Number*
Calculates natural logarithm

Math.max(x1:Number, x2:Number) : *Number*
Returns greater of 2 numbers

Math.min(x1:Number, x2:Number) : *Number*
Returns lesser of 2 numbers

Math.pow(x:Number, y:Number) : *Number*
Returns number raised to the y power

Math.random() : *Number*
Returns random number from 0.0 to 1.0.

Math.round(x:Number) : *Number*
Rounds up to the next integer

Math.sin(x:Number) : *Number*
Calculates sine

Math.sqrt(x:Number) : *Number*
Calculates square root

Math.tan(x:Number) : *Number*
Calculate tangent

Mathematical variables
Incorporated variables that can be used in calculations

Math.E : *Number*
Base for the natural logarithm (approx. 2.718).

Math.LN2 : *Number*
Natural logarithm of 2 (approx. 0.693).

Math.LOG2E : *Number*
approx. 1.442.

Math.LN10 : *Number*
Natural logarithm of 10 (approx. 2.302).

Math.LOG10E : *Number*
approx. 0.434

Math.PI : *Number*
PI (approx. 3.14159).

Math.SQRT1_2 : *Number*
Square root of 1/2 (approx. 0.707).

Math.SQRT2 : *Number*
Square root of 2 (approx. 1.414).

Example:

```
Edit1 = "Area of the circle with the radius of 5 is " + Math.PI *  
Math.pow(5,2);
```

Date

Date class represents an object with the current time or any time specified by the user.

UTC is a coordinated universal time, independent on seasons or time zone.
Local time is a standard time accounting daylight savings time and time zone.

new Date()

creates a new Date class object with current time

to display of a current year use the code

```
d = new Date();  
Edit1 = d.getFullYear();
```

new Date(year:Number, month:Number [, date:Number [, hour:Number [, minute:Number [, second:Number [, millisecond:Number]]]]])

Creates a new date object with specified time

year: year
month: month number from 0 to 11
date: day from 1 to 31
hour: hour from 0 to 23
minute: minute from 0 to 59
second: second from 0 to 59
millisecond: 1/1000 seconds from 0 to 999

Example: date 12 February 1990

```
mydate = new Date(1990, 1, 12);
```

Example: calculation of the number of days between 2 dates: 1 January 1980 and 14 March 2009

```
date1 = new Date(1980, 0, 1);  
date2 = new Date(2009, 2, 14);  
days = ( date2.getTime() - date1.getTime() ) / (1000 * 60 * 60 * 24);  
Edit1 = days;
```

new Date(timeValue:Number)

Creates Date class object with time specified in milliseconds, from 1 January 1970, UTC

Example: creating 3 seconds after 1 January 1970, UTC

```
d = new Date(3000);  
Edit1 = d;
```

Date.getDate() : Number

Returns day of the month

Date.getDay() : Number

Returns day of the week

Date.getFullYear() : *Number*
Returns 4 digit year

Date.getHours() : *Number*
Returns hour

Date.getMilliseconds() : *Number*
Returns milliseconds

Date.getMinutes() : *Number*
Returns minutes

Date.getMonth() : *Number*
Returns month

Date.getSeconds() : *Number*
Returns seconds

Date.getTime() : *Number*
Returns milliseconds from midnight 1 January 1970, UTC

Date.getTimezoneOffset() : *Number*
Returns time difference in seconds, UTC

Date.getYear() : *Number*
Returns year

Date.getUTCDate() : *Number*

Date.getUTCDay() : *Number*

Date.getUTCFullYear() : *Number*

Date.getUTCHours() : *Number*

Date.getUTCMilliseconds() : *Number*

Date.getUTCMinutes() : *Number*

Date.getUTCMonth() : *Number*

Date.getUTCSeconds() : *Number*

Date.getUTCYear() : *Number*

Identical functions, although they return time converted to the UTC

Date.setDate() : *Number*

Date.setFullYear() : *Number*

Date.setHours() : *Number*

Date.setMilliseconds() : *Number*

Date.setMinutes() : *Number*

Date.setMonth() : *Number*

Date.setSeconds() : *Number*

Date.setTime() : *Number*

Date.setYear() : *Number*

Functions, which modify time in Date object

Date.toString() : *String*
Returns time as a string

Date.UTC() : *Number*

Number of milliseconds between 1 January 1970, UTC and the time stored in the object

Variable classes

Arguments

Object representing list of function parameters

arguments.callee : *Function*

Pointer to the function called by the specific function

arguments.caller : *Function*

Pointer to the function being called

arguments.length : *Number*

Number of parameters

Example:

```
function getArgCount(param_arg1, param_arg2, param_arg3)
{
    return (arguments.length);
}
```

```
Edit1 = getArgCount("par1", "par2", "par3");
```

Boolean

Class represents Boolean type variable, i.e. true or false

Boolean.toString() : *String*

Returns text representation of a variable ("true" or "false")

Boolean.valueOf() : *Boolean*

Returns object value ("true" or "false")

Hexadecimal system

System used for specifying color values. RGB colors are stored as hexadecimal numbers (6 characters).

In hexadecimal notation, every digit instead of 10 values takes on a values between 0 to 15, numerals above the value of 9 are denoted with a,b,c,d,e,f or A,B,C,D,E,F.

To differentiate hexadecimal numbers from decimal numbers, they are preceded with 0x, otherwise hexadecimal numbers without any value above 9 might be confused with decimal number.

Examples of hexadecimal numbers and equivalent decimal values

```
0x2 = 2
0x9 = 9
0xF = 15
0x10 = 16
```

0x18 = 32
0xFF = 255

Color in computer graphics is defined using 3 values, corresponding to the intensity of red, green and blue. All Colors can be obtained by mixing primary Colors with correct ratio. Intensity of all Colors can have a value of 0 (no Color) to 255 (maximum brightness of a component Color). Maximum brightness of all Colors gives white, and no brightness gives black.

To code a single component Color, also referred to as a channel it is required to use 2 hexadecimal numbers. Maximum value is 255, i.e. 0xFF.

Color code is created by specifying hexadecimal number with 6 digits:

0xRRGGBB

where RR means intensity of red, GG - green and BB - blue.

example of Colors denoted in hexadecimal numbers:

0x000000 black
0xFFFFFFFF white
0xFF0000 red
0x00FF00 green
0x0000FF blue
0x808080 grey 50%

Color

Color class modifies Sprite color matrix. Sprite matrix allows to change Colors or transparency. Color gain is a percentage gain of specific channel of all graphic items within a Sprite, Color phase means adding all items to the current channel. E.g. by shifting red to 255 and other Colors to -255, all sprite items will be red, irrespective of their previous Color.

new Color(target:Sprite)

Creates new Color type object related to a specific Sprite

Color.getTransform() : Object

Downloads a current sprite Color matrix. It is an object incorporating the following attributes:

ra red gain percentage (-100 to 100).
rb red value shift (-255 to 255).
ga green gain percentage (-100 to 100).
gb green value shift (-255 to 255).
ba blue gain percentage (-100 to 100).
bb blue value shift (-255 to 255).
aa opacity gain percentage (-100 to 100).
ab opacity value shift (-255 to 255).

Default matrix include 100 gain and 0 shift for each channel.

Color.setTransform(matrix:Object)

Creates new sprite Color matrix

Color.getRGB() : Number

Returns numerical value corresponding to a Color code, incorporating rb, gb and bb values

Color.setRGB(0xRRGGBB:Number)

Sets Color phase in current matrix to the specific numerical value (stores in rb, gb and bb fields)

Example: draw Sprite and a grey circle inside. Exit the sprite and enter the frame code:

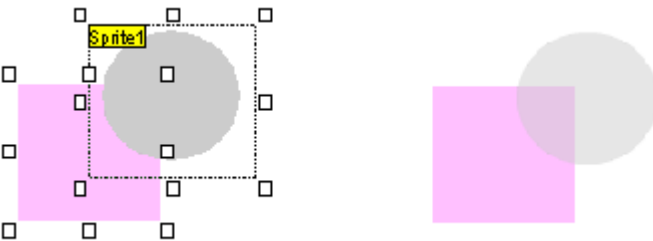
```
c = new Color(Sprite1);  
c.setRGB(0xFF0000);
```

Sprite will change Color to red:



Change to 50% opacity

```
c = new Color(Sprite1);  
m = c.getTransform();  
m.aa = 50;  
c.setTransform(m);
```



Number

Class represents numerical object

Number.toString()

Returns string

Number.valueOf()

Returns numerical value of an object

Number.MAX_VALUE

The highest possible numerical value, approx. 1.79E+308.

Number.MIN_VALUE

The lowest possible numerical value, approx. 5e-324.

Number.NaN

Expression value for comparing, if an object is a number, Not a Number (NaN).

Number.NEGATIVE_INFINITY

Positive infinity value

Number.POSITIVE_INFINITY

Negative infinity value

Sound

Class provides sound control

new Sound([target:Sprite])

Creates new Sound type object

When Sprite parameter is specified, object controls sounds within the Sprite

Sound.attachSound("idName":String)

Attaches sound with a specific ID to the object As a default it is a file name of the sound. Name can be changed in "Movie" > "Sounds"

Sound.getBytesLoaded()

If the sound is loaded from the file, returns downloaded bytes

Sound.getBytesTotal()

Returns total size of a sound file

Sound.getPan()

Returns balance value -100 (left channel) to 100 (right channel)

Sound.getTransform()

Returns object with the following attributes

ll: intensity of a left track in left speaker
lr: intensity of a left track in right speaker
rl: intensity of a right track in left speaker
rr: intensity of a right track in right speaker

values 0 to 100

Sound.getVolume()

Returns sound intensity 0 to 100

Sound.loadSound(url:String)

Downloads MP3 sound from the internet address

Sound.setPan(balance:Number)

Set balance from -100 to 100

Sound.setTransform(mixer:Object)

Define channel mixing

mixer is an object with ll, lr, rl and rr attributes
see also getTransform()

Sound.setVolume(volume:Number)

Sets sound intensity 0 to 100

Sound.start()

Starts sound playback from the beginning

start(secondOffset:Number)

Starts sound playback from the specific second

start(secondOffset:Number, loop:Number)

Starts sound playback from the specific second and with specific number of repetitions

Sound.stop()

Stop sound

Sound.duration

Sound duration in milliseconds

Sound.id3

Pointer to the ID3 object of a MP3 file, if present

Incorporates the following attributes

Sound.id3.comment	Comments
Sound.id3.album	Album
Sound.id3.genre	Genre
Sound.id3.songname	Song name
Sound.id3.artist	Artist
Sound.id3.track	Track number
Sound.id3.year	Release year

Attributes with names defined for ID3 specification are also available:

COMM	Comment
TALB	Album title
TBPM	Pace (per minute)
TCOM	Composer
TCOP	Copyrights
TDAT	Data
TEXT	Text author

Sound.position

Position of a current sound playback in milliseconds

Sound.onID3

Function called when ID3 is available

Sound.onLoad

Function called when sound is read from the file

Sound.onSoundComplete

Function called when sound playback is finished

String

Class represents string of alphanumeric characters. Characters in the string are indexed from 0 (first character of a string from the number smaller by 1 from the string length)

Letters in "Pamela" string will have the following indices:

```
0 P
1 a
2 m
3 e
4 l
5 a
```

String.length : *Number*

Number specifying the current number of characters in the string

String.charAt(x:Number) : *String*

Returns character in position x (from 0)

String.charCodeAt(x:Number) : *Number*

Returns ASCII of the character as a number in position x (from 0)

String.concat(val1:String, ... valN:String) : *String*

Creates and returns combination of a string with specified parameters.

```
stringA = "Hello";
stringB = "World";
Edit1 = stringA.concat(" ", stringB);
```

"Hello World" is displayed

String.fromCharCode(c1:Number,c2,...cN) : *String*

Returns string consisting of characters in ASCII

```
Edit1 = "dog"+String.fromCharCode(64)+"house.net";
```

dog@house.net is displayed

String.indexOf(substring:String) : *Number*

Returns index of a first instance of a substring from 0 at the beginning, or -1 if the substring is not found

String.indexOf(substring:String, startIndex:Number) : *Number*

Returns index of a substring instance, beginning from the startIndex

String.lastIndexOf(substring:String) : *Number*

Returns index of a last instance of a substring or -1 if the substring is not found

String.lastIndexOf(substring:String, startIndex:Number) : *Number*

Returns index of a last instance of a substring, beginning search from the startIndex

String.slice(start:Number) : *String*

Returns substring from the start character to the end character

String.slice(start:Number, end:Number) : String

Returns substring consisting of a start character and end character

String.split("delimiter"):String) : Array

Divides the string into substrings using delimiter and returns string array

```
s = "Maria:Pamela:Sandra";  
a = s.split(":");  
Edit1 = a[1];
```

String.substr(start:Number) : String

Returns substring from the start position to the end position, if start is a negative number, returns substring counted from the end

String.substr(start:Number, n:Number) : String

Returns n character substring from the start position

String.substring(start:Number, end:Number) : String

Returns substring from the start character to the end character, not including end character

String.toLowerCase() : String

Returns string of characters in lowercase without changing the original object

String.toUpperCase() : String

Returns string of characters in uppercase without changing the original object

Stage

Stage class correspond to the Flash movie clip located in the browser window

Stage.align : *String*

Alignment of a Flash object in the browser window

"T" top center

"B" bottom center

"L" center left

"R" center right

"TL" top left

"TR" top right

"BL" bottom left

"BR" bottom right

Stage.height : *Number*

Stage.width : *Number*

Width and height of a movie clip in pixels

Stage.scaleMode : *String*

Movie clip scale in the browser, available values: "exactFit", "showAll", "noBorder" and "noScale"

Stage.showMenu : *Boolean*

True if the whole context menu is available, false if the menu is limited

Stage.addListener(myListener:Object)

Adds listener checking if the movie clip is scaled in the browser

Stage.removeListener(myListener:Object) : *Boolean*

Removes listener added by the addlistener command

Stage.onResize : *Function*

Pointer to the function with notification about the movie clip scale in the browser. scaleMode parameter must be set to "noScale".

System

System

System.setClipboard(string:String) : Boolean

Copies string to the clipboard

System.showSettings()

Displays setting panel of a Flash player

System.showSettings(n:Number)

Displays setting panel in the n tab:

0 Privacy

1 Local Storage

2 Microphone

3 Camera

System.exactSettings : Boolean

True if the access settings apply to the specific domain, false if they apply to domains and subdomains in a specific domain.

System.useCodepage : Boolean

If false, Flash treats external text files as a Unicode, true if the files are stored in a code page. It applies to the files loaded by the LoadVars class.

System.onStatus : Function(genericError:Object)

Called in the case of a Flash plugin error

System.security

Object includes information on access permissions for SWF files run in the specific domain.

System.security.allowDomain("domain1":String, "domain2", ... "domainN")

Allows SWF files from specified domains to use this SWF file

System.security.allowInsecureDomain("domain":String)

Allows files from the domain to use this SWF file, if it is provided by the HTTPS

System.security.loadPolicyFile(url: String)

Download XML permission file from the specific internet address

File example:

```
<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>
```


System.capabilities

Object contains information about capabilities of the system, where Flash file is executed

System.capabilities.avHardwareDisable : *Boolean*

Is camera and microphone available

System.capabilities.hasAccessibility : *Boolean*

Is system equipped with accessibility features

System.capabilities.hasAudio : *Boolean*

Does system play sound

System.capabilities.hasAudioEncoder : *Boolean*

Does system store sound

System.capabilities.hasEmbeddedVideo : *Boolean*

Does system play video

System.capabilities.hasMP3 : *Boolean*

Does system play MP3 files

System.capabilities.hasPrinting : *Boolean*

Is printing available

System.capabilities.hasScreenBroadcast : *Boolean*

System.capabilities.hasScreenPlayback : *Boolean*

Does system use Flash Communication Server

System.capabilities.hasStreamingAudio : *Boolean*

Does system play stream audio

System.capabilities.hasStreamingVideo : *Boolean*

Does system play stream video

System.capabilities.hasVideoEncoder : *Boolean*

Does system store video in a file (e.g. from a camera)

System.capabilities.isDebugger : *Boolean*

Is plugin version featured with a debugging function

System.capabilities.language : *String*

System language as a two-letter code, e.g. "en" English

System.capabilities.localFileReadDisable : *Boolean*

Is access to system files blocked on a disk

System.capabilities.manufacturer : *String*

Flash plugin author

System.capabilities.os : *String*

Operating system

System.capabilities.pixelAspectRatio : *Number*

Ratio of physical pixels to logical pixels of a display, usually 1

System.capabilities.playerType : *String*

Plugin type, available values: "StandAlone", "External", "PlugIn" or "ActiveX".

System.capabilities.screenColor : *String*

Screen Color, available values: "Color", "gray", "bw".

System.capabilities.screenDPI : *Number*

Screen resolution in pixels per inch, usually 72

System.capabilities.screenResolutionX : *Number*

Horizontal screen resolution

System.capabilities.screenResolutionY : *Number*

Vertical screen resolution

System.capabilities.serverString : *String*

Variable string coded as an URL call

System.capabilities.version : *String*

Plugin version

TextField

Class corresponds to text fields.

For text fields, variable name must be differentiated from the field object name. Text fields as objects are referred to by name, not by variable name. Select the field and press F2 to check the field name. Field name is usually EditField1, EditField2 etc.

Also check "ActionScript object" option (after pressing F2)

TextField.autoSize : *Boolean*

If true, field will automatically extend to include the whole text.

TextField.background : *Boolean*

Field has a uniform background, otherwise it is transparent

TextField.backgroundColor : *Number*

Background Color

TextField.border : *Boolean*

Field has borders

TextField.borderColor : *Number*

Frame Color

TextField.bottomScroll : *Number*

Index of a last visible line of text

TextField..condenseWhite : *Boolean*

If true, in the HTML field all marks of a new line and additional spaces are ignored

TextField.embedFonts : *Boolean*

If true, font from a Flash file is used, if false system font is used

TextField._height : *Number*

Total field height in pixels

TextField.hscroll : *Number*

Position in pixels of a vertically scrolled text

TextField.html : *Boolean*

If true, fields interpret HTML tags

TextField.htmlText : *String*

Field HTML code may include the following tags:

 new line

, <i>, <u> bold, italic, underline end with: , </i>, </u>

 list

 font face, end with:

 font Color, end with:

 font size, end with:

TextField.length : *Number*
Number of characters

TextField.maxChars : *Number*
maximum allowable number of characters in the field, null = no limits

TextField.maxhscroll : *Number*
Maximum possible value of horizontal scroll

TextField.maxscroll : *Number*
Maximum possible value of vertical scroll

TextField.menu : *ContextMenu*
pointer to the field context menu

TextField.mouseWheelEnabled
If true, field supports mouse scroll actions

TextField.multiline : *Boolean*
if true, field can be multiline

TextField._name : *String*
Field object name

TextField._parent : *MovieClip*
Pointer to the sprite containing the field

TextField.password : *Boolean*
Password type field, characters are masked

TextField.restrict : *String*
Set of characters that can be entered in the field.

Examples:

Allow numerals only
`EditField1.restrict = "0123456789";`

Same as above
`EditField1.restrict = "0-9";`

Numerals and uppercase only
`EditField1.restrict = "A-Z 0-9";`

^ character forbids entering the specific character

* cannot be entered
`EditField1.restrict = "^*";`

Numerals cannot be entered
`EditField1.restrict = "^0-9";`

If you want to use ^ or - or \ it must be preceded with \

TextField._rotation : *Number*

Rotates text field by a specific angle

TextField.scroll : *Number*

Vertical field scrolling, index of a first visible line

TextField.selectable : *Boolean*

If true, allows selecting text in the field

TextField.tabEnabled : *Boolean*

If true, field is included in the chain of tab switching

TextField.tabIndex : *Number*

Index of an item in the chain of tab switching

TextField._target : *String*

Absolute object path

TextField.text : *String*

Text in field

TextField.textColor : *Number*

Font Color

TextField.textHeight : *Number*

TextField.textWidth : *Number*

Text size inside the field

TextField.type : *String*

"input" text may be input

"dynamic" text may not be input

TextField._url : *String*

Internet address of a file that created the field

TextField.variable : *String*

Variable name related to the field, usually Edit1 for the EditField1

TextField._visible : *Boolean*

True if the field is visible

TextField._width : *Number*

Total width in pixels

TextField.wordWrap : *Boolean*

if true, line are broken if longer than the field

TextField._x : *Number*

TextField._y : *Number*

Field x and y position

TextField._xmouse : *Number*

TextField._ymouse : *Number*

Cursor location

TextField._xscale : *Number*

TextField._yscale : *Number*

Vertical and horizontal scale in percents

TextField.addListener()

Adds listener to the events of text change within field

TextField.getFontList() : Array

Returns list of fonts available in the system as an array

Method must be called for a global TextField class, not for a single field

```
a = TextField.getFontList();  
Edit1 = a.join();
```

TextField.getDepth()

Return object depth

TextField.removeListener() : Boolean

Removes listener

TextField.removeTextField()

Removes field created with MovieClip.createTextField()

TextField.replaceSel(text:String)

Changes the text selected in the field to a new text

TextField.replaceText(beginIndex:Number, endIndex:Number, text:String)

Changes the text in a field from beginIndex to endIndex with a new text

Function available in a Flash Player 8 plugin or higher

Text field event support

TextField.onChanged : Function

Function called when the field is modified

Example: draw text fields Edit1 and Edit2, paste the frame code:

```
EditField1.onChanged = function ()  
{  
    Edit2 = Edit1;  
}
```

text entered in the Edit1 field will be copied to the Edit2 field

TextField.onKillFocus : Function

Function called when field cannot accept input characters

TextField.onScroller : Function

Function called when field is scrolled

TextField.onSetFocus : Function

Function called when field accepts input characters

Example: draw text fields Edit1 and Edit2, paste the frame code:

```

TextField1.onChanged = function()
{
  Edit2 = "modified text";
}

TextField1.onKillFocus = function()
{
  Edit2 = "finished entering";
}

TextField1.onSetFocus = function()
{
  Edit2 = "start typing";
}

```

Text formatting

TextField.getNewTextFormat()

Creates and returns text formatting object, which will be applied to the new text

TextField.getTextFormat()

Returns default text formatting object

TextField.getTextFormat(index:Number)

Returns text formatting objects from the index character

TextField.getTextFormat(start:Number, end:Number)

Returns text formatting object from the start character to the end character

TextField.setNewTextFormat(tf:TextFormat)

Sets default text formatting

TextField.setNewTextFormat(index:Number, tf:TextFormat)

Sets text formatting from the index character

TextField.setNewTextFormat(start:Number, end:Number, tf:TextFormat)

Sets text formatting from the start character to the end character

TextFormat class

TextFormat.align : *String*

Text adjustment to the left, to the right or centered
Values "left", "right" or "center"

TextFormat.blockIndent : *Number*

Paragraph indent in points, applies to all text lines

TextFormat.bold : *Boolean*

Bold text

TextFormat.bullet : *Boolean*

Bullets

TextFormat.Color : *Number*
Font Color

TextFormat.font : *String*
Font face

TextFormat.indent : *Number*
Indent of the first text line

TextFormat.italic : *Boolean*
Text in italic

TextFormat.leading : *Number*
Horizontal distance between text lines

TextFormat.leftMargin : *Number*
Left text margin

TextFormat.rightMargin : *Number*
Right text margin

TextFormat.size : *Number*
Font size in points

TextFormat.tabStops : *Array[Number]*
Array of a tabulator position in pixels

TextFormat.underline : *Boolean*
Underlined text

TextFormat.url : *String*

TextFormat.target : *String*
Internet link and link target e.g. `_self`, `_blank` etc.

Example:

Draw Edit1 field, double-click, check HTML and click OK.

Paste the following frame code:

```
Edit1 = "www.selteco.com - Click";

tf = new TextFormat();
tf.font = "Tahoma";
tf.Color = 0x0000ff;
tf.bold = true;
tf.url = "http://www.selteco.com";

EditField1.setTextFormat(0,15,tf);
```


CSS

TextField.StyleSheet

Class allows text field formatting using CSS code and cascade styles.

Style example:

```
.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; }
```

TextField.StyleSheet.clear()

Removes formatting with styles

TextField.StyleSheet.getStyle(styleName:String) : Object

Returns style object with styleName and attributes e.g. fontWeight = bold
FontSize = 24px, fontFamily = Arial, Helvetica, sans-serif itd

Example

```
css = new TextField.StyleSheet();  
  
css.parseCSS(".header { font-size:24pt; Color:#0000FF; font-family:times;}");  
  
headerObject = css.getStyle(".header");  
  
Edit1 = headerObject.fontSize;
```

TextField.StyleSheet.getStyleNames() : Array

Returns style name array, e.g. "heading", "mainBody"

TextField.StyleSheet.load(url:String)

Downloads styles from internet address

TextField.StyleSheet.parseCSS(cssText:String) : Boolean

Creates style from the string, returns false in the case of an error

TextField.StyleSheet.setStyle(name:String, style:Object)

Adds style to the collection

```
my_styleSheet:TextField.StyleSheet = new TextField.StyleSheet();  
styleObj = new Object();  
styleObj.Color = "#000000";  
styleObj.fontWeight = "bold";  
my_styleSheet.setStyle("emphasized", styleObj);
```

TextField.StyleSheet.transform(style:Object) : TextFormat

Changes styleSheet type object to TextFormat

TextField.StyleSheet.onLoad : Function(success:Boolean)

Function called when style loaded from file, success is true, if the operation is finished successfully.

Example: creating style and adding it to the edit field.

Draw Edit1 field and paste the frame code

```
css = new TextField.StyleSheet();  
css.parseCSS(".header { font-size:24pt; Color:#0000FF; font-family:times;}");  
EditField1.styleSheet = css;  
EditField1.html = true;  
EditField1.multiline = true;  
Edit1 = "<p class=\"header\">The Dog</p><p>The dog is brown</p>";
```

XML

Class allows to load and use XML files

XML file consists of tags:

XML file example

```
<globe name="World">
  <continent code="na">North America</continent>
  <continent code="sa">South America</continent>
  <continent code="eu">Europe</continent>
  <continent code="af">Africa</continent>
  <continent code="as">Asia</continent>
  <continent code="au">Australia</continent>
</globe>
```

File contains main node (globe) and 6 child nodes (continents), each with a code attribute

XML.attributes : *Array*

Object with the attributes of a current node

XML.childNodes : *Array*

Child node array

XML.firstChild : *XMLNode*

Pointer to the first child node

XML.ignoreWhite : *Boolean*

If true, empty nodes are ignored

XML.lastChild : *XMLNode*

Pointer to the last child node

XML.loaded : *Boolean*

Specifies if the file is loaded

XML.nextSibling : *XMLNode*

Pointer to the next node on the same level

XML.nodeName : *String*

Node name in the brackets < >

XML.nodeType : *Number*

Node type, 1 node < >, 3 text node between < > a </ >

XML.nodeValue : *String*

Node value in the case of a text node (nodeType == 3)

XML.parentNode : *XMLNode*

Parent indicator

XML.previousSibling : *XMLNode*

Pointer to the previous node on the same level

XML.status : *Number*

XML file processing state

0 No error

-2 CDATA section without closing

-3 XML declaration without closing

-4 DOCTYPE declaration without closing

-5 Comment without closing

-6 Incorrect element

-7 No memory

-8 Attribute without closing

-9 No proper closing tag

-10 No proper opening tag

XML.xmlDecl : *String*

XML file declaration, if exists

XML.addRequestHeader(headerName:String, headerValue:String)

In case the file is downloaded from the internet, you can add additional headings and call parameters

XML.appendChild(childNode:XMLNode)

Adds child node to the end of the list

XML.cloneNode(deep:Boolean) : *XMLNode*

Clones and returns node with child nodes to the specific depth

XML.createElement(name:String) : *XMLNode*

Creates and returns new tree element with a specific name

XML.createTextNode(text:String) : *XMLNode*

Creates text node

XML.getBytesLoaded() : *Number*

Returns bytes downloaded when loading file

XML.getBytesTotal() : *Number*

Return XML file size

XML.hasChildNodes() : *Boolean*

Returns true, if the current node has child nodes

XML.insertBefore(childNode:XMLNode, beforeNode:XMLNode)

Add note through other node

XML.load(url:String)

Load XML file from the internet

XML.parseXML(source:String)

Process XML data from the string

XML.removeNode()

Removes node

XML.send(send(url:String, [target:String]))

Sends XML data to the file on www

XML.sendAndLoad(url:String, targetXMLObject:XML)

Sends XML file to the www address and downloads the server reply in XML format to the other XML object

XML.toString() : String

Returns XML data in text format

XML.docTypeDecl: String

XML file !DOCTYPE declaration

XML.onData : function ()

Function called when XML file download is completed

XML.onLoad : function (success:Boolean)

Function called during XML file download

Example 1

draw Edit1 field and paste the frame code

```
str = "<root><node/></root>";
```

```
xml = new XML(str);
```

```
rootNode = xml.firstChild;
```

```
Edit1 = rootNode.nodeName;
```

Example: XML tree traversal

draw Edit1 field and paste the following frame code:

```
str = "<globe name=\"World\">Continents<continent code=\"na\">North  
America</continent><continent code=\"sa\">South America</continent><continent  
code=\"eu\">Europe</continent><continent  
code=\"af\">Africa</continent><continent  
code=\"as\">Asia</continent><continent  
code=\"au\">Australia</continent></globe>";  
  
xml = new XML(str);  
  
globeNode = xml.firstChild;  
  
Edit1 = "Status: " + xml.status + " ";  
  
Edit1 = Edit1 + globeNode.nodeName + ", " + globeNode.attributes.name + ":"  
";  
  
continentNode = globeNode.firstChild;  
  
while(continentNode!=null)  
{  
    if(continentNode.nodeType==1)  
    {  
        Edit1 = Edit1 + continentNode.nodeName;  
        Edit1 = Edit1 + " [" + continentNode.attributes.code + "] ";  
  
        continentText = continentNode.firstChild;  
        Edit1 = Edit1 + continentText.nodeValue + ", ";  
    }  
    continentNode = continentNode.nextSibling;  
}
```

Code creates the following result:

```
Status: 0 globe, World: continent [na] North America, continent [sa] South America, continent [eu]  
Europe, continent [af] Africa, continent [as] Asia, continent [au] Australia,
```

LoadVars

Class allows to import parameters from the text file to SWF file Parameters in the text file:
param1=value1¶m2=value2 .. etc.

Loaded variables can be referred to through an object attribute name LoadVars e.g. LoadVars.param1

new LoadVars()
Creates new object

LoadVars.setRequestHeader(headerName:String, headerValue:String)
Adds additional headings to the file call through the internet

LoadVars.decode(params:String)
Loads and processes string into variables as param1=value1¶m2=value2

LoadVars.getBytesLoaded() : *Number*
Bytes loaded by LoadVars.load() or LoadVars.sendAndLoad() function

LoadVars.getBytesTotal() : *Number*
Total file size with variables

LoadVars.load(url:String) : *Boolean*
Loads variable data from the specific address, variable must have a text form:
param1=value1¶m2=value2 .. etc.

LoadVars.send(url:String) : *Boolean*
Sends variables to the specific internet address as a string
url?param1=value1¶m2=value2

LoadVars.sendAndLoad(url:String) : *Boolean*
Sends query to the specific www address and loads the server reply

LoadVars.toString() : *String*
Returns parameters as
param1=value1¶m2=value2

LoadVars.contentType : *String*
MIME data type

LoadVars.loaded : *Boolean*
Returns true, if data download is completed

LoadVars.onData : *function*
Function called when data download is completed

LoadVars.onLoad : *function*
Function called during data download

Example:

```
lv = new LoadVars();
```

```
lv.decode("name=Pamela&age=25");
```

```
Edit1 = lv.name + " is " + lv.age + " old.";
```


Functions not supported by Alligator Flash Designer

trace()

Trace function is not supported by the standard Flash plugin in the browser. Instead of trace(variable) use:

```
Edit1 = variable;
```

break, continue

Causes the loop to finish or start over, but are ignored

case

Use if function set

class

Custom classes cannot be defined

for in

Object attributes cannot be enumerated

? :

Conditional statement is not supported

{ }

Object attribute initiator

Instead of

```
object = { attr1 : "value1", attr2 : "value2" }
```

Use

```
object = new Object();  
object.attr1 = "value1";  
object.attr2 = "value2";
```