Ruby On Flash API Reference

Ruby On Flash: Compiling Ruby to Swf

http://sourceforge.net/projects/rubyonflash

Table of Contents

Overview

Ruby On Flash standard library can be logically divided into 2 sections, namely the Ruby built-in library and the Augmented Actionscript library.

The Ruby built-in library is an implementation of the official Ruby built-in library. However, for practical reasons, this is only a partial implementation, i.e. not all modules/classes/methods are implemented.
In addition, even implemented classes might not offer the full interface (i.e. the instance and class methods). This is mainly due to limitations of the Swf virtual machine as well as the restrictions of the supported language elements. Most notably, String methods that involve the use of regular expressions are not supported, since regular expressions as a whole are not supported.

Augmented Actionscript library are native Actionscript classes that are augmented with Ruby methods. These classes were chosen as they are directly or indirectly incorporated into Ruby On Flash's component framework. In general, these classes are augmented with methods which serve as wrappers for the native methods and properties. This shields the developers from having to explicitly box and unbox primitive values when using native methods.
Additionally, classes newly introduced by Ruby On Flash, but extends native Actionscript classes would fall under this category.

Ruby Built-In Library

Note that since the classes below are Ruby On Flash's implementation of the official Ruby library, the class and method descriptions are also directly adapted from the official Ruby library documentation.
Refer to http://www.ruby-doc.org/core/

Classes implemented:

1. Object
2. Module
3. Class
4. FalseClass
5. TrueClass
6. NilClass
7. Numeric
8. Fixnum
9. Float
10. String
11. Range
12. Array
13. Hash
14. Proc

Modules implemented:

1. Math

Object

Class Methods:

Object.toObject(expr) => an Object

A utility function for boxing an Actionscript primitive to its Ruby equivalent. This is mostly used by the Ruby built in libraries to interface with the native Actionscript functions.

True -> TrueClass
False -> FalseClass
Null -> NilClass
Integer/Float -> Float

Note: Strings are not supported as it's fairly straightforward to do a manual boxing. Actionscript arrays are equivalent to the Ruby ones.

Object.primitiveEqual?(val1,val2) => true or false

Does a primitive compare. I.e. does the Actionscript equivalent:

```
if(val1==val2){
        return new TrueClass();
}else{
        return new FalseClass();
}
```

This is mostly used by the Ruby built in libraries to interface with the native Actionscript functions.

Object.superclass() => nil

Returns nil

Instance Methods:

anObj.==(other) => true or false

Default implementation of == simply does an Actionscript equivalent: anObj.valueOf() == other.valueOf()

anObj.===(other) => true or false

Returns anObj==other

anObj.eql?(other) => true or false

Returns anObj==other

anObj.inspect() => true or false

Returns anObj.to_s()

anObj.to_s() => String instance

Default implementation calls:
Return String.new(self.toString());

anObj.!=(other) => true or false

Default implementation of != simply does an Actionscript equivalent:
anObj.valueOf() != other.valueOf()

anObj.clone() => an Object

Creates a new object with all the properties of the original. Does the Actionscript equivalent of:

```
newObj = new Object();
for(key in this){
        newObj[key] = this[key];
}
return newObj;
```

anObj.equal?(other) => true or false

Returns true if obj and other are the same object, false otherwise.

anObj.nil?() => false

Return false

anObj.to_a() => an array

Returns an array representation of obj. For objects of class Object and others that don‘t explicitly override the method, the return value is an array containing self. However, this latter behavior will soon be obsolete.

anObj.instance_of?(aClass) => true or false

Returns true if obj is an instance of the given class.

anObj.kind_of?(aClass) => true or false

Returns true if class is the class of obj, or if class is one of the superclasses of obj or modules included in obj.

anObj.is_a?(aClass) => true or false

Returns true if class is the class of obj, or if class is one of the superclasses of obj or modules included in obj.

anObj.class() => a Class

Returns the class of obj. This method must always be called with an explicit receiver, as class is also a reserved word in Ruby.

anObj.rand(max=0) => a Float

Converts max to an integer using max1 = max.to_i.abs. If the result is zero, returns a pseudorandom floating point number greater than or equal to 0.0 and less than 1.0. Otherwise, returns a pseudorandom integer greater than or equal to zero and less than max.

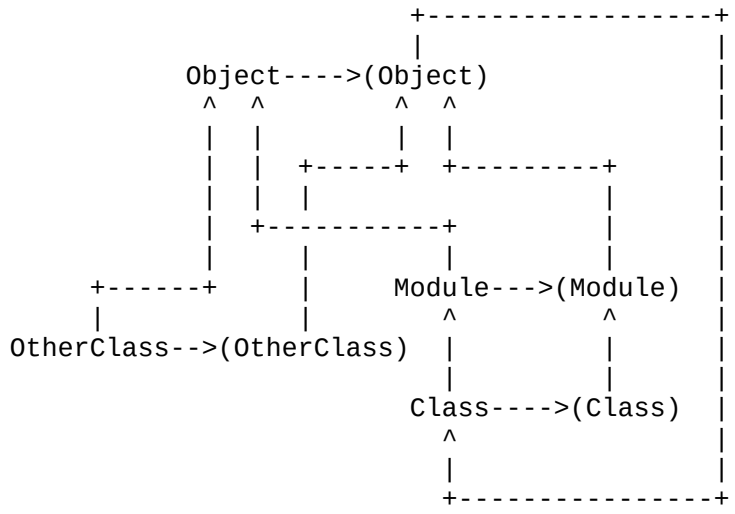
Module < Object

Instance Methods:

mod.include(aMod) => mod
Includes a module.

Class < Module

Classes, modules, and objects are interrelated. In the diagram that follows, the vertical arrows represent inheritance, and the parentheses meta-classes. All metaclasses are instances of the class `Class'.



(Taken from http://www.ruby-doc.org/core/classes/Class.html)

Instance Methods:

class.superclass() => its superclass or nil
Returns the superclass or nil.

class.new(args,…) => obj
Creates a new instance of type class. Invokes initialize with args.

class.subclass_of?(aClass) => true or false
Returns true is this class is a subclass of aClass.
Note: This is not an official Ruby method.

FalseClass < Object

Instance Methods:

false.&(obj) => false

And—Returns false. obj is always evaluated as it is the argument to a method call—there is no short-circuit evaluation in this case.

false.^(obj) =>true or false

Exclusive or. If obj is nil or false, returns false; otherwise, returns true.

false.to_s() =>"false"

Returns "false"

false.|(obj) =>true or false

Or—Returns false if obj is nil or false; true otherwise.

TrueClass < Object

Instance Methods:

true.&(obj) => true or false
And—Returns false if obj is nil or false, true otherwise.

true.^(obj) =>true or false
Exclusive Or—Returns true if obj is nil or false, false otherwise.

true.to_s() =>"true"
Returns "true"

true.|(obj) =>true or false
Or—Returns true. As anObject is an argument to a method call, it is always evaluated; there is no short-circuit evaluation in this case.

NilClass < Object

Instance Methods:

nil.&(obj) => false

And—Returns false. obj is always evaluated as it is the argument to a method call—there is no short-circuit evaluation in this case.

nil.^(obj) =>true or false

Exclusive or. If obj is nil or false, returns false; otherwise, returns true.

nil.to_s() =>" nil"

Returns "nil"

nil.|(obj) =>true or false

Or—Returns false if obj is nil or false; true otherwise.

nil.nil?(obj) =>true

Returns true.

nil.to_s() =>""

Returns "".

nil.to_i() =>0

Returns 0.

nil.to_a() =>[]

Returns an empty array

Numeric < Object

Its only subclasses are Fixnum(Integer) and Float. In situations where it's more convenient, "inherited" methods are implemented at the subclass level, instead of at the "Numeric" level. E.g. the abs method can be more easily implemented at the subclass level, as we do not have to determine the return type, i.e. a Fixnum would return a Fixnum and a Float would return a Float. Semantically, there would be no difference.

Instance Methods:

num.nonzero?() => num or nil
Returns num if num is not zero, nil otherwise.

num.zero?() => true or false
Returns true if num has a zero value.

num.coerce(aNumeric) => array
If aNumeric is the same type as num, returns an array containing aNumeric and num. Otherwise, returns an array with both aNumeric and num represented as Float objects.

num.divmod(aNumeric) => array
Returns an array containing the quotient and modulus obtained by dividing num by aNumeric.

num.modulo(aNumeric) => result
Equivalent to num.divmod(aNumeric)[1].

num.div(aNumeric) => Fixnum
Uses / to perform division, then converts the result to an integer. Numeric does not define the / operator; this is left to subclasses.

num.remainder(aNumeric) => Fixnum
If num and numeric have different signs, returns mod-numeric; otherwise, returns mod. In both cases mod is the value num.modulo(numeric).

Fixnum < Numeric

Since there is no Bignum, Fixnum is synonymous with Integer. In addition, I believe that most programmers would find Integer more intuitive.

Instance Methods:

+fix => fix
Unary plus.

-fix => fixnum
Unary minus.

fix + aNumeric => aNumeric
Addition – Returns a fixnum if aNumeric is a fixnum, Float if aNumeric is a Float.

fix - aNumeric => aNumeric
Minus – Returns a fixnum if aNumeric is a fixnum, Float if aNumeric is a Float.

fix * aNumeric => aNumeric
Multiplication – Returns a fixnum if aNumeric is a fixnum, Float if aNumeric is a Float.

fix / aNumeric => aNumeric
fix.div(aNumeric) => Numeric
Division – Returns a fixnum if aNumeric is a fixnum (value is the result of an integer divide), Float if aNumeric is a Float.

fix % aNumeric => fixnum
Modulo – Returns a fix.

fix ** aNumeric => aNumeric
Power – Returns a fix if aNumeric is a fixnum, Float if aNumeric is a Float.

fix < aNumeric => true or false
Returns true if fix < aNumeric.

fix <= aNumeric => true or false
Returns true if fix <= aNumeric.

fix > aNumeric => true or false
Returns true if fix > aNumeric.

fix >= aNumeric => true or false
Returns true if fix >= aNumeric.

fix == aNumeric => true or false
Returns true if fix and aNumeric have the same value.

fix <=> aNumeric => fixnum
Returns 0 if fix==aNumeric, 1 if fix > aNumeric, -1 otherwise.

fix & other => fixnum
Bitwise And.

fix ^ other => fixnum
Bitwise Exclusive Or.

fix |other => fixnum
Bitwise Or.

fix << count => fixnum
Shifts fix left count positions (right if count is negative).

fix >> count => fixnum
Shifts fix right count positions (left if count is negative).

fix.abs() => fixnum
Returns the absolute value.

fix.integer?() => true
Returns true.

fix.to_i() => fix
fix.to_int() => fix
fix.ceil() => fix
fix.floor() => fix
fix.truncate() => fix
fix.round() => fix
Always return self.

fix.to_f() => aFloat
Returns a Float instance of the same value.

fix.denominator() => 1
Always return 1.

fix.numerator() => fix
Always return self.

fix.chr() => String

Returns a string containing the ASCII character represented by the receiver‘s value.

fix.downto(limit) {|i| block } => int

Iterates block, passing decreasing values from fix down to and including limit.

fix.next => integer
fix.succ => integer

Returns the Integer equal to fix + 1.

fix.upto(limit) {|i| block } => fix

Iterates block, passing in integer values from fix up to and including limit.

fix.times {|i| block } => fix

Iterates block fix times, passing in values from zero to fix - 1.

fix.step(limit, step) {|i| block } => fix

Invokes block with the sequence of numbers starting at num, incremented by step on each call. The loop finishes when the value to be passed to the block is greater than limit (if step is positive) or less than limit (if step is negative).The loop starts at num, uses either the < or > operator to compare the counter against limit, and increments itself using the + operator.

fix[n] => 0, 1

Bit Reference—Returns the nth bit in the binary representation of fix, where fix[0] is the least significant bit.

Float < Numeric

Instance Methods:

+float => float
Unary plus.

- float => -aFloat
Unary minus.

float + aNumeric => aFloat
Addition – Returns a Float.

float - aNumeric => aFloat
Minus – Returns a Float.

float * aNumeric => aFloat
Multiplication – Returns a Float.

float / aNumeric => aFloat
float.div(aNumeric) => aFloat
Division – Returns a Float.

float % aNumeric => aFloat
Modulo – Returns a fix.

float ** aNumeric => aFloat
Power – Returns a Float.

float < aNumeric => true or false
Returns true if float < aNumeric.

float <= aNumeric => true or false
Returns true if float <= aNumeric.

float > aNumeric => true or false
Returns true if float > aNumeric.

float >= aNumeric => true or false
Returns true if float >= aNumeric.

float == aNumeric => true or false
Returns true if float and aNumeric have the same value.

float <=> aNumeric => fixnum

Returns 0 if float ==aNumeric, 1 if float > aNumeric, -1 otherwise.

float.abs() => float.
Returns the absolute value.

float.integer?() => false.
Returns false.

float.ceil() => fixnum
Round up to the smallest integer >= float.

float.floor() => fixnum
float.truncate() => fixnum
float.to_i() => fixnum
float.to_int() => fixnum
Round down to the greatest integer <= float.

float.round() => fixnum
Round to the nearest integer.

float.finite?() => true or false
Returns true if flt is a valid IEEE floating point number (it is not infinite, and nan? is false).

float.infinite?() => true or false
Returns nil, -1, or +1 depending on whether flt is finite, -infinity, or +infinity.

float.nan?() => true or false
Returns true if flt is an invalid IEEE floating point number.

float.to_f() => float
Returns self.

String < Object

Represents a string object. Since the current implementation of Ruby On Flash does not include regular expressions, methods which use regex, such as the % operator, are not supported.

Instance methods:

str * integer => new_str

Copy—Returns a new String containing integer copies of the receiver.

str + other_str => new_str

Concatenation—Returns a new String containing other_str concatenated to str.

str << fixnum => str
str.concat(fixnum) => str
str << obj => str
str.concat(obj) => str

Append—Concatenates the given object to str. If the object is a Fixnum between 0 and 255, it is converted to a character before concatenation.

str <=> other_str => -1, 0, +1

Comparison—Returns -1 if other_str is less than, 0 if other_str is equal to, and +1 if other_str is greater than str. If the strings are of different lengths, and the strings are equal when compared up to the shortest length, then the longer string is considered greater than the shorter one.

str == obj => true or false

Equality—If obj is not a String, returns false. Otherwise, returns true if str <=> obj returns zero.

str[fixnum] => fixnum or nil
str[fixnum, fixnum] => new_str or nil
str[range] => new_str or nil
str[other_str] => new_str or nil
str.slice(fixnum) => fixnum or nil
str.slice(fixnum, fixnum) => new_str or nil
str.slice(range) => new_str or nil
str.slice(other_str) => new_str or nil

Element Reference—If passed a single Fixnum, returns the code of the character at that position. If passed two Fixnum objects, returns a substring starting at the offset given by the first, and a length given by the second. If given a range, a substring containing characters at offsets given by the range is returned. In all three cases, if an offset is negative, it is counted from the end of str. Returns nil if

the initial offset falls outside the string, the length is negative, or the beginning of the range is greater than the end.
NOTE: Regex is not supported.

str.capitalize() => new_str

Returns a copy of str with the first character converted to uppercase and the remainder to lowercase.

str.capitalize!() => str or nil

Modifies str by converting the first character to uppercase and the remainder to lowercase. Returns nil if no changes are made.

str.casecmp(other_str) => -1, 0, +1

Case-insensitive version of String#<=>.

str.center(integer, padstr) => new_str

If integer is greater than the length of str, returns a new String of length integer with str centered and padded with padstr; otherwise, returns str.

str.chomp(separator=$/) => new_str

Returns a new String with the given record separator removed from the end of str (if present). If $/ has not been changed from the default Ruby record separator, then chomp also removes carriage return characters (that is it will remove \n, \r, and \r\n).

str.chomp!(separator=$/) => str or nil

Modifies str in place as described for String#chomp, returning str, or nil if no modifications were made.

str.chop() => new_str

Returns a new String with the last character removed. If the string ends with \r\n, both characters are removed.

str.chop!() => str or nil

Processes str as for String#chop, returning str, or nil if str is the empty string.

str.count(other_str) => fixnum

Counts the number of occurrences of other_str.
NOTE: differs from the official Ruby implementation, which takes in a set of arguments, the intersection of which defines the characters to count in str.

str.delete(other_str) => new_str

Finds all occurrences of other_str and deletes them.
NOTE: differs from the official Ruby implementation, which takes in a set of arguments, the intersection of which defines the characters to delete in str.

str.downcase() => new_str

Returns a copy of str with all uppercase letters replaced with their lowercase counterparts.

str.downcase!() => str or nil

Downcases the contents of str, returning nil if no changes were made.

str.each(separator=$/) {|substr| block } => str
str.each_line(separator=$/) {|substr| block } => str

Splits str using the supplied parameter as the record separator ($/ by default), passing each substring in turn to the supplied block. If a zero-length record separator is supplied, the string is split on \n characters, except that multiple successive newlines are appended together.

str.each_char() {|x| ...}

Passes each char in str to the given block.

str.empty?() => true or false

Returns true if str has a length of zero.

str.eql?(other) => true or false

Two strings are equal if the have the same length and content.

str.include?(other_str) => true or false
str.include?(fixnum) => true or false

Returns true if str contains the given string or character.

str.index(substring [, offset]) => fixnum or nil
str.index(fixnum [, offset]) => fixnum or nil

Returns the index of the first occurrence of the given substring or character (fixnum), in str. Returns nil if not found. If the second parameter is present, it specifies the position in the string to begin the search.
NOTE: does not support regex

str.replace(other_str) => str

Replaces the contents of str with the corresponding values in other_str.

str.insert(index, other_str) => str

Inserts other_str before the character at the given index, modifying str. Negative indices count from the end of the string, and insert after the given character. The intent is insert aString so that it starts at the given index.

str.length() => integer

Returns the length of str.

str.ljust(integer, padstr=' ') => new_str

If integer is greater than the length of str, returns a new String of length integer with str left justified and padded with padstr; otherwise, returns str.

str.lstrip() => new_str

Returns a copy of str with leading whitespace removed.

str.lstrip!() => self or nil

Removes leading whitespace from str, returning nil if no change was made.

str.succ() => new_str
str.next() => new_str

Returns the successor to str. The successor is calculated by incrementing characters starting from the rightmost alphanumeric (or the rightmost character if there are no alphanumerics) in the string. Incrementing a digit always results in another digit, and incrementing a letter results in another letter of the same case. Incrementing nonalphanumerics uses the underlying character set‘s collating sequence.

If the increment generates a ``carry,'' the character to the left of it is incremented. This process repeats until there is no carry, adding an additional character if necessary.

"abcd".succ #=> "abce"
"THX1138".succ #=> "THX1139"
"<<koala>>".succ #=> "<<koalb>>"
"1999zzz".succ #=> "2000aaa"
"ZZZ9999".succ #=> "AAAA0000"
"***".succ #=> "**+"

str.succ!() => str
str.next!() => str

Equivalent to String#succ, but modifies the receiver in place.

str.reverse() => new_str

Returns a new string with the characters from str in reverse order.

str.reverse!() => str

Reverses str in place.

str.rindex(substring [, fixnum]) => fixnum or nil
str.rindex(fixnum [, fixnum]) => fixnum or nil

Returns the index of the last occurrence of the given substring or character (fixnum) in str. Returns nil if not found. If the second parameter is present, it specifies the position in the string to end the search—characters beyond this point will not be considered.

NOTE: regex is not supported.

str.rjust(integer, padstr=' ') => new_str

If integer is greater than the length of str, returns a new String of length integer with str right justified and padded with padstr; otherwise, returns str.

str.rstrip() => new_str

Returns a copy of str with trailing whitespace removed.

str.rstrip!() => self or nil

Removes trailing whitespace from str, returning nil if no change was made.

str.split(pattern=$;, [limit]) => anArray

Divides str into substrings based on a delimiter, returning an array of these substrings.

If pattern is a String, then its contents are used as the delimiter when splitting str. If pattern is a single space, str is split on whitespace, with leading whitespace and runs of contiguous whitespace characters ignored.

If the limit parameter is omitted, trailing null fields are suppressed. If limit is a positive number, at most that number of fields will be returned (if limit is 1, the entire string is returned as the only entry in an array). If negative, there is no limit to the number of fields returned, and trailing null fields are not suppressed.

NOTE: regex is not supported

str.squeeze() => new_str

All runs of identical characters are replaced by a single character.

NOTE: differs from the official Ruby implementation, where a set of arguments is accepted.

str.strip() => new_str

Returns a copy of str with leading and trailing whitespace removed.

str.strip!() => str or nil

Removes leading and trailing whitespace from str. Returns nil if str was not altered.

str.swapcase() => new_str

Returns a copy of str with uppercase alphabetic characters converted to lowercase and lowercase characters converted to uppercase.

str.swapcase!() => str or nil

Equivalent to String#swapcase, but modifies the receiver in place, returning str, or nil if no changes were made.

str.to_f() => float

Returns the result of interpreting leading characters in str as a floating point number. Extraneous characters past the end of a valid number are ignored. If there is not a valid number at the start of str, 0.0 is returned.

str.to_i() => integer

Returns the result of interpreting leading characters in str as an integer. Extraneous characters past the end of a valid number are ignored. If there is not a valid number at the start of str, 0 is returned.
NOTE: differs from the official Ruby implementation where one can choose an integer base (2, 8, 10, 16).

str.upcase() => new_str

Returns a copy of str with all lowercase letters replaced with their uppercase counterparts.

str.upcase!() => str or nil

Upcases the contents of str, returning nil if no changes were made.

str.upto(other_str) {|s| block } => str

Iterates through successive values, starting at str and ending at other_str inclusive, passing each value in turn to the block. The String#succ method is used to generate each value.

Range < Object

Class methods:

Range.new(start, end, exclusive=false) => range
Constructs a range using the given start and end. If the third parameter is omitted or is false, the range will include the end object; otherwise, it will be excluded.

Instance methods:

rng == obj => true or false
Returns true only if obj is a Range, has equivalent beginning and end items (by comparing them with ==), and has the same exclude_end? setting as rng.

rng === obj => true or false
rng.member?(val) => true or false
rng.include?(val) => true or false
Returns true if obj is an element of rng, false otherwise. Conveniently, === is the comparison operator used by case statements.

rng.first() => obj
rng.begin() => obj
Returns the first object in rng.

rng.each {| i | block } => rng
Iterates over the elements rng, passing each in turn to the block. You can only iterate if the start object of the range supports the succ method (which means that you can‘t iterate over ranges of Float objects).

rng.end() => obj
rng.last() => obj
Returns the object that defines the end of rng.

rng.eql?(obj) => true or false
Returns true only if obj is a Range, has equivalent beginning and end items (by comparing them with eql?), and has the same exclude_end? setting as rng.

rng.exclude_end?() => true or false
Returns true if rng excludes its end value.

rng.step(n=1) {| obj | block } => rng
Iterates over rng, passing each nth element to the block. If the range contains numbers or strings, natural ordering is used. Otherwise step invokes succ to iterate through range elements. The following code uses class Xs, which is defined in the class-level documentation.

Array < Object

Represents an array object. Note that the native Actionscript Array and this Ruby Array are the same object. As a result, the length() method is not available, as native Actionscript Array objects use the length attribute to maintain its state. Use size() instead.

Class Methods:

Array[…] => array

Returns a new array populated with the given objects.

Array.new(size=0, obj=nil)
Array.new(array)
Array.new(size) {|index| block }

Returns a new array. In the first form, the new array is empty. In the second it is created with size copies of obj (that is, size references to the same obj). The third form creates a copy of the array passed as a parameter (the array is generated by calling to_ary on the parameter). In the last form, an array of the given size is created. Each element in this array is calculated by passing the element‘s index to the given block and storing the return value.

Instance methods:

array & other_array

Set Intersection—Returns a new array containing elements common to the two arrays, with no duplicates.

array * int => an_array
array * str => a_string

Repetition—With a String argument, equivalent to self.join(str). Otherwise, returns a new array built by concatenating the int copies of self.

array + other_array => an_array

Concatenation—Returns a new array built by concatenating the two arrays together to produce a third array.

array - other_array => an_array

Array Difference—Returns a new array that is a copy of the original array, removing any items that also appear in other_array. (If you need set-like behavior, see the library class Set.)

array << obj => array

Append—Pushes the given object on to the end of this array. This expression returns the array itself, so several appends may be chained together.

array <=> other_array => -1, 0, +1

Comparison—Returns an integer (-1, 0, or +1) if this array is less than, equal to, or greater than other_array. Each object in each array is compared (using <=>). If any value isn‘t equal, then that inequality is the return value. If all the values found are equal, then the return is based on a comparison of the array lengths. Thus, two arrays are ``equal'' according to Array#<=> if and only if they have the same length and the value of each element is equal to the value of the corresponding element in the other array.

array == other_array => bool

Equality—Two arrays are equal if they contain the same number of elements and if each element is equal to (according to Object.==) the corresponding element in the other array.

array[index] => obj or nil
array[start, length] => an_array or nil
array[range] => an_array or nil
array.slice(index) => obj or nil
array.slice(start, length) => an_array or nil
array.slice(range) => an_array or nil

Element Reference—Returns the element at index, or returns a subarray starting at start and continuing for length elements, or returns a subarray specified by range. Negative indices count backward from the end of the array (-1 is the last element). Returns nil if the index (or starting index) are out of range.

array[index] = obj => obj
array[start, length] = obj or an_array or nil => obj or an_array or nil
array[range] = obj or an_array or nil => obj or an_array or nil

Element Assignment—Sets the element at index, or replaces a subarray starting at start and continuing for length elements, or replaces a subarray specified by range. If indices are greater than the current capacity of the array, the array grows automatically. A negative indices will count backward from the end of the array. Inserts elements if length is zero. If nil is used in the second and third form, deletes elements from self. An IndexError is raised if a negative index points past the beginning of the array.

array.assoc(obj) => an_array or nil

Searches through an array whose elements are also arrays comparing obj with the first element of each contained array using obj.==. Returns the first contained array that matches (that is, the first associated array), or nil if no match is found.

array.at(index) => obj or nil

Returns the element at index. A negative index counts from the end of self. Returns nil if the index is out of range.

array.clear() => array

Removes all elements from self.

array.collect() {|item| block } => an_array
array.map() {|item| block } => an_array

Invokes block once for each element of self. Creates a new array containing the values returned by the block.

array.compact() => an_array

Returns a copy of self with all nil elements removed.

array.compact!() => array or nil

Removes nil elements from array. Returns nil if no changes were made.

array.concat(other_array) => array

Appends the elements in other_array to self.

array.delete(obj) => obj or nil
array.delete(obj) { block } => obj or nil

Deletes items from self that are equal to obj. If the item is not found, returns nil. If the optional code block is given, returns the result of block if the item is not found.

array.delete_at(index) => obj or nil

Deletes the element at the specified index, returning that element, or nil if the index is out of range.

array.delete_if() {|item| block } => array

Deletes every element of self for which block evaluates to true.

array.each() {|item| block } => array

Calls block once for each element in self, passing that element as a parameter.

array.each_index() {|index| block } => array

Same as Array#each, but passes the index of the element instead of the element itself.

array.empty?() => true or false

Returns true if self array contains no elements.

array.fetch(index) => obj
array.fetch(index, default) => obj
array.fetch(index) {|index| block } => obj

Tries to return the element at position index. If the index lies outside the array, the first form returns nil, the second form returns default, and the third form returns the value of invoking the block, passing in the index. Negative values of index count from the end of the array.

NOTE: differs from official Ruby implementation in that in the first form, this returns nil instead of throwing an exception.

array.fill(obj) => array
array.fill(obj, start [, length]) => array
array.fill(obj, range) => array
array.fill {|index| block } => array
array.fill(start [, length]) {|index| block } => array
array.fill(range) {|index| block } => array

The first three forms set the selected elements of self (which may be the entire array) to obj. A start of nil is equivalent to zero. A length of nil is equivalent to self.length. The last three forms fill the array with the value of the block. The block is passed the absolute index of each element to be filled.

array.first() => obj or nil
array.first(n) => an_array

Returns the first element, or the first n elements, of the array. If the array is empty, the first form returns nil, and the second form returns an empty array.

array.flatten() => an_array

Returns a new array that is a one-dimensional flattening of this array (recursively). That is, for every element that is an array, extract its elements into the new array.

array.flatten!() => array or nil

Flattens self in place. Returns nil if no modifications were made (i.e., array contains no subarrays.)

array.include?(obj) => true or false

Returns true if the given object is present in self (that is, if any object == anObject), false otherwise.

array.index(obj) => int or nil

Returns the index of the first object in self such that is == to obj. Returns nil if no match is found.

array.indexes(i1, i2, ... iN) => an_array
array.indices(i1, i2, ... iN) => an_array

Returns an array of values at positions i1, i2, …

array.replace(other_array) => array

Replaces the contents of self with the contents of other_array, truncating or expanding if necessary.

array.insert(index, obj...) => array

Inserts the given values before the element with the given index (which may be negative).

array.join(sep=$,) => str

Returns a string created by converting each element of the array to a string, separated by sep.

array.last() => obj or nil
array.last(n) => an_array
Returns the last element(s) of self. If the array is empty, the first form returns nil.

array.nitems() => int
Returns the number of non-nil elements in self. May be zero.

array.pop() => obj or undefined
Removes the last element from self and returns it, or undefined if the array is empty.
NOTE: differs from official Ruby implementation in that this returns undefined (Actonscript null) if the array is empty, instead of nil. This is because the native Actionscript components, such as Key, relies on this method, thus we cannot override this method.

array.push(obj, ...) => array
Append—Pushes the given object(s) on to the end of this array. This expression returns the array itself, so several appends may be chained together.
NOTE: The native Actionscript components rely on this method, so override this method at your own risk! Refer to the technical specifications for more details.

array.rassoc(key) => an_array or nil
Searches through the array whose elements are also arrays. Compares key with the second element of each contained array using ==. Returns the first contained array that matches.

array.reject() {|item| block } => an_array
Returns a new array containing the items in self for which the block is not true.

array.reject!() {|item| block } => array or nil
Equivalent to Array#delete_if, deleting elements from self for which the block evaluates to true, but returns nil if no changes were made.

array.reverse() => an_array
Returns a new array containing self‘s elements in reverse order.

array.reverse!() => array
Reverses self in place.

array.reverse_each() {|item| block }
Same as Array#each, but traverses self in reverse order.

array.rindex(obj) => int or nil

Returns the index of the last object in array == to obj. Returns nil if no match is found.

array.select() {|item| block } => an_array

Invokes the block passing in successive elements from array, returning an array containing those elements for which the block returns a true value

array.shift() => obj or undefined

Returns the first element of self and removes it (shifting all other elements down by one). Returns undefined if the array is empty.
NOTE: differs from official Ruby implementation in that this returns undefined (Actonscript null) if the array is empty, instead of nil. This is because the native Actionscript components, such as Key, relies on this method, thus we cannot override this method.

array.size() => int

Returns the size of this array.

array.sort() => an_array
array.sort() {| a,b | block } => an_array

Returns a new array created by sorting self. Comparisons for the sort will be done using the <=> operator or using an optional code block. The block implements a comparison between a and b, returning -1, 0, or +1.

array.sort!() => array
array.sort!() {| a,b | block } => array

Sorts self. Comparisons for the sort will be done using the <=> operator or using an optional code block. The block implements a comparison between a and b, returning -1, 0, or +1.

array.to_a() => array

Returns self. If called on a subclass of Array, converts the receiver to an Array object.

array.to_ary() => array

Returns self.

array.to_s() => string

Returns self.join.

array.uniq() => an_array

Returns a new array by removing duplicate values in self.

array.uniq!() => array or nil

Removes duplicate elements from self. Returns nil if no changes are made (that is, no duplicates are found).

array.unshift(obj, ...) => array

Prepends objects to the front of array. other elements up one.

array | other_array => an_array

Set Union—Returns a new array by joining this array with other_array, removing duplicates.

Hash < Object

Class methods:

Hash[[key, value]*] => hash

Creates a new hash populated with the given objects. Equivalent to the literal { key, value, … }. Keys and values occur in pairs, so there must be an even number of arguments.
NOTE: Does not support Hash[“a”=> 1, “b”=>2,…]

Hash.new() => hash
Hash.new(obj) => aHash
Hash.new() {|hash, key| block } => aHash

Returns a new, empty hash. If this hash is subsequently accessed by a key that doesn‘t correspond to a hash entry, the value returned depends on the style of new used to create the hash. In the first form, the access returns nil. If obj is specified, this single object will be used for all default values. If a block is specified, it will be called with the hash object and the key, and should return the default value. It is the block‘s responsibility to store the value in the hash if required.

Instance methods:

hsh == other_hash => true or false

Equality—Two hashes are equal if they each contain the same number of keys and if each key-value pair is equal to (according to Object#==) the corresponding elements in the other hash.

hsh[key] => value

Element Reference—Retrieves the value object corresponding to the key object. If not found, returns the a default value

hsh[key] = value => value
hsh.store(key, value) => value

Element Assignment—Associates the value given by value with the key given by key.
NOTE: In the official Ruby implementation, a String passed as a key will be duplicated and frozen.

hsh.clear => hsh

Removes all key-value pairs from hsh.

hsh.get_default(key=nil) => obj

Returns the default value, the value that would be returned by hsh[key] if key did not exist in hsh.

NOTE: This method does not exist in the official Ruby implementation. The official Ruby implementation uses a read/write attribute instead.

hsh.set_default(obj) => hsh

Sets the default value, the value returned for a key that does not exist in the hash. NOTE: This method does not exist in the official Ruby implementation. The official Ruby implementation uses a read/write attribute instead.

hsh.default_proc => anObject

If Hash::new was invoked with a block, return that block, otherwise return nil.

hsh.delete(key) => value
hsh.delete(key) {| key | block } => value

Deletes and returns a key-value pair from hsh whose key is equal to key. If the key is not found, returns the default value. If the optional code block is given and the key is not found, pass in the key and return the result of block.

hsh.delete_if {| key, value | block } => hsh

Deletes every key-value pair from hsh for which block evaluates to true.

hsh.each {| key, value | block } => hsh

Calls block once for each key in hsh, passing the key and value to the block as a two-element array. Because of the assignment semantics of block parameters, these elements will be split out if the block has two formal parameters. Also see Hash.each_pair, which will be marginally more efficient for blocks with two parameters.

hsh.each_key {| key | block } => hsh

Calls block once for each key in hsh, passing the key as a parameter.

hsh.each_pair {| key_value_array | block } => hsh

Calls block once for each key in hsh, passing the key and value as parameters.

hsh.each_value {| value | block } => hsh

Calls block once for each key in hsh, passing the value as a parameter.

hsh.empty? => true or false

Returns true if hsh contains no key-value pairs.

hsh.fetch(key [, default]) => obj
hsh.fetch(key) {| key | block } => obj

Returns a value from the hash for the given key. If the key can‘t be found, there are several options: With no other arguments, it will return nil; if default is given, then that will be returned; if the optional code block is specified, then that will be run and its result returned.

NOTE: differs from the official Ruby implementation, as an IndexError is not thrown. Instead, nil is returned

hsh.has_key?(key) => true or false
hsh.include?(key) => true or false
hsh.key?(key) => true or false
hsh.member?(key) => true or false
Returns true if the given key is present in hsh.

hsh.has_value?(value) => true or false
hsh.value?(value) => true or false
Returns true if the given value is present for some key in hsh.

hsh.index(value) => key
Returns the key for a given value. If not found, returns nil.

hsh.replace(other_hash) => hsh
Replaces the contents of hsh with the contents of other_hash.

hsh.invert => aHash
Returns a new hash created by using hsh‘s values as keys, and the keys as values.

hsh.length => fixnum
hsh.size => fixnum
Returns the number of key-value pairs in the hash.

hsh.merge(other_hash) => a_hash
Returns a new hash containing the contents of other_hash and the contents of hsh, overwriting entries in hsh with duplicate keys with those from other_hash.

hsh.merge!(other_hash) => hsh
hsh.update(other_hash) => hsh
Adds the contents of other_hash to hsh, overwriting entries with duplicate keys with those from other_hash.

hsh.reject {| key, value | block } => a_hash
Same as Hash#delete_if, but works on (and returns) a copy of the hsh.

hsh.reject! {| key, value | block } => hsh or nil
Equivalent to Hash#delete_if, but returns nil if no changes were made.

hsh.select {|key, value| block} => array
Returns a new array consisting of [key,value] pairs for which the block returns true.

hsh.shift => anArray or obj

Removes a key-value pair from hsh and returns it as the two-item array [key, value], or the hash‘s default value if the hash is empty.

hsh.sort => array
hsh.sort {| a, b | block } => array
Converts hsh to a nested array of [key, value] arrays and sorts it, using Array#sort.

hsh.to_a => array
Converts hsh to a nested array of [key, value] arrays.

hsh.to_hash => hsh
Returns self.

hsh.to_s => string
Converts hsh to a string by converting the hash to an array of [key, value] pairs and then converting that array to a string using Array#join with the default separator.

hsh.values => array
Returns a new array populated with the values from hsh.

hsh.values_at(key, ...) => array
Return an array containing the values associated with the given keys.

Proc < Object

Class methods:

Proc.new() {|...| block } => a_proc
Converts the block into a Proc object and return it.

Instance methods:

prc == other_proc => true or false
Return true if prc is the same object as other_proc.
NOTE: according to the official Ruby documentation (http://www.ruby-doc.org/core/), this method would return true if both have the same body. However, there is no way these two can be compared during runtime in Ruby On Flash.

prc.call(params,...) => obj
prc[params,...] => obj
Invokes the block, setting the block‘s parameters to the values in params using something close to method calling semantics. Generates a warning if multiple values are passed to a proc that expects just one (previously this silently converted the parameters to an array).

prc.arity() => fixnum
Returns the number of arguments that would not be ignored. If the block is declared to take no arguments, returns 0. If the block is known to take exactly n arguments, returns n. If the block has optional arguments, return -n-1, where n is the number of mandatory arguments. A proc with no argument declarations is the same a block declaring || as its arguments.

prc.to_proc() => prc
Part of the protocol for converting objects to Proc objects. Instances of class Proc simply return themselves.

prc.to_s() => “proc instance”
Returns “proc instance”.
NOTE: The official Ruby implementation returns the unique id.

Module Math

Constants:

Math.PI
Math.E

Class methods:

Math.acos(x) => float
Computes the arc cosine of x. Returns 0..PI.

Math.acosh(x) => float
Computes the inverse hyperbolic cosine of x.

Math.asin(x) => float
Computes the arc sine of x. Returns 0..PI.

Math.asinh(x) => float
Computes the inverse hyperbolic sine of x.

Math.atan(x) => float
Computes the arc tangent of x. Returns -{PI/2} .. {PI/2}.

Math.atan2(y, x) => float
Computes the arc tangent given y and x. Returns -PI..PI.

Math.atanh(x) => float
Computes the inverse hyperbolic tangent of x.

Math.cos(x) => float
Computes the cosine of x (expressed in radians). Returns -1..1.

Math.cosh(x) => float
Computes the hyperbolic cosine of x (expressed in radians).

Math.exp(x) => float
Returns e**x.

Math.frexp(numeric) => [fraction, exponent]
Returns a two-element array containing the normalized fraction (a Float) and exponent (a Fixnum) of numeric.

Math.hypot(x, y) => float

Returns sqrt(x**2 + y**2), the hypotenuse of a right-angled triangle with sides x and y.

Math.ldexp(flt, int) => float
Returns the value of flt*(2**int).

Math.log(numeric) => float
Returns the natural logarithm of numeric.

Math.log10(numeric) => float
Returns the base 10 logarithm of numeric.

Math.sin(x) => float
Computes the sine of x (expressed in radians). Returns -1..1.

Math.sinh(x) => float
Computes the hyperbolic sine of x (expressed in radians).

Math.sqrt(numeric) => float
Returns the non-negative square root of numeric.

Math.tan(x) => float
Returns the tangent of x (expressed in radians).

Math.tanh() => float
Computes the hyperbolic tangent of x (expressed in radians).

Augmented Actionscript Library

Note that since most methods are already native to the Actionscript classes or are wrappers for existing methods, most of the method descriptions are directly adapted from the official Actionscript documentation.
Refer to http://livedocs.adobe.com/flash/mx2004

Native Classes Augmented:

1. Key
2. Mouse
3. Sound
4. MovieClip
5. TextField

New Classes Introduced:

1. Sprite
2. ImageSprite
3. SwfSprite

Key < Object

Key is a native ActionScript class.
This class is augmented with wrappers for its original methods in order to box the original methods' primitive arguments and return values.

Constants:

Key.BACKSPACE = 27
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.PGDN = 34
Key.PGUP = 33
Key.RIGHT = 39
Key.SHIFT = 16
Key.SPACE = 32
Key.TAB = 9
Key.UP = 38

Class methods:

Key.addListener(listenerObj)
listenerObj: An object with methods onKeyDown and onKeyUp.
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

Key.getAscii() => fixnum
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.

Key.getCode() => fixnum
Returns the key code value of the last key pressed.

Key.isDown(keycode) => true or false

keycode The key code value assigned to a specific key or a Key class property associated with a specific key.

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.

Key.isToggled(keycode) => true or false

keycode The key code for the Caps Lock key (20) or the Num Lock key (144).

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.

Key.removeListener(listener) => true or false

Removes an object previously registered with Key.addListener().

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.

Mouse < Object

Mouse is a native Actionscript class.
This class is augmented with wrappers for its original methods in order to box the original methods' primitive arguments and return values.

Class methods:

Mouse.addListener(newListener)

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.

Mouse.hide() => true or false

Hides the pointer in a SWF file. The pointer is visible by default.
Returns true if the pointer is visible; false otherwise.

Mouse.removeListener(listener) => true or false

Removes an object that was previously registered with addListener().
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.

Mouse.show() => fixnum

Displays the mouse pointer in a SWF file. The pointer is visible by default.
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.

Sound < Object

Sound is a native Actionscript class.
This class is augmented with wrappers for its original methods in order to box the original methods' primitive arguments and return values.

Event handlers:

snd.onID3()

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.

snd.onLoad(success)

success A Boolean value of true if my_sound has been loaded successfully, false otherwise.
Invoked automatically when a sound loads. You must create a function that executes when the this handler is invoked.
You should define this handler before you call mySound.loadSound().

snd.onSoundComplete()

Invoked automatically when a sound finishes playing. You can use this handler to trigger events in a SWF file when a sound finishes playing.

Instance methods:

snd.attachSound(idName)

idName The identifier of an exported sound in the library.
Attaches the sound specified in the idName parameter to the specified Sound object.
NOTE: The current implementation of Ruby On Flash does not support embedding of sound, thus effectively rendering this method useless. However, we expect to support embedding of sound files soon!

snd.getDuration() => fixnum

Returns the duration of a sound, in milliseconds.

snd.getBytesLoaded() => fixnum

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.

snd.getBytesTotal() => fixnum

Returns the size, in bytes, of the specified Sound object.

snd.getPan() => fixnum

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().

snd.getTransform() => an object

Returns the object with properties that contain the channel percentage values for the specified sound object set with the last Sound.setTransform() call.

snd.getVolume() => fixnum

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.

snd.getId3() => an object

Returns an object that 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.
NOTE: For more information on the tags supported, please visit Adobe's website: http://livedocs.adobe.com/flash/mx2004/main_7_2/wwhelp/wwhimpl/js/html/wwhelp.htm?href=Part_ASLR.html

snd.loadSound(url, isStreaming)

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).
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.
All MP3s (event or streaming) loaded with this method are saved in the browser's file cache on the user's system.

snd.getPosition() => fixnum

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.

snd.setPan(pan) => fixnum

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.

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.

Sound.setTransform(soundTransformObject)

soundTransformObject An object created with the constructor for the generic Object class.
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:
11: A percentage value specifying how much of the left input to play in the left speaker (0-100).
1r: 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).
Refer to
http://livedocs.adobe.com/flash/mx2004/main_7_2/00001686.html#wp4004984

snd.setVolume(volume)

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.
Sets the volume for the Sound object.

snd.start(secondOffset, loop)

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.
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.

snd.stop(idName)

idName An optional parameter specifying a specific sound to stop playing.

stops all sounds currently playing if no parameter is specified, or just the sound specified in the idName parameter.

MovieClip < Object

MovieClip is a native Actionscript class.
This class is augmented with wrappers for its original methods in order to box the original methods' primitive arguments and return values.

mc.getAlpha() => float
mc.setAlpha(alpha) => float

Gets or sets the alpha transparency value of the movie clip specified by 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.

mc.attachAudio(source)

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).
Specifies the audio source to be played. To stop playing the audio source, pass false for source.

mc.isEnabled?() => true or false
mc.setEnabled?(enabled)

Gets or sets the enabled property of mc.
NOTE: for more information, pleas refer to the Adobe website:
http://livedocs.adobe.com/flash/mx2004/main_7_2/wwhelp/wwhimpl/js/html/wwhelp.htm?href=Part_ASLR.html

mc.isFocusEnabled?() => true or false
mc.setFocusEnabled?(enabled)

A boolean value which indicates whether mc can receive input focus.
If the enabled is false, a movie clip cannot receive input focus unless it is a button.
If the enabled is true, a movie clip can receive input focus even if it is not a button.

mc.isFocusRect?() => true or false
mc.setFocusRect?(enabled)

A Boolean value that specifies whether a movie clip has a yellow rectangle around it when it has keyboard focus.
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.

mc.getBounds(targetCoordinateSpace)=> Object

Returns properties that are the minimum and maximum x and y coordinate values of the instance specified by my_mc for the targetCoordinateSpace parameter.

mc.getHeight() => a Float
mc.setHeight(height)

Gets or sets the height of the movie clip to height pixels.

mc.getHitArea() => object
mc.setHitArea(obj)

Retrieves or designates another movie clip to serve as the hit area for a movie clip.

mc.isLockRoot?() => true or false
mc.setLockRoot?(flag)

Gets or sets what the _root property refers to.
http://livedocs.adobe.com/flash/mx2004/main_7_2/00001522.html#3999468

mc.getMenu() => object
mc.setMenu(menu)

Gets or associates the specified ContextMenu object with the movie clip 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.

mc.getName() => a String
mc.setName(name)

Gets or sets the instance name of the movie clip specified by mc.

mc.getParent() => an Object
mc.setParent(parent)

Gets or sets the reference to the movie clip or object that contains the current movie clip or object.

mc.getQuality() => a String
mc.setQuality(aString)

Gets or sets the rendering quality used for a SWF file. Refer to above for the valid values.
Although you can specify this property for a Movie Clip object, it is actually a global 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.

mc.getRotation() => float
mc.setRotation(degrees)

Gets or sets 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.

mc.getSoundBufTime() => an Integer
mc.setSoundBufTime(time)

Gets or sets an integer that specifies the number of seconds a sound prebuffers before it starts to stream.
Although you can specify this property for a MovieClip object, it is actually a global property

mc.isTabChildren?() => true or false
mc.setTabChildren?(tabChildren)

Indicates whether the children of this movie clip is included in the tab ordering. 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.

mc.isTabEnabled?() => nil, true or false
mc.setTabEnabled?(flag)

Indicates whether my_mc is included in automatic tab ordering.
If tabEnabled is nil, 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.

mc.getTabIndex() => an Integer
mc.setTabIndex(index)

Gets or sets the tab index of mc.

mc.getTarget() => a String

Returns the target path of the movie clip instance specified by mc in slash notation.

mc.isTrackAsMenu?() => true or false
mc.setTrackAsMenu(flag)

Indicates whether or not other buttons or movie clips can receive mouse release events.
Sets whether or not other buttons or movie clips can receive mouse release events.

mc.getMcUrl() => a String

Retrieves the URL of the SWF or JPEG file from which the movie clip was downloaded.

mc.isUseHandCursor?() => true or false
mc.setUseHandCursor?(flag)

Indicares or sets 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.

mc.isVisible?() => true or false
mc.setVisible?(flag)

Indicates or sets whether the movie clip specified by my_mc is visible. Movie clips that are not visible (_visible property set to false) are disabled.

mc.getWidth() => a Float
mc.setWidth(width)

Gets or sets the width of the movie clip in pixels.

mc.getX() => Float
mc.setX()

Gets or sets the x coordinate of the movie clip relative to the parent movie clip, in pixels.

mc.getXMouse() => a Float

Gets the x coordinate of the mouse position.

mc.getXScale() => a Float
mc.setXScale(xscale)

Gets or sets the horizontal scale (percentage) of the movie clip as applied from the registration point of the movie clip.

mc.getY() => Float
mc.setY()

Gets or sets the y coordinate of the movie clip relative to the parent movie clip, in pixels.

mc.getYMouse() => a Float

Gets the y coordinate of the mouse position.

mc.getYScale() => a Float
mc.setYScale(xscale)

Gets or sets the vertical scale (percentage) of the movie clip as applied from the registration point of the movie clip.

mc.setPosition(x,y)

Combines both setX and setY method calls.

mc.attachMovie(idName, newName, depth [, initObject:Object]) => MovieClip

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.
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().
Returns a reference to the newly created instance.

mc.createEmptyMovieClip(instanceName, depth) => MovieClip

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.
Creates an empty movie clip as a child of an existing movie clip.
Returns a reference to the newly created movie clip.

mc.createTextField(instanceName, depth, x, y, width, height)

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.
Creates a new, empty text field as a child of the movie clip specified by my_mc.

mc.duplicateMovieClip(newName,depth,initObj) => MovieClip

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.
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.
Returns a reference to the duplicated movie clip.

mc.getBounds(targetCoordinateSpace)

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.

mc.getBytesLoaded() => an Integer

Returns an integer indicating the number of bytes loaded.

mc.getDepth() => Number

Returns the depth of a movie clip instance

mc.getInstanceAtDepth(depth) => MovieClip

depth An integer that specifies the depth level to query.
Returns a reference to the MovieClip instance located at the specified depth, or nil if there is no movie clip at that depth.

mc.getNextHighestDepth() => Number

An integer that reflects the next available depth index that would render above all other objects on the same level and layer within mc.

mc.getSWFVersion() => Integer

An integer that specifies the Flash Player version that was targeted when the SWF file loaded into mc was published.

mc.getURL(URL [,window, variables])

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.
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.
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.

mc.globalToLocal(point)

point The name or identifier of an object created with the generic Object class. The object specifies the x and y coordinates as properties.
Converts the point object from Stage (global) coordinates to the movie clip's (local) coordinates.

mc.hitTest(x, y, shapeFlag) => true
mc.hitTest(target) => true

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.
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.
Usage 2: Evaluates the bounding boxes of the target and specified instance, and returns true if they overlap or intersect at any point.

mc.loadMovie(url [,variables])

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.
Loads SWF or JPEG files into a movie clip in Flash Player while the original SWF file is playing.

mc.loadVariables(url [, variables])

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.
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().

mc.localToGlobal(point)

Converts the point object from the movie clip's (local) coordinates to the Stage (global) coordinates.

mc.startDrag([lock, [left, top, right, bottom]])

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.

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.

mc.stopDrag()

Ends a MovieClip.startDrag() method.

mc.swapDepths(depth)
mc.swapDepths(target)

depth A number specifying the depth level where mc is to be placed.
target A string specifying the movie clip instance whose depth is swapped by the instance specified by mc. Both instances must have the same parent movie clip.
Swaps the stacking, or z-order (depth level), of the specified instance (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.

mc.unloadMovie()

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().

mc.removeMovieClip()

Removes a movie clip instance created with duplicateMovieClip(), MovieClip.duplicateMovieClip(), or MovieClip.attachMovie().

mc.beginFill([rgb [, alpha]])

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.
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.

mc.beginGradientFill(fillType,colors, alphas, ratios, matrix)

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.
1) a, b, c, d, e, f, g, h, i
2) matrixType, x, y, w, h, r.
NOTE: for more information, please refer to the Adobe website:
http://livedocs.adobe.com/flash/mx2004/main_7_2/00001489.html#wp3998401

mc.clear()

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.

mc.curveTo(controlX, controlY, anchorX, anchorY)

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.
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).

mc.endFill()

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.

mc.lineStyle([thickness [, rgb [, alpha]]])

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.
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.

mc.lineTo(x, y)

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.
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.

mc.moveTo(x, y)

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.
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.

TextField < Object

TextField is a native Actionscript class.
This class is augmented with wrappers for its original methods in order to box the original methods' primitive arguments and return values.

Class methods

TextField.getFontList() => an Array of Strings

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.

Instance methods

my_txt.getAlpha() => Float
my_txt.setAlpha(alpha)

Retrieves/sets the alpha transparency value of the text field specified by my_txt. 0 for fully transparent, 100 for full opacity.

my_txt.getAutoSize() => String or Boolean
my_txt.SetAutoSize(autoSize)

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". Refer to:
http://livedocs.adobe.com/flash/mx2004/main_7_2/00001772.html#4007698

my_txt.hasBackground?() => true or false
my_txt.setBackground?(flag)

If true, the text field has a background fill. If false, the text field has no background fill.

my_txt.getBackgroundColor() => an Integer
my_txt.setBackgroundColor(color)

Gets or sets the color of the text field background. Default is 0xFFFFFF (white). The color is only visible if the text field has a border.

my_txt.hasBorder?() => true or false
my_txt.setBorder?(flag)

If true, the text field has a border. If false, the text field has no border.

my_txt.getBorderColor() => Integer
my_txt.setBorderColor(color)

Gets or sets the color of the text field border, the Default is 0x000000 (black).

my_txt.getBottomScroll() => Integer

Retrieves an integer (one-based index) that indicates the bottommost line that is currently visible in my_txt.

my_txt.isEmbedFonts?() => true or false
my_txt.setEmbedFonts?(flag)

If true, renders the text field using embedded font outlines. If false, it renders the text field using device fonts.

my_txt.getHeight() => a Float
my_txt.setHeight(height)

Gets or sets the height of the text field, in pixels.

my_txt.getHScroll() => an Integer
my_txt.setHScroll(val)

Gets or sets the current horizontal scrolling position. If the hscroll property is 0, the text is not horizontally scrolled.

my_txt.isHtml?() => true or false
my_txt.setHtml?(flag)

If true, the text field is an HTML text field. If false, the text field is a non-HTML text field.

my_txt.getHtmlText() => a String
my_txt.setHtmlText(aString)

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.

my_txt.getLength() => an Integer

Returns the number of characters in a text field.

my_txt.getMaxChars() => an Integer
my_txt.setMaxChars(maxChars)

Gets or sets 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.

my_txt.getMaxHScroll() => an Integer

Gets the maximum horizontal scroll.

my_txt.getMaxScroll() => an Integer

Gets the maximum scroll.

my_txt.getMenu() => a ContextMenu object
my_txt.setMenu(contextMenu)

Gets or sets 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.

my_txt.isMouseWheelEnabled?() => true or false
my_txt.setMouseWheelEnabled?(flag)

Gets or sets 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.

my_txt.isMultiline?() => true or false
my_txt.setMultiline?(flag)

If true, the text field is multiline; if false, the text field is a single-line text field.

my_txt.getName() => a String
my_txt.setName(name)

Gets or sets the instance name of the text field specified by my_txt.

my_txt.getParent() => a MovieClip object
my_txt.setParent(parent)

Gets or sets a reference to the movie clip or object that contains the current text field or object.

my_txt.isPassword?() => true or false
my_txt.setPassword?(flag)

If 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.

my_txt.getQuality() => a String
my_txt.setQuality(quality)

Gets or sets the rendering quality used for a SWF file. Device fonts are always aliased and, therefore, are unaffected by the _quality property.
•"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.

my_txt.getRestrict() => nil or a String
my_txt.setRestrict(restrict)

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.
Refer to:
http://livedocs.adobe.com/flash/mx2004/main_7_2/00001807.html#4008356

my_txt.getRotation() => a Float
my_txt.setRotation(degree)

Gets or sets the rotation of the text field, in degrees, from its original orientation.

my_txt.getScroll() => an Integer
my_txt.setScroll(scroll)

Gets or sets the vertical position of text in a text field.

my_txt.isSelectable?() => true or false
my_txt.setSelectable?(flag)

Gets or sets a Boolean value that indicates whether the text field is selectable.

my_txt.isTabEnabled?() => true, false or undefined
my_txt.setTabEnabled?(tabEnabled)

Specifies whether my_txt is included in automatic tab ordering. It is undefined by default.
If 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 false, the object is not included in automatic or custom tab ordering, even if the tabIndex property is set.

my_txt.getTabIndex() => an Integer or undefined
my_txt.setTabIndex(tabIndex)

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.
Refer to:
http://livedocs.adobe.com/flash/mx2004/main_7_2/00001815.html#4008832

my_txt.getTarget() => a String

Returns 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.

my_txt.getText() => a String
my_txt.setText(text)
Gets or sets the current text in the text field.

my_txt.getTextColor => an Integer
my_txt.setTextColor(color)
Gets or sets the color of the text in a text field.

my_txt.getTextHeight() => a Float
my_txt.setTextHeight(height)
Gets or sets the height of the text.

my_txt.getTextWidth() => a Float
my_txt.setTextWidth(width)
Gets or sets the text width.

my_txt.getType() => a String
my_txt.setType(type)
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.

my_txt.getUrl() => a String
Retrieves the URL of the SWF file that created the text field.

my_txt.getVariable() => a String
my_txt.setVariable(varName)
Gets or sets the name of the variable that the text field is associated with.

my_txt.isVisible?() => true or false
my_txt.setVisible?(flag)
Gets or sets 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.

my_txt.getWidth() => a Float
my_txt.setWidth(width)
Gets or sets the width of the text field, in pixels.

my_txt.hasWordWrap?() => true or false
my_txt.setWordWrap?(flag)
Gets or sets a Boolean value that indicates if the text field has word wrap.

my_txt.getX() => a Float

my_txt.setX(x)

Gets or sets a float that sets the x coordinate of a text field relative to the local coordinates of the parent movie clip.

my_txt.getXMouse() => an Integer

Returns the x coordinate of the mouse position relative to the text field.

my_txt.getXScale() => a Float
my_txt.setXScale(scale)

Gets or sets 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).

my_txt.getY() => a Float
my_txt.setY(y)

Gets or sets 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 90º counterclockwise, the enclosed text field inherits a coordinate system that is rotated 90º counterclockwise. The text field's coordinates refer to the registration point position.

my_txt.getYMouse() => an Integer

Returns the y coordinate of the mouse position relative to the text field.

my_txt.getYScale() => a Float
my_txt.setYScale(scale)

Gets or sets 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).

my_txt.addListener(listener)

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.
Refer to:
http://livedocs.adobe.com/flash/mx2004/main_7_2/00001770.html#4007629

my_txt.getDepth() => an Integer

Returns the depth of a text field..

my_txt.removeListener(listener) => true or false

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.

my_txt.removeTextField()

Removes the text field specified by my_txt.

my_txt.replaceSel(text)

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.

Sprite < MovieClip

A sprite, in game development terminology, refers to a visual component, including its properties and methods.
A sprite instance is a generic visual component in Ruby On Flash, and can be visualized as a blank box on top of its parent sprite.

Example use:

```
emptySprite = Sprite.createInstance(self); #self refers to the main
program
emptySprite.setWidth(200);
emptySprite.setHeight(100);
emptySprite.setX(100);
emptySprite.setY(50);

emptySprite.beginFill();
#code to draw something
…
emptySprite.endFill();
```

Note that Sprite and its subclasses implements the Factory design pattern in object instantiation. The "initialize" method will be called by the factory method. In addition, the arguments to the "initialize" method are passed through the arguments of the createInstance class method.

For example:

```
class CustomSprite < Sprite
    def initialize(width, height, x,y)
        self.setWidth(width);
        self.setHeight(height);
        self.setX(x);
        self.setY(y);

        self.beginFill();
        #code to draw something
        …
        self.endFill();
    end
end
#same as previous example
#note that arguments to the initialize method are passed through
arguments to the createInstance class #method
emptySprite = CustomSprite.createInstance(self,200,100,100,50);
```

Sprite is the super class of ImageSprite and SwfSprite.

Class methods

Sprite.createInstance(parent,…) => a Sprite instance

Instantiates an instance of Sprite and attaches the newly created instance to the parent sprite.
The arguments for initialize method in subclasses can be passed through the arguments to this method, behind the parent parameter. For an example, refer to the class description.

Instance methods:

my_sprite.remove()

Removes my_sprite from the program.

ImageSprite < Sprite

ImageSprite represents a sprite that is represented as a sprite.
An image in Ruby On Flash can be loaded dynamically during runtime or as an embedded image (Refer to User's Guide on how to embed images).

To load an image dynamically:
ImageSprite.createInstanceFromFile(filename, parent);
However, note that only jpeg images can be loaded dynamically. Also, createInstanceFromFile returns an instance of class ImageSprite, thus dynamically loaded image sprites cannot have a custom-defined class.

To load an embedded image:
ImageSprite.createInstanceFromResource(imageId, parent), where imageId refers to the embedded image's resource id.
Similar to ImageSprite.createInstanceFromFile, createInstanceFromResource returns an instance of class ImageSprite, an thus embedded images loaded this way cannot have a custom-defined class.
To associate an image sprite with a class, first define a subclass of ImageSprite.
Thereafter, we can associate this class with the embedded image via a simple naming convention:
If the embedded image's id is "Blinky", then we simply call the subclass BlinkyImageSprite.
Alternatively, we can explicity specify the image id in the "@@imageId" class variable.
Finally, we'll instantiate an instance via a call to createInstance, similarly to Sprite.createInstance.

For example:

```
class BlinkyImageSprite < ImageSprite
    def initialize(x,y)
        self.setX(x);
        self.setY(y);
    end
end

blinky = BlinkyImageSprite.createInstance(self,100,100);
```

The following is equivalent, except that the image id was explicitly specified.

```
class YetAnotherSprite < ImageSprite
    @@imageId = "Blinky"
    def initialize(x,y)
        self.setX(x);
        self.setY(y);
    end
end

blinky = YetAnotherSprite.createInstance(self,100,100);
```

Class methods

ImageSprite.createInstanceFromFile(filename, parent) => an ImageSprite instance

Dynamically loads a jpeg image and creates an ImageSprite instance, then attaches this image sprite to parent.

ImageSprite.createInstanceFromResource(imageId, parent) => an ImageSprite instance

Loads an embedded image and creates an ImageSprite instance, then attaches this image sprite to parent.

ImageSprite.createInstance(parent, …) => an ImageSprite instance

Creates and instantiates a subclass associated with an embedded image, then attaches this image to parent.

The arguments for initialize method in subclasses can be passed through the arguments to this method, behind the parent parameter.

SwfSprite < Sprite

In flash, external flash movies can be loaded dynamically, as such, it is not surprising that Ruby On Flash offers this capability as well. A dynamically loaded flash movie can be visualized as a standalone sprite, thus Ruby On Flash refers to these as swf sprites and represents them with the class SwfSprite.
One way of loading swf sprites in Ruby On Flash:
SwfSprite.createInstanceFromFile(filename, parent)
Similar to ImageSprite.createInstanceFromFile, SwfSprite.createInstanceFromFile only returns an instance of class SwfSprite.
To associate a class with a swf sprite, we must first define a subclass of SwfSprite. This subclass will be linked to a swf file via a simple naming convention similar to that of ImageSprite. If the file is called "Inky.swf" (note that this means that this swf file must be in the same directory as the main program), then name the subclass as "InkySwfSprite". Alternatively, we can explicitly specify the swf file in the "@@swfFilename" class variable.

For example:

```
class InkySwfSprite < SwfSprite
    def initialize(x,y)
        self.setX(x);
        self.setY(y);
    end
end

inky =
InkySwfSprite.createInstance(self,1
00,100);
```

```
class SomeSwfSprite < SwfSprite
    @@swfFilename = "Inky.swf";
    def initialize(x,y)
        self.setX(x);
        self.setY(y);
    end
end

inky =
SomeSwfSprite.createInstance(self,1
00,100);
```

The above two code snippets are equivalent except that the code on the right specifies the swf file name.

Class methods:

SwfSprite.createInstanceFromFile(filename, parent) => a SwfSprite instance

Dynamically loads and creates a swf sprite, then attaches this sprite to parent.

SwfSprite.createInstance(parent,…) => a SwfSprite instance.

Creates and instantiates a subclass associated with a swf file, then attaches this sprite to parent.
The arguments for initialize method in subclasses can be passed through the arguments to this method, behind the parent parameter.