

ActionScript Dictionary

Trademarks

Afterburner, AppletAce, Attain, Attain Enterprise Learning System, Attain Essentials, Attain Objects for Dreamweaver, Authorware, Authorware Attain, Authorware Interactive Studio, Authorware Star, Authorware Synergy, Backstage, Backstage Designer, Backstage Desktop Studio, Backstage Enterprise Studio, Backstage Internet Studio, Design in Motion, Director, Director Multimedia Studio, Doc Around the Clock, Dreamweaver, Dreamweaver Attain, Drumbeat, Drumbeat 2000, Extreme 3D, Fireworks, Flash, Fontographer, FreeHand, FreeHand Graphics Studio, Generator, Generator Developer's Studio, Generator Dynamic Graphics Server, Knowledge Objects, Knowledge Stream, Knowledge Track, Lingo, Live Effects, Macromedia, Macromedia M Logo & Design, Macromedia Flash, Macromedia Xres, Macromind, Macromind Action, MAGIC, Mediamaker, Object Authoring, Power Applets, Priority Access, Roundtrip HTML, Scriptlets, SoundEdit, ShockRave, Shockmachine, Shockwave, Shockwave Remote, Shockwave Internet Studio, Showcase, Tools to Power Your Ideas, Universal Media, Virtuoso, Web Design 101, Whirlwind and Xtra are trademarks of Macromedia, Inc. and may be registered in the United States or in other jurisdictions including internationally. Other product names, logos, designs, titles, words or phrases mentioned within this publication may be trademarks, servicemarks, or tradenames of Macromedia, Inc. or other entities and may be registered in certain jurisdictions including internationally.

Third-Party Information

Speech compression and decompression technology licensed from Nellymoser, Inc. (www.nellymoser.com).



Sorenson™ Spark™ video compression and decompression technology licensed from
Sorenson Media, Inc.

This guide contains links to third-party Web sites that are not under the control of Macromedia, and Macromedia is not responsible for the content on any linked site. If you access a third-party Web site mentioned in this guide, then you do so at your own risk. Macromedia provides these links only as a convenience, and the inclusion of the link does not imply that Macromedia endorses or accepts any responsibility for the content on those third-party sites.

Apple Disclaimer

APPLE COMPUTER, INC. MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING THE ENCLOSED COMPUTER SOFTWARE PACKAGE, ITS MERCHANTABILITY OR ITS FITNESS FOR ANY PARTICULAR PURPOSE. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY PROVIDES YOU WITH SPECIFIC LEGAL RIGHTS. THERE MAY BE OTHER RIGHTS THAT YOU MAY HAVE WHICH VARY FROM STATE TO STATE.

Copyright © 2002 Macromedia, Inc. All rights reserved. This manual may not be copied, photocopied, reproduced, translated, or converted to any electronic or machine-readable form in whole or in part without prior written approval of Macromedia, Inc.

Acknowledgments

Director: Erick Vera

Producer: Wayne Wieseler

Writing: Jody Bleyle, JuLee Burdekin, Mary Burger, Dale Crawford, Marcelle Taylor

Instructional Design: Stephanie Gowin, Barbara Nelson

Editing: Rosana Francescato, Lisa Stanziano, Anne Szabla

Multimedia Design and Production: Aaron Begley, Benjamin Salles, Noah Zilberberg

Print Design and Production: Chris Basmajian, Caroline Branch

First Edition: March 2002

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

CONTENTS

Sample entry for most ActionScript elements	21
Sample entry for objects and components	22
Contents of the dictionary	23
— (decrement)	41
++ (increment)	42
! (logical NOT)	43
!= (inequality)	44
!== (strict inequality)	44
% (modulo)	45
%= (modulo assignment)	46
& (bitwise AND)	46
&& (short-circuit AND)	47
&= (bitwise AND assignment)	48
() (parentheses)	48
– (minus)	50
* (multiplication)	50
*= (multiplication assignment)	51
, (comma)	52
. (dot)	52
?: (conditional)	53
/ (division)	54
// (comment delimiter)	54
/* (comment delimiter)	55
/= (division assignment)	56
[] (array access)	56
^(bitwise XOR)	58
^= (bitwise XOR assignment)	59
{ } (object initializer)	60
(bitwise OR)	61
(logical OR)	61
= (bitwise OR assignment)	62
~ (bitwise NOT)	63
+ (addition)	64
+= (addition assignment)	65
< (less than)	66
<< (bitwise left shift)	67
<<= (bitwise left shift and assignment)	68
<= (less than or equal to)	68

<> (inequality)	70
= (assignment)	70
-= (subtraction assignment)	71
== (equality)	72
=== (strict equality)	73
> (greater than)	74
>= (greater than or equal to)	75
>> (bitwise right shift)	76
>>= (bitwise right shift and assignment)	77
>>> (bitwise unsigned right shift)	78
>>>= (bitwise unsigned right shift and assignment)	78
Accessibility (object)	79
Accessibility.isActive	79
add	80
and	80
arguments (object)	80
arguments.callee	81
arguments.caller	81
arguments.length	82
Array (object)	82
Array.concat	84
Array.join	85
Array.length	86
Array.pop	87
Array.push	87
Array.reverse	88
Array.shift	88
Array.slice	89
Array.sort	89
Array.sortOn	90
Array.splice	91
Array.toString	92
Array.unshift	92
asfunction	93
Boolean (function)	93
Boolean (object)	94
Boolean.toString	95
Boolean.valueOf	95
break	95
Button (object)	96
Event summary for the Button object	98
Button._alpha	98
Button.enabled	98
Button._focusrect	99
Button.getDepth	99
Button._height	99
Button._highquality	100
Button._name	100
Button.onDragOut	100
Button.onDragOver	100

Button.onKeyDown	101
Button.onKeyUp	102
Button.onKillFocus	102
Button.onPress	103
Button.onRelease	103
Button.onReleaseOutside	104
Button.onRollOut	104
Button.onRollOver	105
Button.onSetFocus	105
Button._parent	106
Button._quality	106
Button._rotation	107
Button._soundbuftime	107
Button.tabEnabled	107
Button.tabIndex	108
Button._target	108
Button.trackAsMenu	108
Button._url	109
Button.useHandCursor	109
Button._visible	109
Button._width	110
Button._x	110
Button._xmouse	110
Button._xscale	111
Button._y	111
Button._ymouse	111
Button._yscale	112
call	112
call function	112
case	113
chr	113
clearInterval	114
Color (object)	114
Color.getRGB	115
Color.getTransform	116
Color.setRGB	116
Color.setTransform	117
continue	118
CustomActions (object)	119
CustomActions.get	119
CustomActions.install	120
CustomActions.list	120
CustomActions.uninstall	121
Date (object)	121
Date.getDate	124
Date.getDay	124
Date.getFullYear	125
Date.getHours	125
Date.getMilliseconds	126
Date.getMinutes	126

Date.getMonth	127
Date.getSeconds	127
Date.getTime	128
Date.getTimezoneOffset	128
Date.getUTCDate	129
Date.getUTCDay	129
Date.getUTCFullYear	129
Date.getUTCHours	130
Date.getUTCMilliseconds	130
Date.getUTCMinutes	130
Date.getUTCMonth	131
Date.getUTCSeconds	131
Date.getYear	131
Date.setDate	132
Date.setFullYear	132
Date.setHours	132
Date.setMilliseconds	133
Date.setMinutes	133
Date.setMonth	134
Date.setSeconds	134
Date.setTime	134
Date.setUTCDate	135
Date.setUTCFullYear	135
Date.setUTCHours	136
Date.setUTCMilliseconds	136
Date.setUTCMinutes	136
Date.setUTCMonth	137
Date.setUTCSeconds	137
Date.setYear	138
Date.toString	138
Date.UTC	139
default	140
delete	140
do while	142
duplicateMovieClip	142
else	143
else if	144
#endinitclip	145
eq (equal—string specific)	145
escape	146
eval	146
evaluate	147
false	147
FCheckBox (component)	148
FCheckBox.setEnabled	149
FCheckBox.getLabel	149
FCheckBox.getValue	150
FCheckBox.registerSkinElement	150
FCheckBox.setChangeHandler	151
FCheckBox.setEnabled	152

FCheckBox.setLabel	152
FCheckBox.setLabelPlacement	153
FCheckBox.setSize	154
FCheckBox.setStyleProperty	154
FCheckBox.setValue	155
FComboBox (component)	155
FComboBox.addItem	157
FComboBox.addItemAt	158
FComboBox.setEnabled	158
FComboBox.getItemAt	159
FComboBox.getLength	160
FComboBox.getRowCount	160
FComboBox.getScrollPosition	161
FComboBox.getSelectedIndex	161
FComboBox.getSelectedItem	162
FComboBox.getValue	162
FComboBox.registerSkinElement	163
FComboBox.removeAll	164
FComboBox.removeItemAt	164
FComboBox.replaceItemAt	165
FComboBox.setChangeHandler	165
FComboBox.setDataProvider	166
FComboBox.setEditable	168
FComboBox.setEnabled	168
FComboBox.setItemSymbol	169
FComboBox.setRowCount	169
FComboBox.setSelectedIndex	170
FComboBox.setSize	170
FComboBox.setStyleProperty	171
FComboBox.setValue	171
FComboBox.sortItemsBy	172
FListBox (component)	172
FListBox.addItem	174
FListBox.addItemAt	175
FListBox.setEnabled	175
FListBox.getItemAt	176
FListBox.getLength	177
FListBox.getRowCount	177
FListBox.getScrollPosition	178
FListBox.getSelectedIndex	178
FListBox.getSelectedIndices	179
FListBox.getSelectedItem	179
FListBox.getSelectedItems	180
FListBox.getSelectMultiple	180
FListBox.getValue	181
FListBox.registerSkinElement	181
FListBox.removeAll	182
FListBox.removeItemAt	183
FListBox.replaceItemAt	183
FListBox.setAutoHideScrollBar	184

FListBox.setChangeHandler	184
FListBox.setDataProvider	185
FListBox.setEnabled	187
FListBox.setItemSymbol	187
FListBox.setRowCount	188
FListBox.setScrollPosition	188
FListBox.setSelectedIndex	189
FListBox.setSelectedIndices	189
FListBox.setSelectMultiple	190
FListBox.setSize	190
FListBox.setStyleProperty	191
FListBox.setWidth	191
FListBox.sortItemsBy	192
_focusrect	193
for	193
for..in	194
FPushButton (component)	196
FPushButton.setEnabled	196
FPushButton.getLabel	197
FPushButton.registerSkinElement	197
FPushButton.setClickHandler	198
FPushButton.setEnabled	199
FPushButton.setLabel	200
FPushButton.setSize	200
FPushButton.setStyleProperty	201
FRadioButton (component)	201
FRadioButton.getData	202
FRadioButton.setEnabled	203
FRadioButton.getLabel	204
FRadioButton.getState	204
FRadioButton.getValue	205
FRadioButton.registerSkinElement	205
FRadioButton.setChangeHandler	206
FRadioButton.setData	208
FRadioButton.setEnabled	208
FRadioButton.setGroupName	209
FRadioButton.setLabel	210
FRadioButton.setLabelPlacement	210
FRadioButton.setSize	211
FRadioButton.setState	212
FRadioButton.setStyleProperty	212
FRadioButton.setValue	213
FScrollBar (component)	214
FScrollBar.setEnabled	215
FScrollBar.getScrollPosition	215
FScrollBar.registerSkinElement	216
FScrollBar.setChangeHandler	217
FScrollBar.setEnabled	218
FScrollBar.setHorizontal	219
FScrollBar.setLargeScroll	219

FScrollBar.setScrollContent	220
FScrollBar.setScrollPosition	220
FScrollBar.setScrollProperties	221
FScrollBar.setScrollTarget	222
FScrollBar.setSize	222
FScrollBar.setSmallScroll	223
FScrollBar.setStyleProperty	223
FScrollPane (component)	224
FScrollPane.getPaneHeight	225
FScrollPane.getPaneWidth	225
FScrollPane.getScrollContent	226
FScrollPane.getScrollPosition	226
FScrollPane.loadScrollContent	227
FScrollPane.refreshPane	228
FScrollPane.registerSkinElement	229
FScrollPane.setDragContent	230
FScrollPane.setHScroll	230
FScrollPane.setScrollContent	231
FScrollPane.setScrollPosition	231
FScrollPane.setSize	232
FScrollPane.setStyleProperty	232
FScrollPane.setVScroll	233
FStyleFormat (object)	233
Property summary for the FStyleFormat object	234
FStyleFormat.addListener	236
FStyleFormat.applyChanges	236
FStyleFormat.arrow	237
FStyleFormat.background	238
FStyleFormat.backgroundDisabled	238
FStyleFormat.check	238
FStyleFormat.darkshadow	239
FStyleFormat.face	239
FStyleFormat.foregroundDisabled	240
FStyleFormat.highlight	240
FStyleFormat.highlight3D	241
FStyleFormat.radioDot	241
FStyleFormat.removeListener	241
FStyleFormat.scrollTrack	242
FStyleFormat.selection	243
FStyleFormat.selectionDisabled	243
FStyleFormat.selectionUnfocused	244
FStyleFormat.shadow	244
FStyleFormat.textAlign	245
FStyleFormat.textBold	245
FStyleFormat.textColor	246
FStyleFormat.textDisabled	246
FStyleFormat.textFont	246
FStyleFormat.textIndent	247
FStyleFormat.textItalic	247
FStyleFormat.textLeftMargin	248

FStyleFormat.textRightMargin	248
FStyleFormat.textSelected	249
FStyleFormat.textSize	249
FStyleFormat.textUnderline	250
Function (object).	250
Function.apply	250
Function.call	251
Function.prototype	252
fscommand	253
function	255
ge (greater than or equal to—string specific)	256
getProperty	256
getTimer	257
getURL	257
getVersion	258
_global	259
globalStyleFormat	259
gotoAndPlay	260
gotoAndStop	261
gt (greater than —string specific)	261
_highquality	262
if	262
ifframeLoaded	263
#include	264
#initclip	264
instanceof	265
int	266
isFinite	267
isNaN	267
Key (object).	268
Key.addListener	269
Key.BACKSPACE	270
Key.CAPSLOCK	270
Key.CONTROL	270
Key.DELETEKEY	270
Key.DOWN	270
Key.END	271
Key.ENTER	271
Key.ESCAPE	271
Key.getAscii	271
Key.getCode	272
Key.HOME	272
Key.INSERT	272
Key.isDown	273
Key.isToggled	273
Key.LEFT	273
Key.onKeyDown	274
Key.onKeyUp	274
Key.PGDN	274
Key.PGUP	275

Key.removeListener	275
Key.RIGHT	275
Key.SHIFT	275
Key.SPACE	276
Key.TAB	276
Key.UP	276
le (less than or equal to — string specific)	276
length	277
_level.	277
loadMovie	278
loadMovieNum	279
loadVariables	280
loadVariablesNum	282
LoadVars (object)	283
LoadVars.contentType	284
LoadVars.getBytesLoaded	284
LoadVars.getBytesTotal	285
LoadVars.load	285
LoadVars.loaded	286
LoadVars.onLoad	286
LoadVars.send	286
LoadVars.sendAndLoad	287
LoadVars.toString	288
lt (less than — string specific)	288
Math (object)	289
Math.abs	290
Math.acos	290
Math.asin	291
Math.atan	291
Math.atan2	292
Math.ceil	292
Math.cos	292
Math.E	293
Math.exp	293
Math.floor	294
Math.log	294
Math.LOG2E	294
Math.LOG10E	295
Math.LN2	295
Math.LN10	296
Math.max	296
Math.min	296
Math.PI	297
Math.pow	297
Math.random	298
Math.round	298
Math.sin	298
Math.sqrt	299
Math.SQRT1_2	299
Math.SQRT2	300

Math.tan	300
maxscroll	300
mbchr	301
mblength	301
mbord	302
mbsubstring	302
method	302
Mouse (object)	303
Mouse.addListener	304
Mouse.hide	304
Mouse.onMouseDown	305
Mouse.onMouseMove	305
Mouse.onMouseUp	306
Mouse.removeListener	306
Mouse.show	307
MovieClip (object)	307
MovieClip._alpha	310
MovieClip.attachMovie	311
MovieClip.beginFill	312
MovieClip.beginGradientFill	312
MovieClip.clear	317
MovieClip.createEmptyMovieClip	317
MovieClip.createTextField	318
MovieClip._currentframe	319
MovieClip.curveTo	320
MovieClip._droptarget	321
MovieClip.duplicateMovieClip	321
MovieClip.enabled	322
MovieClip.endFill	322
MovieClip.focusEnabled	323
MovieClip._focusrect	323
MovieClip._framesloaded	323
MovieClip.getBounds	324
MovieClip.getBytesLoaded	324
MovieClip.getBytesTotal	325
MovieClip.getDepth	325
MovieClip.getURL	325
MovieClip.globalToLocal	326
MovieClip.gotoAndPlay	327
MovieClip.gotoAndStop	327
MovieClip._height	327
MovieClip._highquality	328
MovieClip.hitArea	328
MovieClip.hitTest	328
MovieClip.lineStyle	329
MovieClip.lineTo	330
MovieClip.loadMovie	331
MovieClip.loadVariables	332
MovieClip.localToGlobal	333
MovieClip.moveTo	333

MovieClip._name	334
MovieClip.nextFrame	334
MovieClip.onData	335
MovieClip.onDragOut	335
MovieClip.onDragOver	336
MovieClip.onEnterFrame	336
MovieClip.onKeyDown	337
MovieClip.onKeyUp	337
MovieClip.onKillFocus	338
MovieClip.onLoad	338
MovieClip.onMouseDown	339
MovieClip.onMouseMove	339
MovieClip.onMouseUp	340
MovieClip.onPress	340
MovieClip.onRelease	341
MovieClip.onReleaseOutside	341
MovieClip.onRollOut	342
MovieClip.onRollOver	342
MovieClip.onSetFocus	343
MovieClip.onUnload	343
MovieClip._parent	344
MovieClip.play	344
MovieClip.prevFrame	344
MovieClip.removeMovieClip	345
MovieClip._rotation	345
MovieClip.setMask	345
MovieClip._soundbuftime	346
MovieClip.startDrag	346
MovieClip.stop	347
MovieClip.stopDrag	347
MovieClip.swapDepths	348
MovieClip.tabChildren	348
MovieClip.tabEnabled	349
MovieClip.tabIndex	349
MovieClip._target	350
MovieClip._totalframes	350
MovieClip.trackAsMenu	350
MovieClip.unloadMovie	350
MovieClip._url	351
MovieClip.useHandCursor	351
MovieClip._visible	351
MovieClip._width	352
MovieClip._x	352
MovieClip._xmouse	352
MovieClip._xscale	353
MovieClip._y	353
MovieClip._ymouse	353
MovieClip._yscale	354
NaN	354
ne (not equal — string specific)	354

new	355
newline	355
nextFrame	356
nextScene	356
not	357
null	357
Number (function)	358
Number (object)	358
Number.MAX_VALUE	360
Number.MIN_VALUE	360
Number.NaN	360
Number.NEGATIVE_INFINITY	360
Number.POSITIVE_INFINITY	360
Number.toString	361
Number.valueOf	361
Object (object)	362
Object.addProperty	362
Object.__proto__	364
Object.registerClass	364
Object.toString	367
Object.unwatch	367
Object.valueOf	367
Object.watch	368
onClipEvent	369
on	371
or	372
ord	372
_parent	372
parseFloat	373
parseInt	374
play	374
prevFrame	375
prevScene	376
print	376
printAsBitmap	377
printAsBitmapNum	378
printNum	379
_quality	380
random	381
removeMovieClip	381
return	382
_root	382
scroll	383
Selection (object)	383
Selection.addListener	384
Selection.getBeginIndex	384
Selection.getCaretIndex	385
Selection.getEndIndex	385
Selection.getFocus	386
Selection.onSetFocus	386

Selection.removeListener	386
Selection.setFocus	387
Selection.setSelection	388
set variable	388
setInterval	389
setProperty	390
Sound (object)	391
Sound.attachSound	393
Sound.duration	393
Sound.getBytesLoaded	394
Sound.getBytesTotal	394
Sound.getPan	394
Sound.getTransform	395
Sound.getVolume	395
Sound.loadSound	396
Sound.onLoad	396
Sound.onSoundComplete	397
Sound.position	397
Sound.setPan	398
Sound.setTransform	398
Sound.setVolume	400
Sound.start	401
Sound.stop	401
_soundbuftime	402
Stage (object)	402
Stage.addListener	403
Stage.align	404
Stage.height	404
Stage.onResize	405
Stage.removeListener	405
Stage.scaleMode	405
Stage.width	406
startDrag	406
stop	407
stopAllSounds	407
stopDrag	407
String (function)	408
" " (string delimiter)	409
String (object)	409
String.charAt	411
String.charCodeAt	411
String.concat	412
String.fromCharCode	412
String.indexOf	412
String.lastIndexOf	413
String.length	413
String.slice	414
String.split	414
String.substr	415
String.substring	416

String.toLowerCase	416
String.toUpperCase	416
substring	417
super	417
switch	418
System (object)	419
System.capabilities (object)	420
System.capabilities.hasAudioEncoder	421
System.capabilities.hasAccessibility	421
System.capabilities.hasAudio	421
System.capabilities.hasMP3	421
System.capabilities.language	422
System.capabilities.manufacturer	423
System.capabilities.os	423
System.capabilities.pixelAspectRatio	423
System.capabilities.screenColor	423
System.capabilities.screenDPI	424
System.capabilities.screenResolution.x	424
System.capabilities.screenResolution.y	424
System.capabilities.version	424
System.capabilities.hasVideoEncoder	425
targetPath	425
tellTarget	425
TextField (object)	426
TextField._alpha	429
TextField.addListener	429
TextField.autoSize	430
TextField.background	430
TextField.backgroundColor	430
TextField.border	431
TextField.borderColor	431
TextField.bottomScroll	431
TextField.embedFonts	432
TextField._focusrect	432
TextField.getDepth	432
TextField.getFontList	433
TextField.getNewTextFormat	433
TextField.getTextFormat	433
TextField._height	434
TextField._highquality	434
TextField.hscroll	435
TextField.html	435
TextField.htmlText	436
TextField.length	436
TextField.maxChars	436
TextField.maxhscroll	437
TextField.maxscroll	437
TextField.multiline	437
TextField._name	437
TextField.onChanged	438

TextField.onKillFocus	438
TextField.onScroller.	438
TextField.onSetFocus.	439
TextField._parent	439
TextField.password	439
TextField._quality	440
TextField.removeListener.	440
TextField.removeTextField.	441
TextField.replaceSel	441
TextField.restrict	442
TextField._rotation	443
TextField.scroll	443
TextField.selectable	443
TextField.setNewTextFormat.	444
TextField.setTextFormat	444
TextField._soundbuftime.	445
TextField.tabEnabled.	445
TextField.tabIndex.	446
TextField._target	446
TextField.text	447
TextField.textColor	447
TextField.textHeight	447
TextField.textWidth	447
TextField.type	448
TextField._url	448
TextField.variable	448
TextField._visible.	448
TextField._width	449
TextField.wordWrap	449
TextField._x.	449
TextField._xmouse.	450
TextField._xscale	450
TextField._y.	450
TextField._ymouse	451
TextField._yscale	451
TextFormat (object).	451
TextFormat.align	453
TextFormat.blockIndent	453
TextFormat.bold	454
TextFormat.bullet	454
TextFormat.color.	454
TextFormat.font	454
TextFormat.getTextExtent.	455
TextFormat.indent.	455
TextFormat.italic	455
TextFormat.leading	456
TextFormat.leftMargin	456
TextFormat.rightMargin	456
TextFormat.size	456
TextFormat.tabStops	457

TextFormat.target	457
TextFormat.underline	457
TextFormat.url	458
this	458
toggleHighQuality.	459
trace	459
true	460
typeof	461
undefined	461
unescape	462
unloadMovie	463
unloadMovieNum.	464
updateAfterEvent	464
var.	465
void.	465
while	466
with.	467
XML (object)	469
XML.appendChild	472
XML.attributes	472
XML.childNodes	473
XML.cloneNode	473
XML.contentType.	474
XML.createElement	474
XML.createTextNode	475
XML.docTypeDecl	475
XML.firstChild	476
XML.getBytesLoaded	476
XML.getBytesTotal	476
XML.hasChildNodes	477
XML.ignoreWhite.	477
XML.insertBefore	478
XML.lastChild	478
XML.load	478
XML.loaded	479
XML.nextSibling.	479
XML.nodeName	480
XML.nodeType.	480
XML.nodeValue	480
XML.onData.	481
XML.onLoad	481
XML.parentNode	482
XML.parseXML	483
XML.previousSibling	483
XML.removeNode	483
XML.send	484
XML.sendAndLoad.	484
XML.status	485
XML.toString	485
XML.xmlDecl.	486

XMLSocket (object)	487
XMLSocket.close	489
XMLSocket.connect	489
XMLSocket.onClose	490
XMLSocket.onConnect	490
XMLSocket.onData	492
XMLSocket.onXML	492
XMLSocket.send	493

ActionScript Dictionary

This dictionary describes the syntax and use of ActionScript elements in Macromedia Flash MX. To use examples in a script, copy the example code from the ActionScript Dictionary and paste it in the Actions panel in expert mode.

The dictionary lists all ActionScript elements—operators, keywords, statements, actions, properties, functions, objects, components, and methods. For an overview of all dictionary entries, see “Contents of the dictionary” on page 23; the tables in this section are a good starting point for looking up symbolic operators or methods whose object or component class you don’t know.

ActionScript follows the ECMA-262 Standard (the specification written by the European Computer Manufacturers Association) unless otherwise noted. Some Flash 5 (and earlier) ActionScript elements have been deprecated and replaced with new ActionScript elements that correspond to the ECMA Standard. It is recommended that you use the new Flash MX elements, although deprecated elements are still supported by Flash Player 5.

There are two types of entries in this dictionary:

- Individual entries for operators, keywords, functions, variables, properties, methods, and statements
- Object and component entries, which provide general information about built-in objects and the Flash components

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

Sample entry for most ActionScript elements

The following sample dictionary entry explains the conventions used for all ActionScript elements that are not objects or components.

Entry title

All entries are listed alphabetically. The alphabetization ignores capitalization, leading underscores, and so on.

Availability

This section tells which versions of the Flash Player support the element. This is not the same as the version of Flash used to author the content. For example, if you use the Flash MX authoring tool to create content for Flash Player 5, you can only use ActionScript elements that are available to Flash Player 5.

Usage

This section provides correct syntax for using the `ActionScript` element in your code. The required portion of the syntax is in `code font`, and the user provided code is in *italicized code font*. Brackets ([]) indicate optional parameters.

Parameters

This section describes any parameters listed in the syntax.

Returns

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

Description

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

Example

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

See also

This section lists related `ActionScript` dictionary entries.

Sample entry for objects and components

The following sample dictionary entry explains the conventions used for built-in `ActionScript` objects and components. Objects and components are listed alphabetically with all other elements in the dictionary. The Flash components are listed as `FCheckBox`, `FComboBox`, and so on.

Entry title

The entry title provides the name of the object or component. The object or component name is followed by a paragraph containing general descriptive information.

Method and property summary tables

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

Constructor

If an object or component requires you to use a constructor to access its methods and properties, the constructor is described in each object or component entry. This description has all of the standard elements (syntax, description, and so on) of other dictionary entries.

Method and property listings

The methods and properties of an object or component are listed alphabetically after the object or component entry.

Contents of the dictionary

All dictionary entries are listed alphabetically. However, some operators are symbols and are presented in ASCII order. In addition, methods that are associated with an object or component are listed along with the object or component name—for example, the `abs` method of the `Math` object is listed as `Math.abs`, and the `getValue` method of the `FRadioButton` component is listed as `FRadioButton.getValue`.

The following two tables help you locate these elements. The first table lists the symbolic operators in the order in which they occur in the dictionary. The second table lists all other ActionScript elements.

Note: For precedence and associativity of operators, see Appendix A, “Operator Precedence and Associativity” in the “Using Flash” book.

Symbolic operators

--	-- (decrement)
++	++ (increment)
!	! (logical NOT)
!=	!= (inequality)
!==	!== (strict inequality)
%	% (modulo)
%=	%= (modulo assignment)
&	& (bitwise AND)
&&	&& (short-circuit AND)
&=	&= (bitwise AND assignment)
()	() (parentheses)
-	- (minus)
*	* (multiplication)
*=	*= (multiplication assignment)
,	, (comma)
.	. (dot)
?:	?: (conditional)
/	/ (division)
//	// (comment delimiter)
/*	/* (comment delimiter)
/=	/= (division assignment)
[]	[] (array access)
^	^(bitwise XOR)
^=	^= (bitwise XOR assignment)
{}	{ } (object initializer)
	(bitwise OR)

Symbolic operators

	(logical OR)
=	= (bitwise OR assignment)
~	~ (bitwise NOT)
+	+ (addition)
+=	+= (addition assignment)
<	< (less than)
<<	<< (bitwise left shift)
<<=	<<= (bitwise left shift and assignment)
<=	<= (less than or equal to)
<>	<> (inequality)
=	= (assignment)
-=	-= (subtraction assignment)
==	== (equality)
===	=== (strict equality)
>	> (greater than)
>=	>= (greater than or equal to)
>>	>> (bitwise right shift)
>>=	>>= (bitwise right shift and assignment)
>>>	>>> (bitwise unsigned right shift)
>>>=	>>>= (bitwise unsigned right shift and assignment)

The following table lists all **ActionScript** elements that are not symbolic operators.

ActionScript element	See entry
abs	Math.abs
acos	Math.acos
add	add
addItem	FComboBox.addItem, FListBox.addItem
addItemAt	FComboBox.addItemAt, FListBox.addItem
addListener	FStyleFormat.addListener, Key.addListener, Mouse.addListener, Selection.addListener, Stage.addListener, TextField.addListener
addProperty	Object.addProperty
and	and
align	Stage.align, TextFormat.align
_alpha	MovieClip._alpha, Button._alpha, TextField._alpha
appendChild	XML.appendChild
apply	Function.apply

ActionScript element	See entry
applyChanges	FStyleFormat.applyChanges
Arguments	arguments (object)
Array	Array (object)
arrow	FStyleFormat.arrow
asfunction	asfunction
asin	Math.asin
atan	Math.atan
atan2	Math.atan2
attachMovie	MovieClip.attachMovie
attachSound	Sound.attachSound
attributes	XML.attributes
autosize	TextField.autoSize
background	FStyleFormat.background, TextField.background
backgroundColor	TextField.backgroundColor
backgroundDisabled	FStyleFormat.backgroundDisabled
BACKSPACE	Key.BACKSPACE
beginFill	MovieClip.beginFill
beginGradientFill	MovieClip.beginGradientFill
blockIndent	TextFormat.blockIndent
bold	TextFormat.bold
Boolean	Boolean (function), Boolean (object)
border	TextField.border
borderColor	TextField.borderColor
bottomScroll	TextField.bottomScroll
break	break
bullet	TextFormat.bullet
Button	Button (object)
call	call, Function.call
call function	call function
callee	arguments.callee
caller	arguments.caller
capabilities	System.capabilities (object)
CAPSLock	Key.CAPSLock
case	case
ceil	Math.ceil
charAt	String.charAt

ActionScript element	See entry
charCodeAt	String.charCodeAt
check	FStyleFormat.check
childNodes	XML.childNodes
chr	chr
clear	MovieClip.clear
clearInterval	clearInterval
cloneNode	XML.cloneNode
close	XMLSocket.close
Color	Color (object), TextFormat.color
concat	Array.concat, String.concat
connect	XMLSocket.connect
constructor	Array (object), Boolean (object), Color (object), Date (object), Number (function), Object (object), Sound (object), String (object), XML (object), XMLSocket (object)
contentType	LoadVars.contentType, XML.contentType
ccontinue	continue
CONTROL	Key.CONTROL
cos	Math.cos
createElement	XML.createElement
createEmptyMovieClip	MovieClip.createEmptyMovieClip
createTextField	MovieClip.createTextField
createTextNode	XML.createTextNode
_currentframe	MovieClip._currentframe
curveTo	MovieClip.curveTo
Date	Date (object)
darkshadow	FStyleFormat.darkshadow
default	default
delete	delete
DELETEKEY	Key.DELETEKEY
docTypeDecl	XML.docTypeDecl
do while	do while
DOWN	Key.DOWN
_droptarget	MovieClip._droptarget
duplicateMovieClip	duplicateMovieClip, MovieClip.duplicateMovieClip
duration	Sound.duration
E	Math.E
#endinitclip	#endinitclip

ActionScript element	See entry
else	else
else if	else if
embedFonts	TextField.embedFonts
enabled	Button.enabled, MovieClip.enabled
END	Key.END
endFill	MovieClip.endFill
ENTER	Key.ENTER
eq	eq (equal-string specific)
escape (function)	escape
ESCAPE (constant)	Key.ESCAPE
eval	eval
evaluate	evaluate
exp	Math.exp
face	FStyleFormat.face
false	false
FCheckBox	FCheckBox (component)
FComboBox	FListBox (component)
firstChild	XML.firstChild
FListBox	FListBox (component)
floor	Math.floor
focusEnabled	MovieClip.focusEnabled
_focusrect	_focusrect, Button._focusrect, TextField._focusrect, MovieClip._focusrect
font	TextFormat.font
for	for
for..in	for..in
foregroundDisabled	FStyleFormat.foregroundDisabled
FPushButton	FPushButton (component)
FRadioButton	FPushButton (component)
_framesloaded	MovieClip._framesloaded
fromCharCode	String.fromCharCode
fscommand	fscommand
FScrollBar	FScrollBar (component)
FScrollPane	FScrollPane (component)
FStyleFormat	FStyleFormat (object)
function	function, Function (object)

ActionScript element	See entry
ge	ge (greater than or equal to—string specific)
get	CustomActions.get
getAscii	Key.getAscii
getBeginIndex	Selection.getBeginIndex
getBounds	MovieClip.getBounds
getBytesLoaded	LoadVars.getBytesLoaded, MovieClip.getBytesLoaded, Sound.getBytesLoaded, XML.getBytesLoaded
getBytesTotal	LoadVars.getBytesTotal, MovieClip.getBytesTotal, Sound.getBytesTotal, XML.getBytesTotal
getCaretIndex	Selection.getCaretIndex
getCode	Key.getCode
getData	FRadioButton.getData
getDate	Date.getDate
getDay	Date.getDay
getDepth	Button.getDepth, MovieClip.getDepth, TextField.getDepth
getEnabled	FCheckBox.getEnabled, FComboBox.getEnabled, FListBox.getEnabled, FPushButton.getEnabled, FRadioButton.getEnabled, FScrollBar.getEnabled
getEndIndex	Selection.getEndIndex
getFocus	Selection.getFocus
getFontList	TextField.getFontList
getFullYear	Date.getFullYear
getHours	Date.getHours
getItemAt	FComboBox.getItemAt, FListBox.addItemAt
getLabel	FCheckBox.getLabel, FPushButton.getLabel, FRadioButton.getLabel
getLength	FComboBox.getLength, FListBox.getLength
getMilliseconds	Date.getMilliseconds
getMinutes	Date.getMinutes
getMonth	Date.getMonth
getNewTextFormat	TextField.getNewTextFormat
getPan	Sound.getPan
getPaneHeight	FScrollPane.getPaneHeight
getPaneWidth	FScrollPane.getPaneWidth
getProperty	getProperty
getRowCount	FComboBox.getRowCount, FListBox.getRowCount
getRGB	Color.getRGB
getScrollContent	FScrollPane.getScrollContent

ActionScript element	See entry
getScrollPosition	FComboBox.getScrollPosition, FListBox.getScrollPosition, FScrollBar.getScrollPosition, FScrollPane.getScrollPosition
getSeconds	Date.getSeconds
getSelectedIndex	FComboBox.getSelectedIndex, FListBox.getSelectedIndex
getSelectedIndices	FListBox.getSelectedIndices
getSelectedItem	FComboBox.getSelectedItem, FListBox.getSelectedItem
getSelectedItems	FListBox.getSelectedItem
getSelectMultiple	FListBox.getSelectMultiple
getState	FRadioButton.getState
getTextExtent	TextFormat.getTextExtent
getTextFormat	TextField.getTextFormat
getTime	Date.getTime
getTimer	getTimer
getTimezoneOffset	Date.getTimezoneOffset
getTransform	Color.getTransform, Sound.getTransform
getURL	getURL, MovieClip.getURL
getUTCDate	Date.getUTCDate
getUTCDay	Date.getUTCDay
getUTCFullYear	Date.getUTCFullYear
getUTCHours	Date.getUTCHours
getUTCMilliseconds	Date.getUTCMilliseconds
getUTCMinutes	Date.getUTCMinutes
getUTCMonth	Date.getUTCMonth
getUTCSeconds	Date.getUTCSeconds
getValue	FCheckBox.getValue, FComboBox.getValue, FListBox.getValue, FRadioButton.getValue
getVersion	getVersion
getVolume	Sound.getVolume
getYear	Date.getYear
_global	_global
globalStyleFormat	globalStyleFormat
globalToLocal	MovieClip.globalToLocal
goto	gotoAndPlay, gotoAndStop
gotoAndPlay	gotoAndPlay, MovieClip.gotoAndPlay
gotoAndStop	gotoAndStop, MovieClip.gotoAndStop
gt	gt (greater than —string specific)
hasAccessibility	System.capabilities.hasAccessibility

ActionScript element	See entry
hasAudio	System.capabilities.hasAudio
hasAudioEncoder	System.capabilities.hasAudioEncoder
hasMP3	System.capabilities.hasMP3
hasVideoEncoder	System.capabilities.hasVideoEncoder
hasChildNodes	XML.hasChildNodes
height	Stage.height
_height	MovieClip._height, TextField._height, Button._height
hide	Mouse.hide
highlight	FStyleFormat.highlight
highlight3D	FStyleFormat.highlight3D
_highquality	_highquality, Button._highquality, MovieClip._highquality, TextField._highquality
hitArea	MovieClip.hitArea
hitTest	MovieClip.hitTest
HOME	Key.HOME
hscroll	TextField.hscroll
html	TextField.html
htmlText	TextField.htmlText
if	if
ifFrameLoaded	ifFrameLoaded
ignoreWhite	XML.ignoreWhite
#include	#include
indent	TextFormat.indent
indexOf	String.indexOf
#initclip	#initclip
INSERT	Key.INSERT
insertBefore	XML.insertBefore
install	CustomActions.install
instanceof	instanceof
int	int
isActive	Accessibility.isActive
isDown	Key.isDown
isFinite	isFinite
isNaN	isNaN
isToggled	Key.isToggled
italic	TextFormat.italic

ActionScript element	See entry
join	Array.join
Key	Key (object)
language	System.capabilities.language
lastChild	XML.lastChild
lastIndexOf	String.lastIndexOf
le	le (less than or equal to – string specific)
leading	TextFormat.leading
LEFT	Key.LEFT
leftMargin	TextFormat.leftMargin
length	arguments.length, Array.length, String.length, Sound.loadSound, TextField.length
level	_level
lineStyle	MovieClip.lineStyle
lineTo	MovieClip.lineTo
list	CustomActions.uninstall
LN2	Math.LN2
LN10	Math.LN10
load	XML.load, LoadVars.load
loaded	XML.loaded, LoadVars.loaded
loadMovie	loadMovie, MovieClip.loadMovie
loadMovieNum	loadMovieNum
loadScrollContent	FScrollPane.loadScrollContent
loadSound	Sound.loadSound
loadVariables	loadVariables, MovieClip.loadVariables
loadVariablesNum	loadVariablesNum
LoadVars	LoadVars (object)
localToGlobal	MovieClip.localToGlobal
log	Math.log
LOG2E	Math.LOG2E
LOG10E	Math.LOG10E
lt	lt (less than – string specific)
manufacturer	System.capabilities.manufacturer
Math	Math (object)
max	Math.max
maxChars	TextField.maxChars
maxhscroll	TextField.maxhscroll

ActionScript element	See entry
maxscroll	maxscroll, TextField.maxscroll
MAX_VALUE	Number.MAX_VALUE
mbchr	mbchr
mblength	mblength
mbord	mbord
mbsubstring	mbsubstring
method	method
min	Math.min
MIN_VALUE	Number.MIN_VALUE
Mouse	Mouse (object)
moveTo	MovieClip.moveTo
MovieClip	MovieClip (object)
multiline	TextField.multiline
_name	MovieClip._name, TextField._name, Button._name
NaN	NaN, Number.NaN
ne	ne (not equal — string specific)
NEGATIVE_INFINITY	Number.NEGATIVE_INFINITY
new (operator)	new
newline	newline
nextFrame	nextFrame, MovieClip.nextFrame
nextScene	nextScene
nextSibling	XML.nextSibling
nodeName	XML.nodeName
nodeType	XML.nodeType
nodeValue	XML.nodeValue
not	not
null	null
Number	Number (function), Number (object)
Object	Object (object)
on	on
onClipEvent	onClipEvent
onClose	XMLSocket.onClose
onChanged	TextField.onChanged
onConnect	XMLSocket.onConnect
onData	XML.onData, XMLSocket.onData
onDragOut	Button.onDragOut, MovieClip.onDragOut

ActionScript element	See entry
onDragOver	Button.onDragOver, MovieClip.onDragOver
onEnterFrame	MovieClip.onEnterFrame
onKeyDown	Button.onKeyDown, Key.onKeyDown, MovieClip.onKeyDown
onKeyUp	Button.onKeyUp, Key.onKeyUp, MovieClip.onKeyUp
onKillFocus	Button.onKillFocus, MovieClip.onKillFocus, TextField.onKillFocus
onLoad	LoadVars.onLoad, MovieClip.onLoad, Sound.onLoad, XML.onLoad
onMouseDown	Mouse.onMouseDown, MovieClip.onMouseDown
onMouseMove	Mouse.onMouseMove, MovieClip.onMouseMove
onMouseUp	Mouse.onMouseUp, MovieClip.onMouseUp
onPress	Button.onPress, MovieClip.onPress
onRelease	Button.onRelease, MovieClip.onRelease
onReleaseOutside	Button.onReleaseOutside, MovieClip.onReleaseOutside
onResize	Stage.onResize
onRollOut	Button.onRollOut
onRollOver	Button.onRollOver
onScroller	TextField.onScroller
onSetFocus	Button.onSetFocus, MovieClip.onSetFocus, Selection.onSetFocus, TextField.onSetFocus
onSort	Array.pop
onSoundComplete	Sound.onSoundComplete
onUnload	MovieClip.onUnload
onXML	XMLSocket.onXML
or (logical OR)	or
ord	ord
os	System.capabilities.os
_parent	_parent, Button._parent, MovieClip._parent, TextField._parent
parentNode	XML.parentNode
parseFloat	parseFloat
parseInt	parseInt
parseXML	XML.parseXML
password	TextField.password
PGDN	Key.PGDN
PGUP	Key.PGUP
PI	Math.PI
pixelAspectRatio	System.capabilities.pixelAspectRatio
play	play, MovieClip.play

ActionScript element	See entry
pop	Array.pop
position	Sound.position
POSITIVE_INFINITY	Number.POSITIVE_INFINITY
pow	Math.pow
prevFrame	prevFrame, MovieClip.prevFrame
previousSibling	XML.previousSibling
prevScene	prevScene
print	print
printAsBitmap	printAsBitmap
printAsBitmapNum	printAsBitmapNum
printNum	printNum
__proto__	Object.__proto__
push	Array.push
_quality	_quality, TextField._quality, Button._quality
radioDot	FStyleFormat.radioDot
random	random, Math.random
refreshPane	FScrollPane.refreshPane
registerClass	Object.registerClass
registerSkinElement	FCheckBox.registerSkinElement, FComboBox.registerSkinElement, FListBox.registerSkinElement, FPushButton.registerSkinElement, FRadioButton.registerSkinElement, FScrollPane.registerSkinElement, FScrollPane.registerSkinElement
removeAll	FComboBox.removeAll, FListBox.removeAll
removeItemAt	FComboBox.removeItemAt, FListBox.removeItemAt
removeListener	FStyleFormat.removeListener, Key.removeListener, Mouse.removeListener, Selection.removeListener, Stage.removeListener, TextField.removeListener
removeMovieClip	removeMovieClip, MovieClip.removeMovieClip
removeNode	XML.removeNode
removeTextField	TextField.removeTextField
replaceItemAt	FComboBox.replaceItemAt, FListBox.replaceItemAt
replaceSel	TextField.replaceSel
resolutionX	Capabilities.screenResolutionX
resolutionY	Capabilities.screenResolutionY
restrict	TextField.restrict
return	return
reverse	Array.reverse

ActionScript element	See entry
RIGHT	Key.RIGHT
rightMargin	TextFormat.rightMargin
_root	_root
_rotation	MovieClip._rotation, Button._rotation, TextField._rotation
round	Math.round
scaleMode	Stage.scaleMode
screenColor	System.capabilities.screenColor
screenDPI	System.capabilities.screenDPI
screenResolution.x	System.capabilities.screenResolution.x
screenResolution.y	System.capabilities.screenResolution.y
scroll	scroll, TextField.scroll
scrollTrack	FStyleFormat.scrollTrack
selectable	TextField.selectable
selection	FStyleFormat.selection
Selection	Selection (object)
selectionDisabled	FStyleFormat.selectionDisabled
selectionUnfocused	FStyleFormat.selectionUnfocused
send	LoadVars.send, XML.send, XMLSocket.send
sendAndLoad	LoadVars.sendAndLoad, XML.sendAndLoad
set variable	set variable
setAutoHideScrollBar	FListBox.setAutoHideScrollBar
setChangeHandler	FCheckBox.setChangeHandler, FComboBox.setChangeHandler, FListBox.setChangeHandler, FRadioButton.setChangeHandler, FScrollBar.setChangeHandler
setClickHandler	FPushButton.setClickHandler
setData	FRadioButton.setData
setDataProvider	FComboBox.setDataProvider, FListBox.setDataProvider
setDate	Date.setDate
setDragContent	FScrollPane.setDragContent
setEditable	FComboBox.setEditable
setEnabled	FCheckBox.setEnabled, FComboBox.setEnabled, FListBox.setEnabled, FPushButton.setEnabled, FRadioButton.setEnabled, FScrollBar.setEnabled
setFocus	Selection.setFocus
setFullYear	Date.setFullYear
setGroupName	FRadioButton.setGroupName
setHorizontal	FScrollBar.setHorizontal

ActionScript element	See entry
setHours	Date.setHours
setHScroll	FScrollPane.setHScroll
setInterval	setInterval
setItemSymbol	FComboBox.setItemSymbol, FListBox.setItemSymbol
setLabel	FCheckBox.setLabel, FPushButton.setLabel, FRadioButton.setLabel
setLabelPlacement	FCheckBox.setLabelPlacement, FRadioButton.setLabelPlacement
setLargeScroll	FScrollBar.setLargeScroll
setMask	MovieClip.setMask
setMilliseconds	Date.setMilliseconds
setMinutes	Date.setMinutes
setMonth	Date.setMonth
setNewTextFormat	TextField.setNewTextFormat
setPan	Sound.setPan
setProperty	setProperty
setRGB	Color.setRGB
setRowCount	FComboBox.setRowCount, FListBox.setRowCount
setScrollContent	FScrollBar.setSize, FScrollPane.setScrollContent
setScrollPosition	FListBox.setScrollPosition, FScrollBar.setScrollPosition, FScrollPane.setScrollPosition
setScrollProperties	FScrollBar.setScrollProperties
setScrollTarget	FScrollBar.setScrollTarget
setSeconds	Date.setSeconds
setSelectedIndex	FComboBox.setSelectedIndex, FListBox.setSelectedIndex
setSelectedIndices	FListBox.setSelectedIndices
setSelection	Selection.setSelection
setSelectMultiple	FListBox.setSelectMultiple
setSize	FCheckBox.setSize, FComboBox.setSize, FListBox.setSize, FPushButton.setSize, FScrollBar.setSize, FScrollPane.setSize
setSmallScroll	FScrollBar.setSmallScroll
setState	FRadioButton.setState
setStyleProperty	FCheckBox.setStyleProperty, FComboBox.setStyleProperty, FListBox.setStyleProperty, FPushButton.setStyleProperty, FRadioButton.setStyleProperty, FScrollBar.setStyleProperty, FScrollPane.setStyleProperty
setTextFormat	TextField.setTextFormat
setTime	Date.setTime
setTransform	Color.setTransform, Sound.setTransform

ActionScript element	See entry
setUTCDate	Date.setUTCDate
setUTCFullYear	Date.setUTCFullYear
setUTCHours	Date.setUTCHours
setUTCMilliseconds	Date.setUTCMilliseconds
setUTCMinutes	Date.setUTCMinutes
setUTCMonth	Date.setUTCMonth
setUTCSeconds	Date.setUTCSeconds
setValue	FCheckBox.setValue, FComboBox.setValue, FRadioButton.setValue
setVolume	Sound.setVolume
setVScroll	FScrollPane.setVScroll
setWidth	FListBox.setWidth
setYear	Date.setYear
shadow	FStyleFormat.shadow
shift (method)	Array.shift
SHIFT (constant)	Key.SHIFT
show	Mouse.show
sin	Math.sin
size	TextFormat.size
slice	Array.slice, String.slice
sort	Array.sort
sortItemsBy	FComboBox.sortItemsBy, FListBox.sortItemsBy
Sound	Sound (object)
_soundbuftime	_soundbuftime, TextField._soundbuftime, MovieClip._soundbuftime, Button._soundbuftime
SPACE	Key.SPACE
splice	Array.splice
split	String.split
sqrt	Math.sqrt
SQRT1_2	Math.SQRT1_2
SQRT2	Math.SQRT2
start	Sound.start
startDrag	startDrag, MovieClip.startDrag
status	XML.status
stop	stop, MovieClip.stop, Sound.stop
stopAllSounds	stopAllSounds
stopDrag	stopDrag, MovieClip.stopDrag

ActionScript element	See entry
String	String (function), String (object)
substr	String.substr
substring	substring, String.substring
super	super
swapDepths	MovieClip.swapDepths
switch	switch
System	System (object)
TAB	Key.TAB
tabChildren	MovieClip.tabChildren
tabEnabled	Button.tabEnabled, TextField.tabEnabled, MovieClip.tabEnabled
tabIndex	Button.tabIndex, MovieClip.tabIndex, TextField.tabIndex
tabStops	TextFormat.tabStops
tan	Math.tan
target	TextFormat.target
_target	Button._target, MovieClip._target, TextField._target
targetPath	targetPath
tellTarget	tellTarget
text	TextField.text
textAlign	FStyleFormat.textAlign
textBold	FStyleFormat.textBold
textColor	FStyleFormat.textColor, TextField.textColor
textDisabled	FStyleFormat.textDisabled
TextField	TextField (object)
textFont	FStyleFormat.textFont
TextFormat	TextFormat (object)
textHeight	TextField.textHeight
textIndent	FStyleFormat.textIndent
textItalic	FStyleFormat.textItalic
textLeftMargin	FStyleFormat.textLeftMargin
textRightMargin	FStyleFormat.textRightMargin
textSelected	FStyleFormat.textSelected
textSize	FStyleFormat.textSize
textUnderline	FStyleFormat.textUnderline
textWidth	TextField.textWidth
this	this
toggleHighQuality	toggleHighQuality

ActionScript element	See entry
toLowerCase	String.toLowerCase
toString	Array.toString, Boolean.toString, Date.toString, Number.toString, Object.toString, XML.toString
_totalframes	MovieClip._totalframes
toUpperCase	String.toUpperCase
trace	trace
trackAsMenu	Button.trackAsMenu, MovieClip.trackAsMenu
true	true
type	TextField.type
typeof	typeof
undefined	undefined
underline	TextFormat.underline
unescape	unescape
uninstall	CustomActions.uninstall
unloadMovie	unloadMovie, MovieClip.unloadMovie
unloadMovieNum	unloadMovieNum
unshift	Array.unshift
unwatch	Object.unwatch
UP	Key.UP
updateAfterEvent	updateAfterEvent
url	TextFormat.url
_url	MovieClip._url, TextField._url, Button._url
useHandCursor	Button.useHandCursor, MovieClip.useHandCursor
UTC	Date.UTC
valueOf	Boolean.valueOf, Number.valueOf, Object.valueOf
var	var
variable	TextField.variable
version	System.capabilities.version
_visible	MovieClip._visible, Button._visible, TextField._visible
void	void
watch	Object.watch
while	while
width	Stage.width
_width	MovieClip._width, TextField._width, Button._width
with	with
wordwrap	TextField.wordWrap

ActionScript element	See entry
<code>_x</code>	<code>Button._x</code> , <code>MovieClip._x</code> , <code>TextField._x</code>
<code>XML</code>	<code>XML</code> (object)
<code>xmlDecl</code>	<code>XML.xmlDecl</code>
<code>XMLSocket</code>	<code>XMLSocket</code> (object)
<code>_xmouse</code>	<code>Button._xmouse</code> , <code>MovieClip._xmouse</code> , <code>TextField._xmouse</code>
<code>_xscale</code>	<code>Button._xscale</code> , <code>MovieClip._xscale</code> , <code>TextField._xscale</code>
<code>_y</code>	<code>Button._y</code> , <code>MovieClip._y</code> , <code>TextField._y</code>
<code>_ymouse</code>	<code>Button._ymouse</code> , <code>MovieClip._ymouse</code> , <code>TextField._ymouse</code>
<code>_yscale</code>	<code>Button._yscale</code> , <code>MovieClip._yscale</code> , <code>TextField._yscale</code>

— (decrement)

Availability

Flash Player 4.

Usage

--expression

expression--

Parameters

None.

Returns

Nothing.

Description

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

Example

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

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

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

```
x = 3;  
y = x--  
//y is equal to 3
```

++ (increment)

Availability

Flash Player 4.

Usage

`++expression`

`expression++`

Parameters

None.

Returns

Nothing.

Description

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

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

```
x = 1;
y = ++x
//y is equal to 2
```

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

```
x = 1;
y = x++;
//y is equal to 1
```

Example

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

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

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

```
var a = [];
var i = 0;
while (i < 10) {
  a.push(++i);
}
trace(a.join());
```

This script displays the following result in the Output window:

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

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

```
var a = [];  
var i = 0;  
while (i < 10) {  
  a.push(i++);  
}  
trace(a.join());
```

This script displays the following result in the Output window:

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

! (logical NOT)

Availability

Flash Player 4.

Usage

!expression

Parameters

None.

Returns

Nothing.

Description

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

The following expressions illustrate the result of using the ! operator:

!true returns *false*

!false returns *true*

Example

In the following example, the variable *happy* is set to *false*. The *if* condition evaluates the condition *!happy*, and if the condition is *true*, the *trace* action sends a string to the Output window.

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

!= (inequality)

Availability

Flash Player 5.

Usage

expression1 != *expression2*

Parameters

None.

Returns

Nothing.

Description

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

- Numbers, strings, and Boolean values are compared by value.
- Variables, objects, arrays, and functions are compared by reference.

Example

The following example illustrates the result of the != operator:

```
5 != 8 returns true
```

```
5 != 5 returns false
```

This example illustrates the use of the != operator in an if statement.

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

See also

!== (strict inequality), == (equality), === (strict equality)

!== (strict inequality)

Availability

Flash Player 6.

Usage

expression1 !== *expression2*

Description

Operator; tests for the exact opposite of the === operator. The strict inequality operator performs the same as the inequality operator except that data types are not converted. If *expression1* is equal to *expression2*, and their data types are equal, the result is *false*. As with the === operator, the definition of *equal* depends on the data types being compared.

- Numbers, strings, and Boolean values are compared by value.
- Variables, objects, arrays, and functions are compared by reference.

Example

The following code displays the returned value of operations that use the equality, strict equality, and strict inequality operators.

```
s1 = new String("5");
s2 = new String("5");
s3 = new String("Hello");
n  = new Number(5);
b  = new Boolean(true);
```

```
s1 == s2; // true
s1 == s3; // false
s1 == n;  // true
s1 == b;  // false
```

```
s1 === s2; // true
s1 === s3; // false
s1 === n;  // false
s1 === b;  // false
```

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

See also

!= (inequality), == (equality), === (strict equality)

% (modulo)

Availability

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

Usage

expression1 % *expression2*

Parameters

None.

Returns

Nothing.

Description

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

Example

The following is a numeric example that uses the modulo (%) operator.

```
trace (12 % 5);
// returns 2
trace (4.3 % 2.1);
// returns approximately 0.1
```

%= (modulo assignment)

Availability

Flash Player 4.

Usage

expression1 %= *expression2*

Parameters

None.

Returns

Nothing.

Description

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

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

Example

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

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

See also

% (modulo)

& (bitwise AND)

Availability

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

Usage

expression1 & *expression2*

Parameters

None.

Returns

Nothing.

Description

Operator (bitwise); converts *expression1* and *expression2* to 32-bit unsigned integers, and performs a Boolean AND operation on each bit of the integer parameters. The result is a new 32-bit unsigned integer.

&& (short-circuit AND)

Availability

Flash Player 4.

Usage

expression1 && *expression2*

Parameters

None.

Returns

Nothing.

Description

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

Example

This example uses the && operator to perform a test to determine if a player has won the game. The `turns` variable and the `score` variable are updated when a player takes a turn or scores points during the game. The following script displays “You Win the Game!” in the Output window when the player’s score reaches 75 or higher in 3 turns or less.

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

&= (bitwise AND assignment)

Availability

Flash Player 5.

Usage

expression1 &= *expression2*

Parameters

None.

Returns

Nothing.

Description

Operator (bitwise compound assignment); assigns *expression1* the value of *expression1* & *expression2*. For example, the following two expressions are the same.

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

Example

The following example assigns the value 9 to x.

```
x = 15;  
y = 9;  
trace(x &= y);  
// returns 9
```

See also

& (bitwise AND)

() (parentheses)

Availability

Flash Player 4.

Usage

```
(expression1, expression2);  
function(parameter1, ..., parameterN);
```

Parameters

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

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

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

Returns

Nothing.

Description

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

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

Usage 2: Surrounds one or more parameters and passes them as parameters to the function outside the parentheses.

Example

Usage 1: The following statements illustrate the use of parentheses to control the order in which expressions are executed. The value of each expression is displayed below each line, as follows:

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

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

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

Usage 2: The following examples illustrate the use of parentheses with functions.

```
getDate();
```

```
invoice(item, amount);
```

```
function traceParameter(param){  
    trace(param);  
}  
traceParameter(2*2);
```

See also

with

- (minus)

Availability

Flash Player 4.

Usage

(Negation) $-expression$

(Subtraction) $expression1 - expression2$

Parameters

None.

Returns

Nothing.

Description

Operator (arithmetic); used for negating or subtracting.

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

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

Example

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

$-(2 + 3)$

The result is -5 .

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

$5 - 2$

The result is 3, which is an integer.

Usage 2: The following statement subtracts the floating-point number 1.5 from the floating-point number 3.25.

$3.25 - 1.5$

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

* (multiplication)

Availability

Flash Player 4.

Usage

$expression1 * expression2$

Parameters

None.

Returns

Nothing.

Description

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

Example

The following statement multiplies the integers 2 and 3:

```
2 * 3
```

The result is 6, which is an integer.

Example

This statement multiplies the floating-point numbers 2.0 and 3.1416.

```
2.0 * 3.1416
```

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

***= (multiplication assignment)**

Availability

Flash Player 4.

Usage

```
expression1 *= expression2
```

Parameters

None.

Returns

Nothing.

Description

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

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

Example

The following example assigns the value 50 to the variable x.

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

Example

The second and third lines of the following example calculate the expressions on the right-hand side of the equals sign and assign the results to x and y.

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

See also

* (multiplication)

, (comma)

Availability

Flash Player 4.

Usage

expression1, expression2

Parameters

None.

Returns

Nothing.

Description

Operator; evaluates *expression1*, then *expression2*, and returns the value of *expression2*. This operator is primarily used with the `for` loop statement.

Example

The following code sample uses the comma operator:

```
var a=1, b=2, c=3;
```

This is equivalent to writing the following code:

```
var a=1;  
var b=2;  
var c=3;
```

. (dot)

Availability

Flash Player 4.

Usage

object.property_or_method

instancename.variable

instancename.childinstance.variable

Parameters

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

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

instancename The instance name of a movie clip.

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

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

Returns

Nothing.

Description

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

Example

The following statement identifies the current value of the variable `hairColor` in the movie clip `person`.

```
person.hairColor
```

This is equivalent to the following Flash 4 syntax:

```
/person:hairColor
```

Example

The following code illustrates how the dot operator can be used to create an array structure.

```
account.name = "Gary Smith";  
account.address = "123 Main St";  
account.city = "Any Town";  
account.state = "CA";  
account.zip = "12345";
```

See also

[] (array access)

?: (conditional)

Availability

Flash Player 4.

Usage

expression1 ? *expression2* : *expression3*

Parameters

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

expression2, *expression3* Values of any type.

Returns

Nothing.

Description

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

Example

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

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

/ (division)

Availability

Flash Player 4.

Usage

expression1 / *expression2*

Parameters

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

Returns

Nothing.

Description

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

Example

The following statement divides the floating-point number 22.0 by 7.0 and then displays the result in the Output window.

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

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

// (comment delimiter)

Availability

Flash 1.

Usage

// *comment*

Parameters

comment Any characters.

Returns

Nothing.

Description

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

Example

This script uses comment delimiters to identify the first, third, fifth, and seventh lines as comments.

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

See also

`/*` (comment delimiter)

`/*` (comment delimiter)

Availability

Flash Player 5.

Usage

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

Parameters

comment Any characters.

Returns

Nothing.

Description

Comment; indicates one or more lines of script comments. Any characters that appear between the opening comment tag `/*` and the closing comment tag `*/`, are interpreted as a comment and ignored by the ActionScript interpreter. Use the first type of syntax to identify single-line comments. Use the second type of syntax to identify comments on multiple successive lines. Leaving off the closing tag `*/` when using this form of comment delimiter returns an error message.

Example

This script uses comment delimiters at the beginning of the script.

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

ballX = ball._x;
ballY = ball._y;
batX = bat._x;
batY = bat._y;
```

See also

`//` (comment delimiter)

/= (division assignment)

Availability

Flash Player 4.

Usage

expression1 /= expression2

Parameters

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

Returns

Nothing.

Description

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

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

Example

The following code illustrates using the /= operator with variables and numbers.

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

[] (array access)

Availability

Flash Player 4.

Usage

```
myArray = [ "a0", a1, ...aN];
myMultiDimensionalArray = [[ "a0", ...aN], ...[ "a0", ...aN]]
myArray[E] = value
myMultiDimensionalArray[E][E] = value
object["value"];
```

Parameters

myArray The name of an array.

a0, a1, ...aN Elements in an array.

myMultiDimensionalArray The name of a simulated multidimensional array.

E The number (or index) of an element in an array.

object The name of an object.

value A string or an expression that evaluates to a string that names a property of the object.

Returns

Nothing.

Description

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

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

```
employee = [15, "Barbara", "Erick"];
```

Usage 2: You can nest brackets to simulate multidimensional arrays. The following code creates an array called `ticTacToe` with three elements; each element is also an array with three elements.

```
ticTacToe = [[1,2,3],[4,5,6],[7,8,9]];
```

```
// choose Debug > List Variables in test movie mode  
// to see a list of the array elements
```

Usage 3: Surround the index of each element with brackets to access it directly; you can add a new element to an array, change or retrieve the value of an existing element. The first element in an array is always 0:

```
myArray[0] = 15;  
myArray[1] = "Hello";  
myArray[2] = true;
```

You can use brackets to add a fourth element, as in the following:

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

Usage 4: You can use brackets to access an element in a multidimensional array. The first set of brackets identifies the element in the original array, and the second set identifies the element in the nested array. The following line of code sends the number 6 to the Output window.

```
ticTacToe = [[1,2,3],[4,5,6],[7,8,9]];  
trace(ticTacToe[1][2]);
```

```
// returns 6
```

Usage 5: You can use the array access operator instead of the `eval` function to dynamically set and retrieve values for movie clip names or any property of an object:

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

Example

Usage 1: The following code samples show two different ways of creating a new empty Array object; the first line uses brackets.

```
myArray =[];  
myArray = new Array();
```

Usage 1 and 2: The following example creates an array called `employee` and uses the `trace` action to send the elements to the Output window. In the fourth line, an element in the array is changed and the fifth line sends the newly modified array to the Output window:

```
employee=["Barbara", "George", "Mary"];
trace(employee);
// Barbara, George, Mary
employee[2]="Sam";
trace(employee);
// Barbara, George, Sam
```

Usage 3: In the following example, the expression inside the brackets ("`piece`" + `i`) is evaluated and the result is used as the name of the variable to be retrieved from the `mc` movie clip. In this example, the variable `i` must live on the same Timeline as the button. If the variable `i` is equal to 5, for example, the value of the variable `piece5` in the `mc` movie clip will be displayed in the Output window:

```
on(release){
    x = mc["piece"+i];
    trace(x);
}
```

Usage 3: In the following code, the expression inside the brackets is evaluated and the result is used as the name of the variable to be retrieved from movie clip `name`:

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

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

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

Usage 3: You can also use the array access operator on the left side of an assignment statement to dynamically set instance, variable, and object names:

```
name[index] = "Gary";
```

See also

Array (object), Object (object), eval

^(bitwise XOR)

Availability

Flash Player 5.

Usage

```
expression1 ^ expression2
```

Parameters

expression1, *expression2* A number.

Returns

None.

Description

Operator (bitwise); converts *expression1* and *expression2* to 32-bit unsigned integers, and returns a 1 in each bit position where the corresponding bits in *expression1* or *expression2*, but not both, are 1.

Example

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

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

^= (bitwise XOR assignment)

Availability

Flash Player 5.

Usage

expression1 ^= *expression2*

Parameters

expression1, *expression2* Integers and variables.

Returns

None.

Description

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

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

Example

The following is an example of a ^= operation.

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

See also

^(bitwise XOR)

{} (object initializer)

Availability

Flash Player 5.

Usage

```
object = {name1: value1, name2: value2,...nameN: valueN};
```

Parameters

object The object to create.

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

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

Returns

None.

Description

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

Example

The first line of the following code creates an empty object using the object initializer operator; the second line creates a new object using a constructor function.

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

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

```
account = { name: "Betty Skate",  
            address: "123 Main Street",  
            city: "Blossomville",  
            state: "California",  
            zip: "12345",  
            balance: "1000" };
```

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

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

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

```
person = new Person();  
person.name = 'Gina Vechio';  
person.children = new Array();  
person.children[0] = 'Ruby';  
person.children[1] = 'Chickie';  
person.children[2] = 'Puppa';
```

See also

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

| (bitwise OR)

Availability

Flash Player 5.

Usage

expression1 | *expression2*

Parameters

expression1, *expression2* A number.

Returns

None.

Description

Operator (bitwise); converts *expression1* and *expression2* to 32-bit unsigned integers, and returns a 1 in each bit position where the corresponding bits of either *expression1* or *expression2* are 1.

Example

The following is an example of a bitwise OR operation.

```
// 15 decimal = 1111 binary
x = 15;
// 9 decimal = 1001 binary
y = 9;
trace(x | y);
// 1111 | 0011 = 1111
// returns 15 decimal (= 1111 binary)
```

|| (logical OR)

Availability

Flash Player 4.

Usage

expression1 || *expression2*

Parameters

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

Returns

None.

Description

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

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

Example

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

```
x = 10
y = 250
start = false
if(x > 25 || y > 200 || start){
    trace('the logical OR test passed');
}
```

Example

This example demonstrates how a non-Boolean expression can produce an unexpected result. If the expression on the left converts to `true`, that result is returned without converting the expression on the right.

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

|= (bitwise OR assignment)

Availability

Flash Player 5.

Usage

expression1 |= *expression2*

Parameters

expression1, *expression2* A number or variable.

Returns

None.

Description

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

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

Example

The following example uses the `|=` operator:

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

See also

`|` (bitwise OR)

~ (bitwise NOT)

Availability

Flash Player 5.

Usage

`~ expression`

Parameters

expression A number.

Returns

None.

Description

Operator (bitwise); converts the *expression* to a 32-bit unsigned integer, then inverts the bits. A bitwise NOT operation changes the sign of a number and subtracts 1.

Example

The following example shows a bitwise NOT operation performed on a variable.

```
a = 0;
trace ("when a = 0, ~a = "+~a);
// when a = 0, ~a = -1
a = 1;
trace ("when a = 1, ~a = "+~a);
// when a = 0, ~a = -2
// therefore, ~0=-1 and ~1=-2
```

+ (addition)

Availability

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

Flash 4 file:

```
x + y
```

Converted Flash 5 file:

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

Usage

```
expression1 + expression2
```

Parameters

expression1, *expression2* A number or string.

Returns

None.

Description

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

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

Example

The following example concatenates two strings and displays the result in the Output window.

```
name = "Cola";  
instrument = "Drums";  
trace (name + " plays " + instrument);
```

Example

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

```
oldBalance = 1345.23;  
currentBalance = deposit + oldBalance;  
trace (currentBalance);
```

For example, if a user enters 475 in the `deposit` text field, the `trace` action sends the value 4751345.23 to the Output window.

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

```
currentBalance = Number(deposit) + oldBalance;
```


Example

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

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

This statement adds the floating-point numbers 2.5 and 3.25 and displays the result, 5.75, a floating-point number, in the Output window:

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

See also

[add](#)

+= (addition assignment)

Availability

Flash Player 4.

Usage

```
expression1 += expression2
```

Parameters

expression1, *expression2* A number or string.

Returns

Nothing.

Description

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

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

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

Example

The following example shows a numeric use of the += operator.

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

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

```
x = "My name is "  
x += "Gilbert"  
trace (x)
```

See also

[+ \(addition\)](#)

< (less than)

Availability

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

Flash 4 file:

```
x < y
```

Converted Flash 5 file:

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

Usage

```
expression1 < expression2
```

Parameters

expression1, *expression2* A number or string.

Description

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

Example

The following examples illustrate `true` and `false` returns for both numeric and string comparisons.

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

« (bitwise left shift)

Availability

Flash Player 5.

Usage

expression1 << *expression2*

Parameters

expression1 A number or expression to be shifted left.

expression2 A number or expression that converts to an integer from 0 to 31.

Returns

Nothing.

Description

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

Example

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

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

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

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

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

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

See also

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

<<= (bitwise left shift and assignment)

Availability

Flash Player 5.

Usage

expression1 <<= *expression2*

Parameters

expression1 A number or expression to be shifted left.

expression2 A number or expression that converts to an integer from 0 to 31.

Returns

Nothing.

Description

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

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

See also

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

<= (less than or equal to)

Availability

Flash Player 4.

Flash 4 file:

```
x <= y
```

Converted Flash 5 file:

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

Usage

expression1 <= *expression2*

Parameters

expression1, *expression2* A number or string.

Returns

Nothing.

Description

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

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

Example

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

```
5 <= 10;
// true

2 <= 2;
// true

10 <= 3;
// false

"Allen" <= "Jack";
// true

"Jack" <= "Allen";
// false

"11" <= "3";
//true

"11" <= 3;
// numeric comparison
// false

"C" <= "abc";
// false

"A" <= "a";
// true
```

⌵ (inequality)

Availability

Flash 2.

Usage

expression1 <> *expression2*

Parameters

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

Returns

Nothing.

Description

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

- Numbers, strings, and Boolean values are compared by value.
- Variables, objects, arrays, and functions are compared by reference.

This operator has been deprecated in Flash 5, and users are encouraged to use the != operator.

See also

!= (inequality)

= (assignment)

Availability

Flash Player 4.

Flash 4 file:

```
x = y
```

Converted Flash 5 file:

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

Usage

expression1 = *expression2*

Parameters

expression1 A variable, element of an array, or property of an object.

expression2 A value of any type.

Returns

Nothing.

Description

Operator; assigns the type of *expression2* (the parameter on the right) to the variable, array element, or property in *expression1*.

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

Example

The following example uses the assignment operator to assign the Number data type to the variable x.

```
x = 5
```

The following example uses the assignment operator to assign the String data type to the variable x.

```
x = "hello"
```

See also

== (equality)

-= (subtraction assignment)

Availability

Flash Player 4.

Usage

```
expression1 -= expression2
```

Parameters

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

Returns

Nothing.

Description

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

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

String expressions must be converted to numbers or NaN is returned.

Example

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

```
x = 5;  
y = 10;  
x -= y  
trace(x);  
//returns -5
```

Example

The following example shows how strings are converted to numbers.

```
x = "5";
y = "10";
x -= y;
trace(x);
// returns -5
```

== (equality)

Availability

Flash Player 5.

Usage

expression1 == *expression2*

Parameters

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

Returns

Nothing.

Description

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

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

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

Example

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

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


Example

These examples show the results of operations that compare mixed types.

```
x = "5"; y = "5";  
trace(x == y);  
// true
```

```
x = "5"; y = "66";  
trace(x ==y);  
// false
```

```
x = "chris"; y = "steve";  
trace (x == y);  
//false
```

See also

!= (inequality), == (strict equality), !== (strict inequality)

=== (strict equality)

Availability

Flash Player 6.

Usage

expression1 === *expression2*

Description

Operator; tests two expressions for equality; the strict equality operator performs just like the equality operator except that data types are not converted. The result is `true` if both expressions, including their data types, are equal.

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

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

Example

The following code displays the returned value of operations that use the equality, strict equality, and strict inequality operators.

```
s1 = new String("5");
s2 = new String("5");
s3 = new String("Hello");
n  = new Number(5);
b  = new Boolean(true);
```

```
s1 == s2; // true
s1 == s3; // false
s1 == n;  // true
s1 == b;  // false
```

```
s1 === s2; // true
s1 === s3; // false
s1 === n;  // false
s1 === b;  // false
```

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

See also

`==` (equality), `!=` (inequality), `===` (strict equality)

> (greater than)

Availability

Flash Player 5.

Usage

expression1 > *expression2*

Parameters

expression1, *expression2* An integer, floating-point number, or string.

Returns

Nothing.

Description

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

>= (greater than or equal to)

Availability

Flash Player 4.

Flash 4 file:

`x > y`

Converted Flash 5 file:

`Number(x) > Number(y)`

Usage

expression1 >= *expression2*

Parameters

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

Returns

Nothing.

Description

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

In Flash 5, greater than or equal to (>) is a comparison operator capable of handling various data types. In Flash 4, > is an numeric operator. Flash 4 files brought into the Flash 5 authoring environment undergo a conversion process to maintain data type integrity.

>> (bitwise right shift)

Availability

Flash Player 5.

Usage

expression1 >> *expression2*

Parameters

expression1 A number or expression to be shifted right.

expression2 A number or expression that converts to an integer from 0 to 31.

Returns

Nothing.

Description

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

Example

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

```
x = 65535 >> 8
```

The result of the above operation is as follows:

```
x = 255
```

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

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

```
x = -1 >> 1
```

The result of the above operation is as follows:

```
x = -1
```

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

See also

>>= (bitwise right shift and assignment)

>>= (bitwise right shift and assignment)

Availability

Flash Player 5.

Usage

expression1 >>=*expression2*

Parameters

expression1 A number or expression to be shifted left.

expression2 A number or expression that converts to an integer from 0 to 31.

Returns

Nothing.

Description

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

Example

The following two expressions are equivalent.

```
A >>= B
```

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

The following commented code uses the bitwise (>>=) operator . It is also an example of using all bitwise operators.

```
function convertToBinary(number){
    var result = "";
    for (var i=0; i<32; i++) {
        // Extract least significant bit using bitwise AND
        var lsb = number & 1;
        // Add this bit to our result string
        result = (lsb ? "1" : "0") + result;
        // Shift number right by one bit, to see next bit
        number >>= 1;}
    return result;
}
trace(convertToBinary(479));
// Returns the string 00000000000000000000000011101111
// The above string is the binary representation of the decimal
// number 479
```

See also

<< (bitwise left shift)

>>> (bitwise unsigned right shift)

Availability

Flash Player 5.

Usage

```
expression1 >>> expression2
```

Parameters

expression1 A number or expression to be shifted right.

expression2 A number or expression that converts to an integer between 0 and 31.

Returns

Nothing.

Description

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

Example

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

```
x = -1 >>> 1
```

The result of the above operation is as follows:

```
x = 2147483647
```

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

See also

>>= (bitwise right shift and assignment)

>>= (bitwise unsigned right shift and assignment)

Availability

Flash Player 5.

Usage

```
expression1 >>= expression2
```

Parameters

expression1 A number or expression to be shifted left.

expression2 A number or expression that converts to an integer from 0 to 31.

Returns

Nothing.

Description

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

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

See also

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

Accessibility (object)

The Accessibility object is a collection of methods that you can use to create accessible content with ActionScript. In Flash MX, there is only one method.

This object is available in Flash Player 6.

Method summary for the Arguments object

Property	Description
Accessibility.isActive	Indicates whether a screen reader program is active.

Accessibility.isActive

Availability

Flash Player 6.

Usage

Accessibility.isActive()

Parameters

None.

Returns

A Boolean value.

Description

Method; indicates whether a screen reader program is currently active or not. Use this method when you want your movie to behave differently in the presence of a screen reader.

See also

System.capabilities.hasAccessibility

add

Availability

Flash Player 4.

Usage

string1 add *string2*

Parameters

string1, *string2* A string.

Returns

Nothing.

Description

Operator; concatenates (combines) two or more strings. The `add` operator replaces the Flash 4 `&` operator; Flash 4 files using the `&` operator are automatically converted to use the `add` operator for string concatenation when brought into the Flash 5 authoring environment. However, the `add` operator is deprecated in Flash 5, and use of the `+` operator is recommended when creating content for Flash Player 5 or Flash Player 6. Use the `add` operator to concatenate strings if you are creating content for Flash 4 or earlier versions of the Player.

See also

`+` (addition)

and

Availability

Flash Player 4.

Usage

condition1 and *condition2*

Parameters

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

Returns

Nothing.

Description

Operator; performs a logical AND operation in Flash Player 4. If both expressions evaluate to true, then the entire expression is true. This operator has been deprecated in Flash 5, and users are encouraged to make use of the `&&` operator.

See also

`&&` (short-circuit AND)

arguments (object)

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

The arguments object is available in Flash Player 6.

Property summary for the Arguments object

Property	Description
<code>arguments.callee</code>	Refers to the function being called.
<code>arguments.caller</code>	Refers to the calling function.
<code>arguments.length</code>	The number of parameters passed to a function.

arguments.callee

Availability

Flash Player 5.

Usage

`arguments.callee`

Description

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

Example

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

```
factorial = function (x) {  
    if (x <= 1) {  
        return 1;  
    } else {  
        return x * arguments.callee(x-1);  
    }  
};
```

The following is a named recursive function:

```
function factorial (x) {  
    if (x <= 1) {  
        return 1;  
    } else {  
        return x * factorial(x-1);  
    }  
}
```

arguments.caller

Availability

Flash Player 6.

Usage

`arguments.caller`

Description

Property; refers to the Arguments object of the calling function.

arguments.length

Availability

Flash Player 6.

Usage

`arguments.length`

Description

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

Array (object)

The Array object lets you access and manipulate arrays. An array is an object whose properties are identified by a number representing their position in the array. This number is referred to as the *index*. All arrays are zero-based, which means that the first element in the array is [0], the second element is [1], and so on. In the following example, `myArray` contains the months of the year.

```
myArray[0] = "January"  
myArray[1] = "February"  
myArray[2] = "March"  
myArray[3] = "April"
```

To create an Array object, use the constructor `new Array` or the array access operator (`[]`). To access the elements of an array, use the array access operator (`[]`).

In Flash MX, the Array object has become a native object. As such, you will experience dramatic improvement in performance.

Method summary for the Array object

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

Property summary for the Array object

Property	Description
<code>Array.length</code>	Returns the length of the array.

Constructor for the Array object

Availability

Flash Player 5.

Usage

```
new Array()
```

```
new Array(length)
```

```
new Array(element0, element1, element2,...elementN)
```

Parameters

length An integer specifying the number of elements in the array. In the case of noncontiguous elements, the *length* parameter specifies the index number of the last element in the array plus 1.

element0...elementN A list of two or more arbitrary values. The values can be numbers, strings, objects, or other arrays. The first element in an array always has an index or position of 0.

Returns

Nothing.

Description

Constructor; lets you create an array. You can use the constructor to create different types of arrays: an empty array, an array with a specific length but whose elements have no values, or an array whose elements have specific values.

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

Usage 2: If you specify only a length, an array is created with *length* number of elements with no values.

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

Example

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

```
myArray = new Array();
```

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

```
go_gos = new Array("Belinda", "Gina", "Kathy", "Charlotte", "Jane");  
trace(go_gos.join(" + "));
```

The initial elements of the `go_gos` array are identified as follows:

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

The following code adds a fifth element to the `go_gos` array and changes the first element:

```
go_gos[5] = "Donna";  
go_gos[1] = "Nina";  
trace(go_gos.join(" + "));
```

See also

`Array.length`, `[]` (array access)

Array.concat

Availability

Flash Player 5.

Usage

```
myArray.concat(value0,value1,...valueN)
```

Parameters

value0,...*valueN* Numbers, elements, or strings to be concatenated in a new array.

Returns

Nothing.

Description

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

Example

The following code concatenates two arrays.

```
alpha = new Array("a","b","c");
numeric = new Array(1,2,3);
alphaNumeric=alpha.concat(numeric);
trace(alphaNumeric);
// creates array ["a","b","c",1,2,3]
```

The following code concatenates three arrays.

```
num1=[1,3,5];
num2=[2,4,6];
num3=[7,8,9];
nums=num1.concat(num2,num3)
trace(nums);
// creates array [1,3,5,2,4,6,7,8,9]
```

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

```
a = new Array ("a","b","c");
n = new Array(1, [2, 3], 4);
// 2 and 3 are elements in a nested array
x = a.concat(n);
x[0] = "a"
x[1] = "b"
x[2] = "c"
x[3] = 1
x[4] = 2, 3
x[5] = 4
```

Array.join

Availability

Flash Player 5.

Usage

```
myArray.join([separator])
```

Parameters

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

Returns

Nothing.

Description

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

Example

The following example creates an array, with three elements. It then joins the array three times—using the default separator, a comma and a space, and a plus sign—and displays them in the Output window:

```
a = new Array("Earth","Moon","Sun")
trace(a.join());
// returns Earth, Moon, Sun
trace(a.join(" - "));
// returns Earth - Moon - Sun
trace(a.join(" + "));
// returns Earth + Moon + Sun
```

Array.length

Availability

Flash Player 5.

Usage

myArray.length

Description

Property; contains the length of the array. This property is automatically updated when new elements are added to the array. When you assign a value to an array element (for example, *myArray[index] = value*), if *index* is a number, and *index+1* is a greater than the length property, the length property is updated to *index + 1*.

Example

The following code explains how the length property is updated.

```
// initial length is 0
myArray = new Array();
myArray[0] = 'a';
// myArray.length is updated to 1
myArray[1] = 'b';
// myArray.length is updated to 2
myArray[9] = 'c';
// myArray.length is updated to 10
```

Array.pop

Availability

Flash Player 5.

Usage

myArray.pop()

Parameters

None.

Returns

Nothing.

Description

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

Example

The following code creates the `myPets` array containing four elements, then removes its last element.

```
myPets = ["cat", "dog", "bird", "fish"];
popped = myPets.pop();
trace(popped);
// returns fish
```

Array.push

Availability

Flash Player 5.

Usage

myArray.push(*value*,...)

Parameters

value One or more values to append to the array.

Returns

The length of the new array.

Description

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

Example

The following example creates the array `myPets` with two elements, `cat` and `dog`. The second line adds two elements to the array. After the `push` method is called, the variable `pushed` contains four elements. Because the `push` method returns the new length of the array, the `trace` action in the last line sends the new length of `myPets` (4) to the Output window:

```
myPets = ["cat", "dog"];
pushed = myPets.push("bird", "fish");
trace(pushes);
```

Array.reverse

Availability

Flash Player 5.

Usage

```
myArray.reverse()
```

Parameters

None.

Returns

Nothing.

Description

Method; reverses the array in place.

Example

The following is an example of using the `Array.reverse` method.

```
var numbers = [1, 2, 3, 4, 5, 6];  
trace(numbers.join());  
numbers.reverse();  
trace(numbers.join());
```

Output:

```
1,2,3,4,5,6  
6,5,4,3,2,1
```

Array.shift

Availability

Flash Player 5.

Usage

```
myArray.shift()
```

Parameters

None.

Returns

The first element in an array.

Description

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

Example

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

```
myPets = ["cat", "dog", "bird", "fish"];  
shifted = myPets.shift();  
trace(shifted);  
// returns cat
```

See also

`Array.pop`

Array.slice

Availability

Flash Player 5.

Usage

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

Parameters

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

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

Returns

Nothing.

Description

Method; extracts a slice or a substring of the array and returns it as a new array without modifying the original array. The returned array includes the *start* element and all elements up to, but not including, the *end* element.

Array.sort

Availability

Flash Player 5.

Usage

```
myArray.sort([compareFunction])
```

Parameters

compareFunction An optional comparison function used to determine the sorting order of elements in an array. Given the elements A and B, the *orderfunc* parameter can have one of the following three values:

- -1 if A appears before B in the sorted sequence
- 0 if A = B
- 1 if A appears after B in the sorted sequence

Returns

Nothing.

Description

Method; sorts the array in place, without making a copy. If you omit the *compareFunction* parameter, Flash sorts the elements in place using the < comparison operator.

Example

The following example uses `Array.sort` without specifying the *compareFunction* parameter:

```
var fruits = ["oranges", "apples", "strawberries", "pineapples", "cherries"];
trace(fruits.join());
fruits.sort();
trace(fruits.join());
```

Output:

oranges,apples,strawberries,pineapples,cherries
displays apples,cherries,oranges,pineapples,strawberries

The following example uses `Array.sort` with a specified order function.

```
var passwords = [
    "gary:foo",
    "mike:bar",
    "john:snafu",
    "steve:yuck",
    "daniel:1234"
];
function order (a, b) {
    // Entries to be sorted are in form
    // name:password
    // Sort using only the name part of the
    // entry as a key.
    var name1 = a.split(':')[0];
    var name2 = b.split(':')[0];
    if (name1 < name2) {
        return -1;
    } else if (name1 > name2) {
        return 1;
    } else {
        return 0;
    }
}
for (var i=0; i< password.length; i++) {
    trace (passwords.join());
}
passwords.sort(order);
trace ("Sorted:");
for (var i=0; i< password.length; i++) {
    trace (passwords.join());
}
```

Running the previous code displays the following result in the Output window.

```
daniel:1234
gary:foo
john:snafu
mike:bar
steve:yuck
```

Array.sortOn

Availability

Flash Player 6.

Usage

`Array.sortOn(fieldName)`

Parameters

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

Returns

None.

Description

Method; sorts the elements in an array based on a field in the array. If no *fieldName* parameter is passed, the function fails. If multiple *fieldName* parameters are passed, the first field is converted to a string value and the remaining parameters are ignored.

If either of the elements being compared does not contain the field specified in the *fieldName* parameter, the sort defaults to the behavior in the `Array.sort` method.

Example

This following example creates a new array and sorts it based on the field `city`:

```
var recArray = new Array();
recArray.push( { name: "bob", city: "omaha", zip: 68144 } );
recArray.push( { name: "greg", city: "kansas city", zip: 72345 } );
recArray.push( { name: "chris", city: "burlingame", zip: 94010 } );
recArray.sortOn("city");
// results in the following:
recArray[0] = name: "chris", city: "burlingame", zip: 94010
recArray[1] = name: "greg", city: "kansas city", zip: 72345
recArray[2] = name: "bob", city: "omaha", zip: 68144
```

See also

`Array.sort`

Array.splice

Availability

Flash Player 5.

Usage

```
myArray.splice(start, deleteCount, value0,value1...valueN)
```

Parameters

start The index of the element in the array where the insertion or deletion begins.

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

value Zero or more values to insert into the array at the insertion point specified in the *start* parameter. This parameter is optional.

Returns

Nothing.

Description

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

Array.toString

Availability

Flash Player 5.

Usage

```
myArray.toString()
```

Parameters

None.

Returns

A string.

Description

Method; returns a string value representing the elements in the specified Array object. Every element in the array, starting with index 0 and ending with index *myArray.length-1*, is converted to a concatenated string and separated by commas.

Example

The following example creates *myArray*, converts it to a string, and displays 1,2,3,4,5 in the Output window.

```
myArray = new Array();  
myArray[0] = 1;  
myArray[1] = 2;  
myArray[2] = 3;  
myArray[3] = 4;  
myArray[4] = 5;  
  
trace(myArray.toString());
```

Array.unshift

Availability

Flash Player 5.

Usage

```
myArray.unshift(value1, value2, ... valueN)
```

Parameters

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

Returns

The new length of the array.

Description

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

asfunction

Availability

Flash Player 5.

Usage

`asfunction:function, "parameter"`

Parameters

function An identifier for a function.

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

Returns

Nothing.

Description

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

Example

In this example, the `MyFunc` function is defined in the first three lines of code. The `textField` variable is associated with an HTML text field. The text "Click Me!" is a hyperlink inside the text field. The `MyFunc` function is called when the user clicks on the hyperlink:

```
function MyFunc(arg){
    trace ("You clicked me!Argument was "+arg);
}
myTextField.text ="<A HREF=\"asfunction:MyFunc,Foo \">Click Me!</A>";
```

When the hyperlink is clicked, the following results are displayed in the Output window:

```
You clicked me! Parameter was Foo
```

Boolean (function)

Availability

Flash Player 5.

Usage

`Boolean(expression)`

Parameters

expression An expression to convert to a Boolean value.

Returns

Nothing.

Description

Function; converts the parameter *expression* to a Boolean value and returns a value as follows:

- If *expression* is a Boolean value, the return value is *expression*.
- If *expression* is a number, the return value is `true` if the number is not zero, otherwise the return value is `false`.
- If *expression* is a string, the `toNumber` method is called and the return value is `true` if the number is not zero, otherwise the return value is `false`.
- If *expression* is undefined, the return value is `false`.
- If *expression* is a movie clip or an object, the return value is `true`.

Boolean (object)

The Boolean object is a wrapper object with the same functionality as the standard JavaScript Boolean object. Use the Boolean object to retrieve the primitive data type or string representation of a Boolean object. In Flash MX, the Boolean object has become a native object. As such, you will experience dramatic improvement in performance.

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

Method summary for the Boolean object

Method	Description
<code>Boolean.toString</code>	Returns the string representation (<code>true</code>) or (<code>false</code>) of the Boolean object.
<code>Boolean.valueOf</code>	Returns the primitive value type of the specified Boolean object.

Constructor for the Boolean object

Availability

Flash Player 5.

Usage

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

Parameters

x Any expression. This parameter is optional.

Returns

Nothing.

Description

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

Note: To maintain compatibility with Flash Player 4, the handling of strings by the Boolean object is not ECMA-262 standard.

Example

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

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

Boolean.toString

Availability

Flash Player 5.

Usage

```
myBoolean.toString()
```

Parameters

None.

Returns

Nothing.

Description

Method; returns the string representation, `true` or `false` of the Boolean object.

Boolean.valueOf

Availability

Flash Player 5.

Usage

```
Boolean.valueOf()
```

Parameters

None.

Returns

Nothing.

Description

Method; returns the primitive value type of the specified Boolean object.

break

Availability

Flash Player 4.

Usage

```
break
```

Parameters

None.

Returns

Nothing.

Description

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

Example

The following example uses the `break` action to exit an otherwise infinite loop.

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

See also

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

Button (object)

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

The Button object inherits from the Object object.

The Button object is supported by Flash Player 6.

Method summary for the Button object

Method	Description
<code>Button.getDepth</code>	Returns the depth of a button instance.

Property summary for the Button object

Property	Description
<code>Button._alpha</code>	The transparency value of a button instance.
<code>Button.enabled</code>	Indicates whether a button is active.
<code>Button._focusrect</code>	Indicates whether a button with focus has a yellow rectangle around it.
<code>Button._height</code>	The height of a button instance, in pixels.
<code>Button._highquality</code>	Indicates the rendering quality of the movie
<code>Button._name</code>	The instance name of a button instance.
<code>Button._parent</code>	A reference to the movie clip instance that is the parent of this instance.
<code>Button._quality</code>	Indicates the rendering quality of the movie.
<code>Button._rotation</code>	The degree of rotation of a button instance.

Property	Description
<code>Button._soundbuftime</code>	Number of seconds for a sound to preload.
<code>Button.tabEnabled</code>	Indicates whether a button is included in automatic tab ordering.
<code>Button.tabIndex</code>	Indicates the tab order of an object.
<code>Button._target</code>	The target path of a button instance.
<code>Button.trackAsMenu</code>	Indicates whether other buttons can receive mouse release events.
<code>Button._url</code>	The URL of the SWF file that created the button instance.
<code>Button.useHandCursor</code>	Indicates whether the hand cursor is displayed when the mouse passes over a button.
<code>Button._visible</code>	A Boolean value that determines whether a button instance is hidden or visible.
<code>Button._width</code>	The width of a button instance, in pixels.
<code>Button._x</code>	The x coordinate of a button instance.
<code>Button._xmouse</code>	The x coordinate of the cursor relative to a button instance.
<code>Button._xscale</code>	The value specifying the percentage for horizontally scaling a button instance.
<code>Button._y</code>	The y coordinate of a button instance.
<code>Button._ymouse</code>	The y coordinate of the cursor relative to a button instance.
<code>Button._yscale</code>	The value specifying the percentage for vertically scaling a button instance.

Event summary for the Button object

The following table lists the event summaries for the Button object.

Method	Description
<code>Button.onDragOut</code>	Invoked while the pointer is outside the button, the mouse button is pressed and then rolls outside the button area.
<code>Button.onDragOver</code>	Invoked while the pointer is over the button, the mouse button has been pressed then rolled outside the button, and then rolled back over the button.
<code>Button.onKeyUp</code>	Invoked when a key is released.
<code>Button.onKillFocus</code>	Invoked when focus is removed from a button.
<code>Button.onPress</code>	Invoked when the mouse is pressed while the pointer is over a button.
<code>Button.onRelease</code>	Invoked when the mouse is released while the pointer is over a button.
<code>Button.onReleaseOutside</code>	Invoked when the mouse is released while the pointer is outside the button after the button is pressed while the pointer is inside the button.
<code>Button.onRollOut</code>	Invoked when the pointer rolls outside of a button area.
<code>Button.onRollOver</code>	Invoked when the mouse pointer rolls over a button.
<code>Button.onSetFocus</code>	Invoked when a button has input focus and a key is released.

Button._alpha

Availability

Flash Player 6.

Usage

`myButton._alpha`

Description

Property; sets or retrieves the alpha transparency (*value*) of the button specified by *Button*. Valid values are 0 (fully transparent) to 100 (fully opaque). Objects in a button with `_alpha` set to 0 are active, even though they are invisible.

Example

The following sets the `_alpha` property of a button named `star` to 30%.

```
on(release) {  
    star._alpha = 30;  
}
```

Button.enabled

Availability

Flash Player 6.

Usage

`myButton.enabled`

Description

Property; a Boolean value that specifies whether a button is enabled. The default value is `true`.

Button._focusrect

Availability

Flash Player 6.

Usage

myButton._focusrect

Description

Property; a Boolean value that specifies whether a button has a yellow rectangle around it when it has keyboard focus. This property can override the global `_focusrect` property.

Button.getDepth

Availability

Flash Player 6.

Usage

myButton.getDepth()

Returns

An integer.

Description

Method; returns the depth of a button instance.

Button._height

Availability

Flash Player 6.

Usage

myButton._height

Description

Property; sets and retrieves the height of the button in pixels.

Example

The following code example sets the height and width of a button when the user clicks the mouse:

```
myButton._width = 200;  
myButton._height = 200;
```

Button._highquality

Availability

Flash Player 6.

Usage

myButton._highquality

Description

Property (global); specifies the level of anti-aliasing applied to the current movie. Specify 2 (BEST) to apply high quality with bitmap smoothing always on. Specify 1 (high quality) to apply anti-aliasing; this will smooth bitmaps if the movie does not contain animation. Specify 0 (low quality) to prevent anti-aliasing.

Example

```
_highquality = 1;
```

See also

_quality, *toggleHighQuality*

Button._name

Availability

Flash Player 6.

Usage

myButton._name

Description

Property; returns the instance name of the button specified by *myButton*.

Button.onDragOut

Availability

Flash Player 6.

Usage

myButton.onDragOut

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the mouse button is pressed over the button and the pointer then rolls outside the button.

Button.onDragOver

Availability

Flash Player 6.

Usage

myButton.onDragOver

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the user presses and drags the mouse button outside and then over the button.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onKeyDown` method that sends a trace to the Output window:

```
myButton.onDragOver = function () {  
    trace ("onDragOver called");  
};
```

See also

`Button.onKeyUp`

Button.onKeyDown

Availability

Flash Player 6.

Usage

myButton.onKeyDown

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when a button has keyboard focus and a key is pressed. The `onKeyDown` event is invoked with no parameters. You can use the `Key.getAscii` and `Key.getCode` methods to determine which key was pressed.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onKeyDown` method that sends a trace to the Output window:

```
myButton.onKeyDown = function () {  
    trace ("onKeyDown called");  
};
```

See also

`Button.onKeyUp`

Button.onKeyUp

Availability

Flash Player 6.

Usage

myButton.onKeyUp

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when a button has input focus and a key is released. The `onKeyUp` event is invoked with no parameters. You can use the `Key.getAscii` and `Key.getCode` methods to determine which key was pressed.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onKeyPress` method that sends a trace to the Output window:

```
myButton.onKeyUp = function () {  
    trace ("onKeyUp called");  
};
```

Button.onKillFocus

Availability

Flash Player 6.

Usage

```
myButton.onKillFocus = function (newFocus) {  
    statements;  
};
```

Parameters

newFocus The object that is receiving the focus.

Returns

Nothing.

Description

Event handler; an event that is invoked when a button loses keyboard focus. The `onKillFocus` method receives one parameter, *newFocus*, which is an object representing the new object receiving the focus. If no object receives the focus, *newFocus* contains the value `null`.

Button.onPress

Availability

Flash Player 6.

Usage

myButton.onPress

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when a button is pressed. You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onPress` method that sends a `trace` action to the Output window.

```
myButton.onPress = function () {  
    trace ("onPress called");  
};
```

Button.onRelease

Availability

Flash Player 6.

Usage

myButton.onRelease

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when a button is released.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onRelease` method that sends a `trace` action to the Output window.

```
myButton.onRelease = function () {  
    trace ("onRelease called");  
};
```

Button.onReleaseOutside

Availability

Flash Player 6.

Usage

myButton.onReleaseOutside

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the mouse is released while the pointer is outside the button after the button is pressed while the pointer is inside the button.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onReleaseOutside` method that sends a trace to the Output window.

```
myButton.onReleaseOutside = function () {  
    trace ("onReleaseOutside called");  
};
```

Button.onRollOut

Availability

Flash Player 6.

Usage

myButton.onRollOut

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the pointer rolls outside of a button area.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onRollOut` method that sends a trace to the Output window.

```
myButton.onRollOut = function () {  
    trace ("onRollOut called");  
};
```


Button.onRollOver

Availability

Flash Player 6.

Usage

```
myButton.onRollOver
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the pointer rolls over a button area.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onRollOver` method that sends a `trace` action to the Output window.

```
myButton.onRollOver = function () {  
    trace ("onRollOver called");  
};
```

Button.onSetFocus

Availability

Flash Player 6.

Usage

```
myButton.onSetFocus = function(oldFocus) {  
    statements;  
};
```

Parameters

oldFocus The object to lose keyboard focus.

Returns

Nothing.

Description

Event handler; invoked when a button receives keyboard focus. The *oldFocus* parameter is the object that loses the focus. For example, if the user presses the Tab key to move the input focus from a text field to a button, *oldFocus* contains the text field instance.

If there is no previously focused object, *oldFocus* contains a `null` value.

Button._parent

Availability

Flash Player 6.

Usage

_parent.property

Description

Property; specifies or returns a reference to the movie clip or object that contains the current movie clip or object. The current object is the one containing the ActionScript code that references `_parent`. Use `_parent` to specify a relative path to movie clips or objects that are above the current movie clip or object. You can use `_parent` to climb up multiple levels in the display list as in the following:

```
_parent._parent._alpha = 20;
```

See also

`_root`, `targetPath`

Button._quality

Availability

Flash Player 6.

Usage

myButton._quality

Description

Property (global); sets or retrieves the rendering quality used for a movie. Device fonts are always aliased and therefore are unaffected by the `_quality` property.

The `_quality` property can be set to the following values:

- "LOW" Low rendering quality. Graphics are not anti-aliased, bitmaps are not smoothed.
- "MEDIUM" Medium rendering quality. Graphics are anti-aliased using a 2 x 2 pixel grid, but bitmaps are not smoothed. Suitable for movies that do not contain text.
- "HIGH" High rendering quality. Graphics are anti-aliased using a 4 x 4 pixel grid, and bitmaps are smoothed if the movie is static. This is the default rendering quality setting used by Flash.
- "BEST" Very high rendering quality. Graphics are anti-aliased using a 4 x 4 pixel grid, and bitmaps are always smoothed.

Example

The following example sets the rendering quality to LOW:

```
myButton._quality = "LOW";
```

See also

`_highquality`, `toggleHighQuality`

Button._rotation

Availability

Flash Player 6.

Usage

myButton._rotation

Description

Property; specifies the rotation of the button in degrees.

Button._soundbuftime

Availability

Flash Player 6.

Usage

myButton._soundbuftime

Description

Property (global); an integer that specifies the number of seconds a sound prebuffers before it starts to stream.

Button.tabEnabled

Availability

Flash Player 6.

Usage

myButton.tabEnabled

Description

Property; may be set on an instance of the MovieClip, Button, or TextField objects. It is undefined by default.

If the `tabEnabled` property is undefined or `true`, then the object is included in automatic tab ordering. The object is included in custom tab ordering if the `tabIndex` property is also set to a value. If `tabEnabled` is `false`, then the object is not included in automatic tab ordering. For a movie clip, if `tabEnabled` is `false`, the movie clip's children may still be included in automatic tab ordering, unless the `tabChildren` property is also set to `false`.

If `tabEnabled` is undefined or `true`, and if the `tabIndex` property is set, then the object is included in custom tab ordering. If `tabEnabled` is `false`, then the object is not included in custom tab ordering, even if the `tabIndex` property is set. If `tabEnabled` is set to `false` in a movie clip, the movie clip's children can still be included in custom tab ordering.

See also

`Button.tabIndex`

Button.tabIndex

Availability

Flash Player 6.

Usage

myButton.tabIndex

Description

Property; lets you customize the tab ordering of objects in a movie. You can set the `tabIndex` property on a button, movie clip, or text field instance; it is undefined by default.

If any currently displayed object in the Flash movie contains a `tabIndex` property, automatic tab ordering is disabled, and the tab ordering is calculated from the `tabIndex` properties of objects in the movie. The custom tab ordering only includes objects that have `tabIndex` properties.

The `tabIndex` property may be an non-negative integer. The objects are ordered according to their `tabIndex` properties, in ascending order. An object with a `tabIndex` of 1 precedes an object with `tabIndex` 2. If two objects have the same `tabIndex`, the one that precedes the other in the tab ordering is undefined.

The custom tab ordering defined by the `tabIndex` property is *flat*. This means that no attention is paid to the hierarchical relationships of objects in the Flash movie. All objects in the Flash movie with `tabIndex` properties are placed in the tab order, and the tab order is determined by the order of the `tabIndex` values. If two objects have the same `tabIndex` value, the one that goes first is undefined. You shouldn't use the same `tabIndex` value for multiple objects.

Button._target

Availability

Flash Player 6.

Usage

myButton._target

Description

Property (read-only); returns the target path of the button instance specified in the *Button* parameter.

Button.trackAsMenu

Availability

Flash Player 6.

Usage

myButton.trackAsMenu

Description

Property; a Boolean property that indicates whether or not other buttons or movie clips can receive mouse release events. This allows you to create menus. You can set the `trackAsMenu` property on any button or movie clip object. If the `trackAsMenu` property does not exist, the default behavior is `false`.

You can change the `trackAsMenu` property at any time; the modified button immediately takes on the new behavior.

See also

`MovieClip.trackAsMenu`

Button._url

Availability

Flash Player 6.

Usage

myButton._url

Description

Property (read only); retrieves the URL of the SWF file that created the button.

Button.useHandCursor

Availability

Flash Player 6.

Usage

myButton.useHandCursor

Description

Property; a Boolean value that, when set to `true`, indicates whether a hand cursor is displayed when a user rolls over a button. The default value of `useHandCursor` is `true`. If the `useHandCursor` property is set to `false`, the arrow cursor is used instead.

You can change the `useHandCursor` property at any time; the modified button immediately takes on the new cursor behavior. The `useHandCursor` property can be read out of a prototype object.

Button._visible

Availability

Flash Player 6.

Usage

myButton._visible

Description

Property; a Boolean value that indicates whether the button specified by the *Button* parameter is visible. Buttons that are not visible (`_visible` property set to `false`) are disabled.

Button._width

Availability

Flash Player 6.

Usage

myButton._width

Description

Property; sets and retrieves the width of the button, in pixels.

Example

The following example sets the height and width properties of a button.

```
myButton._width=200;  
myButton._height=200;
```

See also

MovieClip._width

Button._x

Availability

Flash Player 6.

Usage

myButton._x

Description

Property; an integer that sets the *x* coordinate of a button relative to the local coordinates of the parent movie clip. If a button is on the main Timeline, then its coordinate system refers to the upper left corner of the Stage as (0, 0). If the button is inside a movie clip that has transformations, the button is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90° counterclockwise, the enclosed button inherits a coordinate system that is rotated 90° counterclockwise. The button's coordinates refer to the registration point position.

See also

Button._xscale, Button._y, Button._yscale

Button._xmouse

Availability

Flash Player 6.

Usage

myButton._xmouse

Description

Property (read-only); returns the *x* coordinate of the mouse position relative to the button.

See also

Button._ymouse

Button._xscale

Availability

Flash Player 6.

Usage

myButton._xscale

Description

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

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

See also

`Button._x`, `Button._y`, `Button._yscale`

Button._y

Availability

Flash Player 6.

Usage

myButton._y

Description

Property; sets the *y* coordinate of button relative to the local coordinates of the parent movie clip. If a button is in the main Timeline, its coordinate system refers to the upper left corner of the Stage as (0, 0). If the button is inside another movie clip that has transformations, the button is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90° counterclockwise, the enclosed button inherits a coordinate system that is rotated 90° counterclockwise. The button's coordinates refer to the registration point position.

See also

`Button._x`, `Button._xscale`, `Button._yscale`

Button._ymouse

Availability

Flash Player 6.

Usage

myButton._ymouse

Description

Property (read-only); indicates the *y* coordinate of the mouse position relative to the button.

See also

`Button._xmouse`

Button._yscale

Availability

Flash Player 6.

Usage

myButton._yscale

Description

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

See also

Button._y, Button._x, Button._xscale

call

Availability

Flash Player 4. This action is deprecated in Flash 5, and it is recommended that you use the `function` action instead.

Usage

call(frame)

Parameters

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

Returns

Nothing.

Description

Action; executes the script in the called frame without moving the playhead to that frame. Local variables will not exist once the script is finished executing.

See also

`function`

call function

Availability

Flash Player 6

Usage

object.function([parameters])

Parameters

object An object (could be a movie clip) in which the function was defined.

function An identifier that specifies a user-defined function.

parameters An optional parameter indicating any parameters that the function requires.

Returns

Nothing.

Description

Action; allows you to use parameter fields to call a user-defined function in normal mode in the Actions panel.

case

Availability

Flash Player 4.

Usage

case expression: statements

Parameters

expression Any expression.

statements Any statements.

Returns

Nothing.

Description

Keyword; defines a condition for the `switch` action. The statements in the *statements* parameter execute if the *expression* parameter that follows the `case` keyword equals the *expression* parameter of the `switch` action using strict equality (`===`)

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

See also

`switch`, `default`, `break`, `===` (strict equality)

chr

Availability

Flash Player 4. This function has been deprecated in Flash 5 in favor of the `String.fromCharCode` method.

Usage

chr(number)

Parameters

number An ASCII code number.

Returns

Nothing.

Description

String function; converts ASCII code numbers to characters.

Example

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

```
myVar = chr(65);
```

See also

`String.fromCharCode`

clearInterval

Availability

Flash Player 6.

Usage

```
clearInterval( intervalID )
```

Parameters

intervalID An object returned from a call to the `setInterval` function.

Returns

Nothing.

Description

Action; clears a call to the `setInterval` function.

Example

The following example first sets and then clears an interval call:

```
function callback() {  
    trace("interval called");  
}  
var intervalID;  
intervalID = setInterval( callback, 1000 );  
  
// sometime later  
clearInterval( intervalID );
```

See also

`setInterval`

Color (object)

The `Color` object lets you set the RGB color value and color transform of movie clips and retrieve those values once they have been set.

You must use the constructor `new Color()` to create an instance of the `Color` object before calling its methods.

The `Color` object is supported by Flash 5 and later versions of the Flash Player.

Method summary for the Color object

Method	Description
<code>Color.getRGB</code>	Returns the numeric RGB value set by the last <code>setRGB</code> call.
<code>Color.getTransform</code>	Returns the transform information set by the last <code>setTransform</code> call.
<code>Color.setRGB</code>	Sets the hexadecimal representation of the RGB value for a <code>Color</code> object.
<code>Color.setTransform</code>	Sets the color transform for a <code>Color</code> object.

Constructor for the Color object

Availability

Flash Player 5.

Usage

```
new Color(target)
```

Parameters

target The instance name of a movie clip.

Returns

Nothing.

Description

Constructor; creates an instance of the Color object for the movie clip specified by the *target* parameter. You can then use the methods of that Color object to change the color of the entire target movie clip.

Example

The following example creates an instance of the Color object called `myColor` for the movie clip `myMovieClip` and sets its RGB value:

```
myColor = new Color(myMovieClip);  
myColor.setRGB(0xff9933);
```

Color.getRGB

Availability

Flash Player 5.

Usage

```
myColor.getRGB()
```

Parameters

None.

Returns

Nothing.

Description

Method; returns the numeric values set by the last `setRGB` call.

Example

The following code retrieves the RGB value for the Color object instance `myColor`, converts it to a hexadecimal string, and assigns it to the `value` variable.

```
value = myColor.getRGB().toString(16);
```

See also

`Color.setRGB`

Color.getTransform

Availability

Flash Player 5.

Usage

```
myColor.getTransform()
```

Parameters

None.

Returns

Nothing.

Description

Method; returns the transform value set by the last `setTransform` call.

See also

`Color.setTransform`

Color.setRGB

Availability

Flash Player 5.

Usage

```
myColor.setRGB(0xRRGGBB)
```

Parameters

`0xRRGGBB` The hexadecimal or RGB color to be set. *RR*, *GG*, and *BB* each consist of two hexadecimal digits specifying the offset of each color component. The `0x` tells the ActionScript compiler that the number is a hexadecimal value.

Description

Method; specifies an RGB color for an instance of the `Color` object. Calling this method overrides any previous settings by the `setTransform` method.

Returns

Nothing.

Example

This example sets the RGB color value for the movie clip `myMovie`. To see this code work, place a movie clip on the Stage with the instance name, `myMovie`. Then place the following code on Frame 1 in the main Timeline and choose **Control > Test Movie**.

```
myColor = new Color(myMovie);  
myColor.setRGB(0x993366);
```

See also

`Color.setTransform`

Color.setTransform

Availability

Flash Player 5.

Usage

```
myColor.setTransform(colorTransformObject);
```

Parameters

colorTransformObject An object created with the new `Object` constructor. This instance of the `Object` object must have the following properties that specify color transform values: `ra`, `rb`, `ga`, `gb`, `ba`, `bb`, `aa`, `ab`. These properties are explained below.

Returns

Nothing.

Description

Method; sets color transform information for an instance of the `Color` object. The *colorTransformObject* parameter is a generic object that you create from the new `Object` constructor. It has parameters specifying the percentage and offset values for the red, green, blue, and alpha (transparency) components of a color, entered in the format `0xRRGGBBAA`.

The parameters for a color transform object correspond to the settings in the Advanced Effect dialog box and are defined as follows:

- *ra* is the percentage for the red component (-100 to 100).
- *rb* is the offset for the red component (-255 to 255).
- *ga* is the percentage for the green component (-100 to 100).
- *gb* is the offset for the green component (-255 to 255).
- *ba* is the percentage for the blue component (-100 to 100).
- *bb* is the offset for the blue component (-255 to 255).
- *aa* is the percentage for alpha (-100 to 100).
- *ab* is the offset for alpha (-255 to 255).

You create a *colorTransformObject* parameter as follows:

```
myColorTransform = new Object();
myColorTransform.ra = 50;
myColorTransform.rb = 244;
myColorTransform.ga = 40;
myColorTransform.gb = 112;
myColorTransform.ba = 12;
myColorTransform.bb = 90;
myColorTransform.aa = 40;
myColorTransform.ab = 70;
```

You can also use the following syntax to create a *colorTransformObject* parameter:

```
myColorTransform = { ra: '50', rb: '244', ga: '40', gb: '112', ba: '12', bb:
    '90', aa: '40', ab: '70' }
```

Example

This example creates a new instance of the `Color` object for a target movie, creates a generic object called `myColorTransform` with the properties defined above, and uses the `setTransform` method to pass the `colorTransformObject` to a `Color` object. To use this code in a Flash (FLA) document, place it on Frame 1 on the main Timeline and place a movie clip on the Stage with the instance name `myMovie`, as in the following code:

```
// Create a color object called myColor for the target myMovie
myColor = new Color(myMovie);
// Create a color transform object called myColorTransform using
// the generic Object object
myColorTransform = new Object();
// Set the values for myColorTransform
myColorTransform = { ra: '50', rb: '244', ga: '40', gb: '112', ba: '12', bb:
    '90', aa: '40', ab: '70'};
// Associate the color transform object with the Color object
// created for myMovie
myColor.setTransform(myColorTransform);
```

continue

Availability

Flash Player 4.

Usage

`continue`

Parameters

None.

Returns

Nothing.

Description

Action; appears within several types of loop statements; it behaves differently in each type of loop.

In a `while` loop, `continue` causes the Flash interpreter to skip the rest of the loop body and jump to the top of the loop, where the condition is tested.

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

In a `for` loop, `continue` causes the Flash interpreter to skip the rest of the loop body and jump to the evaluation of the `for` loop's post-expression.

In a `for..in` loop, `continue` causes the Flash interpreter to skip the rest of the loop body and jump back to the top of the loop, where the next value in the enumeration is processed.

See also

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

CustomActions (object)

The methods of the CustomActions object allow a Flash movie playing in the Flash authoring tool to manage any custom actions that are registered with the authoring tool. A Flash movie can install and uninstall custom actions, retrieve the XML definition of a custom action, and retrieve the list of registered custom actions.

You can use these methods to build Flash movies that are extensions of the Flash authoring tool. Such an extension movie could, for example, use the Flash Application Protocol to navigate a UDDI repository and download Web services into the Actions toolbox.

Method summary for the CustomActions object

Method	Description
<code>CustomActions.get</code>	Reads the contents of a custom action XML definition file.
<code>CustomActions.install</code>	Installs a new custom action XML definition file.
<code>CustomActions.list</code>	Returns a list of all registered custom actions.
<code>CustomActions.uninstall</code>	Removes a custom action XML definition file.

CustomActions.get

Availability

Flash Player 6.

Usage

```
CustomActions.get(customActionsName)
```

Parameters

customActionsName The name of the custom action definition to retrieve.

Returns

Nothing.

Description

Method; reads the contents of the custom action XML definition file named *customActionsName*.

The name of the definition file must be a simple filename, without the .xml file extension, and without any directory separators (':', '/' or '\').

If the definition file specified by the *customActionsName* cannot be found, a value of `undefined` is returned. If the custom action XML definition specified by the *customActionsName* parameter is located, it is read in its entirety and returned as a string.

CustomActions.install

Availability

Flash Player 6.

Usage

```
CustomActions.install(customActionsName, customXMLDefinition)
```

Parameters

customActionsName The name of the custom action definition to install.

customXMLDefinition The text of the XML definition to install.

Returns

Nothing.

Description

Method; installs a new custom action XML definition file indicated by the *customActionsName* parameter. The contents of the file is specified by the string *customXMLDefinition*.

The name of the definition file must be a simple filename, without the .xml file extension, and without any directory separators (:', '/' or '\).

If a custom actions file already exists with the name *customActionsName*, it is overwritten.

If an error occurs during installation, a `false` value is returned; otherwise, a value of `true` is returned to indicate that the custom action has been successfully installed.

If the Configuration/ActionsPanel/CustomActions directory does not exist when this method is invoked, the directory is created.

CustomActions.list

Availability

Flash Player 6.

Usage

```
CustomActions.list()
```

Parameters

None.

Returns

An array.

Description

Method; returns an Array object containing the names of all the custom actions that are registered with the Flash authoring tool. The elements of the array are simple names, without the .xml file extension, and without any directory separators (for example, ":", "/", or "\"). If there are no registered custom actions, the `list` method returns a zero-length array. If an error occurs, the `list` method returns the value `undefined`.

CustomActions.uninstall

Availability

Flash Player 6.

Usage

```
CustomActions.uninstall(customActionsName)
```

Parameters

customActionsName The name of the custom action definition to uninstall.

Returns

Nothing.

Description

Method; removes the Custom Actions XML definition file named *customActionsName*.

The name of the definition file must be a simple filename, without the .xml file extension, and without any directory separators (':', '/' or '\').

If no custom actions are found with the name *customActionsName*, a value of `false` is returned. If the custom actions were successfully removed, a value of `true` is returned.

Date (object)

The Date object lets you retrieve date and time values relative to universal time (Greenwich Mean Time, now called universal time or UTC) or relative to the operating system on which the Flash Player is running. The methods of the Date object are not static, but apply only to the individual instance of the Date object specified when the method is called. The `Date.UTC` method is an exception; it is a static method.

The Date object handles daylight saving time differently depending on the operating system and the Flash Player version. Flash Player 6 handles daylight saving time on the following operating systems in these ways:

- Windows—the Date object automatically adjusts its output for daylight saving time. The Date object detects whether daylight saving time is employed in the current locale, and if so, it detects what the standard-to-daylight-saving-time transition date and times are. However, the transition dates currently in effect are applied to dates in the past and the future, so the daylight saving time bias may be calculated incorrectly for dates in the past when the locale had different transition dates.
- Mac OS 8 and 9—the Date object uses the current daylight-saving-time bias, regardless of the date or time being calculated. For example, in the U.S. Pacific time zone during August, when daylight saving time (DST) is in effect, a Date object containing the date Jan 1 2001 still reports DST time even though DST isn't in effect during January. This problem cannot be remedied on Mac OS 8 or 9 because a time zone information database is not available.
- Mac OS X—the Date object automatically adjusts its output for daylight saving time. The timezone information database in Mac OS X is used to determine whether any date or time in the present or past should have a daylight-saving-time bias applied.

Flash Player 5 handles daylight saving time on the following operating systems as follows:

- Mac OS 8 and 9—the behavior is the same as described for Flash Player 6.
- Windows—the U.S. rules for daylight saving time are always applied, which leads to incorrect transitions in Europe and other areas that employ daylight saving time but have different transition times than the U.S. Flash correctly detects whether DST is employed in the current locale.

To call the methods of the Date object, you must first create an instance of the Date object using the constructor for the Date object.

The Date object requires Flash Player 5.

Method summary for Date object

Method	Description
Date.getDate	Returns the day of the month according to local time.
Date.getDay	Returns the day of the week according to local time.
Date.getFullYear	Returns the four-digit year according to local time.
Date.getHours	Returns the hour according to local time.
Date.getMilliseconds	Returns the milliseconds according to local time.
Date.getMinutes	Returns the minutes according to local time.
Date.getMonth	Returns the month according to local time.
Date.getSeconds	Returns the seconds according to local time.
Date.getTime	Returns the number of milliseconds since midnight January 1, 1970, universal time.
Date.getTimezoneOffset	Returns the difference, in minutes, between the computer's local time and the universal time.
Date.getUTCDate	Returns the day (date) of the month according to universal time.
Date.getUTCDay	Returns the day of the week according to universal time.
Date.getUTCFullYear	Returns the four-digit year according to universal time.
Date.getUTCHours	Returns the hour according to universal time.
Date.getUTCMilliseconds	Returns the milliseconds according to universal time.
Date.getUTCMinutes	Returns the minutes according to universal time.
Date.getUTCMonth	Returns the month according to universal time.
Date.getUTCSeconds	Returns the seconds according to universal time.
Date.getYear	Returns the year according to local time.
Date.setDate	Sets the day of the month according to local time. Returns the new time in milliseconds.
Date.setFullYear	Sets the full year according to local time. Returns the new time in milliseconds.
Date.setHours	Sets the hour according to local time. Returns the new time in milliseconds.
Date.setMilliseconds	Sets the milliseconds according to local time. Returns the new time in milliseconds.
Date.setMinutes	Sets the minutes according to local time. Returns the new time in milliseconds.

Method	Description
<code>Date.setMonth</code>	Sets the month according to local time. Returns the new time in milliseconds.
<code>Date.setSeconds</code>	Sets the seconds according to local time. Returns the new time in milliseconds.
<code>Date.setTime</code>	Sets the date in milliseconds. Returns the new time in milliseconds.
<code>Date.setUTCDate</code>	Sets the date according to universal time. Returns the new time in milliseconds.
<code>Date.setUTCFullYear</code>	Sets the year according to universal time. Returns the new time in milliseconds.
<code>Date.setUTCHours</code>	Sets the hour according to universal time. Returns the new time in milliseconds.
<code>Date.setUTCMilliseconds</code>	Sets the milliseconds according to universal time. Returns the new time in milliseconds.
<code>Date.setUTCMinutes</code>	Sets the minutes according to universal time. Returns the new time in milliseconds.
<code>Date.setUTCMonth</code>	Sets the month according to universal time. Returns the new time in milliseconds.
<code>Date.setUTCSeconds</code>	Sets the seconds according to universal time. Returns the new time in milliseconds.
<code>Date.setYear</code>	Sets the year according to local time.
<code>Date.toString</code>	Returns a string value representing the date and time stored in the specified Date object.
<code>Date.UTC</code>	Returns the number of milliseconds between midnight on January 1, 1970, universal time, and the specified time.

Constructor for the Date object

Availability

Flash Player 5.

Usage

```
new Date()
```

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

Parameters

year A value of 0 to 99 indicates 1900 through 1999; otherwise all four digits of the year must be specified.

month An integer from 0 (January) to 11 (December).

date An integer from 1 to 31. This parameter is optional.

hour An integer from 0 (midnight) to 23 (11 p.m.).

minute An integer from 0 to 59. This parameter is optional.

second An integer from 0 to 59. This parameter is optional.

millisecond An integer from 0 to 999. This parameter is optional.

Returns

An integer.

Description

Object; constructs a new Date object that holds the current date and time, or the date specified.

Example

The following example retrieves the current date and time.

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

The following example creates a new Date object for Gary's birthday, August 7, 1974.

```
gary_birthday = new Date (74, 7, 7);
```

The following example creates a new Date object, concatenates the returned values of the Date object methods `getMonth`, `getDate`, and `getFullYear`, and displays them in the text field specified by the variable `dateTextField`.

```
myDate = new Date();  
dateTextField = ((myDate.getMonth() + 1) + "/" + myDate.getDate() + "/" +  
    myDate.getFullYear());
```

Date.getDate

Availability

Flash Player 5.

Usage

```
myDate.getDate()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the day of the month (an integer from 1 to 31) of the specified Date object according to local time. Local time is determined by the operating system on which the Flash Player is running.

Date.getDay

Availability

Flash Player 5.

Usage

```
myDate.getDay()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the day of the week (0 for Sunday, 1 for Monday, and so on) of the specified Date object according to local time. Local time is determined by the operating system on which the Flash Player is running.

Date.getFullYear

Availability

Flash Player 5.

Usage

```
myDate.getFullYear()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the full year (a four-digit number, for example, 2000) of the specified Date object, according to local time. Local time is determined by the operating system on which the Flash Player is running.

Example

The following example uses the constructor to create a new Date object and send the value returned by the `getFullYear` method to the Output window:

```
myDate = new Date();  
trace(myDate.getFullYear());
```

Date.getHours

Availability

Flash Player 5.

Usage

```
myDate.getHours()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the hour (an integer from 0 to 23) of the specified Date object, according to local time. Local time is determined by the operating system on which the Flash Player is running.

Date.getMilliseconds

Availability

Flash Player 5.

Usage

```
myDate.getMilliseconds()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the milliseconds (an integer from 0 to 999) of the specified Date object, according to local time. Local time is determined by the operating system on which the Flash Player is running.

Date.getMinutes

Availability

Flash Player 5.

Usage

```
myDate.getMinutes()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the minutes (an integer from 0 to 59) of the specified Date object, according to local time. Local time is determined by the operating system on which the Flash Player is running.

Date.getMonth

Availability

Flash Player 5.

Usage

myDate.getMonth()

Parameters

None.

Returns

An integer.

Description

Method; returns the month (0 for January, 1 for February, and so on) of the specified Date object, according to local time. Local time is determined by the operating system on which the Flash Player is running.

Date.getSeconds

Availability

Flash Player 5.

Usage

myDate.getSeconds()

Parameters

None.

Returns

An integer.

Description

Method; returns the seconds (an integer from 0 to 59) of the specified Date object, according to local time. Local time is determined by the operating system on which the Flash Player is running.

Date.getTime

Availability

Flash Player 5.

Usage

```
myDate.getTime()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the number of milliseconds since midnight January 1, 1970, universal time, for the specified Date object. Use this method to represent a specific instant in time when comparing two or more Date objects.

Date.getTimezoneOffset

Availability

Flash Player 5.

Usage

```
mydate.getTimezoneOffset()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the difference, in minutes, between the computer's local time and universal time.

Example

The following example returns the difference between the local daylight saving time for San Francisco and universal time. Daylight savings time is factored into the returned result only if the date defined in the Date object is during the daylight saving time.

```
trace(new Date().getTimezoneOffset());  
// 420 is displayed in the Output window  
// (7 hours * 60 minutes/hour = 420 minutes)  
// This example is Pacific Daylight Time (PDT, GMT-0700).  
// Result will vary depending on locale and time of year.
```


Date.getUTCDate

Availability

Flash Player 5.

Usage

```
myDate.getUTCDate()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the day (date) of the month in the specified Date object, according to universal time.

Date.getUTCDay

Availability

Flash Player 5.

Usage

```
myDate.getUTCDate()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the day of the week of the specified Date object, according to universal time.

Date.getUTCFullYear

Availability

Flash Player 5.

Usage

```
myDate.getUTCFullYear()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the four-digit year of the specified Date object, according to universal time.

Date.getUTCHours

Availability

Flash Player 5.

Usage

```
myDate.getUTCHours()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the hours of the specified Date object, according to universal time.

Date.getUTCMilliseconds

Availability

Flash Player 5.

Usage

```
myDate.getUTCMilliseconds()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the milliseconds of the specified Date object, according to universal time.

Date.getUTCMinutes

Availability

Flash Player 5.

Usage

```
myDate.getUTCMinutes()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the minutes of the specified Date object, according to universal time.

Date.getUTCMonth

Availability

Flash Player 5.

Usage

```
myDate.getUTCMonth()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the month of the specified Date object, according to universal time.

Date.getUTCSeconds

Availability

Flash Player 5.

Usage

```
myDate.getUTCSeconds()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the seconds in the specified Date object, according to universal time.

Date.getYear

Availability

Flash Player 5.

Usage

```
myDate.getYear()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the year of the specified Date object, according to local time. Local time is determined by the operating system on which the Flash Player is running. The year is the full year minus 1900. For example, the year 2000 is represented as 100.

Date.setDate

Availability

Flash Player 5.

Usage

```
myDate.setDate(date)
```

Parameters

date An integer from 1 to 31.

Returns

An integer.

Description

Method; sets the day of the month for the specified Date object, according to local time, and returns the new time in milliseconds. Local time is determined by the operating system on which the Flash Player is running.

Date.setFullYear

Availability

Flash Player 5.

Usage

```
myDate.setFullYear(year [, month [, date]] )
```

Parameters

year A four-digit number specifying a year. Two-digit numbers do not represent years; for example, 99 is not the year 1999, but the year 99.

month An integer from 0 (January) to 11 (December). This parameter is optional.

date A number from 1 to 31. This parameter is optional.

Returns

An integer.

Description

Method; sets the year of the specified Date object, according to local time, and returns the new time in milliseconds. If the *month* and *date* parameters are specified, they are also set to local time. Local time is determined by the operating system on which the Flash Player is running.

Calling this method does not modify the other fields of the specified Date object but the `getUTCDay` and `getDay` methods may report a new value if the day of the week changes as a result of calling this method.

Date.setHours

Availability

Flash Player 5.

Usage

```
myDate.setHours(hour)
```

Parameters

hour An integer from 0 (midnight) to 23 (11 p.m.).

Returns

An integer.

Description

Method; sets the hours for the specified Date object according to local time, and returns the new time in milliseconds. Local time is determined by the operating system on which the Flash Player is running.

Date.setMilliseconds

Availability

Flash Player 5.

Usage

```
myDate.setMilliseconds(milliseconds)
```

Parameters

milliseconds An integer from 0 to 999.

Returns

An integer.

Description

Method; sets the milliseconds for the specified Date object according to local time, and returns the new time in milliseconds. Local time is determined by the operating system on which the Flash Player is running.

Date.setMinutes

Availability

Flash Player 5.

Usage

```
myDate.setMinutes(minute)
```

Parameters

minute An integer from 0 to 59.

Returns

An integer.

Description

Method; sets the minutes for a specified Date object according to local time, and returns the new time in milliseconds. Local time is determined by the operating system on which the Flash Player is running.

Date.setMonth

Availability

Flash Player 5.

Usage

```
myDate.setMonth(month [, date ])
```

Parameters

month An integer from 0 (January) to 11 (December).

date An integer from 1 to 31. This parameter is optional.

Returns

An integer.

Description

Method; sets the month for the specified Date object in local time, and returns the new time in milliseconds. Local time is determined by the operating system on which the Flash Player is running.

Date.setSeconds

Availability

Flash Player 5.

Usage

```
myDate.setSeconds(second)
```

Parameters

second An integer from 0 to 59.

Returns

An integer.

Description

Method; sets the seconds for the specified Date object in local time, and returns the new time in milliseconds. Local time is determined by the operating system on which the Flash Player is running.

Date.setTime

Availability

Flash Player 5.

Usage

```
myDate.setTime(milliseconds)
```

Parameters

milliseconds An integer value where 0 is 0:00 GMT 1970 Jan 1.

Returns

An integer.

Description

Method; sets the date for the specified Date object in milliseconds since midnight on January 1, 1970, and returns the new time in milliseconds.

Date.setUTCDate

Availability

Flash Player 5.

Usage

```
myDate.setUTCDate(date)
```

Parameters

date An integer from 1 to 31.

Returns

An integer.

Description

Method; sets the date for the specified Date object in universal time, and returns the new time in milliseconds. Calling this method does not modify the other fields of the specified Date object, but the `getUTCDay` and `getDay` methods may report a new value if the day of the week changes as a result of calling this method.

Date.setUTCFullYear

Availability

Flash Player 5.

Usage

```
myDate.setUTCFullYear(year [, month [, date]])
```

Parameters

year The year specified as a full four-digit year, for example, 2000.

month An integer from 0 (January) to 11 (December). This parameter is optional.

date An integer from 1 to 31. This parameter is optional.

Returns

An integer.

Description

Method; sets the year for the specified Date object (*mydate*) in universal time, and returns the new time in milliseconds.

Optionally, this method can also set the month and date represented by the specified Date object. No other fields of the Date object are modified. Calling `setUTCFullYear` may cause `getUTCDay` and `getDay` to report a new value if the day of the week changes as a result of this operation.

Date.setUTCHours

Availability

Flash Player 5.

Usage

```
myDate.setUTCHours(hour [, minute [, second [, millisecond]])
```

Parameters

hour An integer from 0 (midnight) to 23 (11 p.m.).

minute An integer from 0 to 59. This parameter is optional.

second An integer from 0 to 59. This parameter is optional.

millisecond An integer from 0 to 999. This parameter is optional.

Returns

An integer.

Description

Method; sets the hour for the specified Date object in universal time, and returns the new time in milliseconds.

Date.setUTCMilliseconds

Availability

Flash Player 5.

Usage

```
myDate.setUTCMilliseconds(millisecond)
```

Parameters

millisecond An integer from 0 to 999.

Returns

An integer.

Description

Method; sets the milliseconds for the specified Date object in universal time, and returns the new time in milliseconds.

Date.setUTCMinutes

Availability

Flash Player 5.

Usage

```
myDate.setUTCMinutes(minute [, second [, millisecond]])
```

Parameters

minute An integer from 0 to 59.

second An integer from 0 to 59. This parameter is optional.

millisecond An integer from 0 to 999. This parameter is optional.

Returns

An integer.

Description

Method; sets the minute for the specified Date object in universal time, and returns the new time in milliseconds.

Date.setUTCMonth

Availability

Flash Player 5.

Usage

```
myDate.setUTCMonth(month [, date])
```

Parameters

month An integer from 0 (January) to 11 (December).

date An integer from 1 to 31. This parameter is optional.

Returns

An integer.

Description

Method; sets the month, and optionally the day (date), for the specified Date object in universal time, and returns the new time in milliseconds. Calling this method does not modify the other fields of the specified Date object, but the `getUTCDay` and `getDay` methods may report a new value if the day of the week changes as a result of specifying the *date* parameter when calling `setUTCMonth`.

Date.setUTCSeconds

Availability

Flash Player 5.

Usage

```
myDate.setUTCSeconds(second [, millisecond])
```

Parameters

second An integer from 0 to 59.

millisecond An integer from 0 to 999. This parameter is optional.

Returns

An integer.

Description

Method; sets the seconds for the specified Date object in universal time, and returns the new time in milliseconds.

Date.setYear

Availability

Flash Player 5.

Usage

```
myDate.setYear(year)
```

Parameters

year If *year* is an integer between 0–99, `setYear` sets the year at 1900 + *year*; otherwise, the year is the value of the *year* parameter.

Returns

An integer.

Description

Method; sets the year for the specified `Date` object in local time, and returns the new time in milliseconds. Local time is determined by the operating system on which the Flash Player is running.

Date.toString

Availability

Flash Player 5.

Usage

```
myDate.toString()
```

Parameters

None.

Returns

A string.

Description

Method; returns a string value for the specified date object in a readable format, and returns the new time in milliseconds.

Example

The following example returns the information in the `dateOfBirth` `Date` object as a string.

```
var dateOfBirth = new Date(74, 7, 7, 18, 15);  
trace (dateOfBirth.toString());
```

Output (for Pacific Standard Time):

```
Wed Aug 7 18:15:00 GMT-0700 1974
```

Date.UTC

Availability

Flash Player 5.

Usage

```
Date.UTC(year, month [, date [, hour [, minute [, second [, millisecond ]]]]])
```

Parameters

year A four-digit number, for example, 2000.

month An integer from 0 (January) to 11 (December).

date An integer from 1 to 31. This parameter is optional.

hour An integer from 0 (midnight) to 23 (11 p.m.).

minute An integer from 0 to 59. This parameter is optional.

second An integer from 0 to 59. This parameter is optional.

millisecond An integer from 0 to 999. This parameter is optional.

Returns

An integer.

Description

Method; returns the number of milliseconds between midnight on January 1, 1970, universal time, and the time specified in the parameters. This is a static method that is invoked through the Date object constructor, not through a specific Date object. This method lets you create a Date object that assumes universal time, whereas the Date constructor assumes local time.

Example

The following example creates a new `gary_birthday` Date object defined in universal time. This is the universal time variation of the example used for the `new Date` constructor method:

```
gary_birthday = new Date(Date.UTC(1974, 7, 8));
```

default

Availability

Flash Player 6.

Usage

default: *statements*

Parameters

statements Any statements.

Returns

Nothing.

Description

Keyword; defines the default case for a `switch` action. The statements execute if the *Expression* parameter of the `switch` action doesn't equal (using strict equality) any of the *Expression* parameters that follow the `case` keywords for a given `switch` action.

A `switch` is not required to have a default case. A default case does not have to be last in the list. Using a default action outside a `switch` action is an error and the script doesn't compile.

Example

In the following example, the expression A does not equal the expressions B or D so the statement following the default keyword is run and the trace action is sent to the Output window.

```
switch ( A ) {  
    case B:  
        C;  
        break;  
    case D:  
        E;  
        break;  
    default:  
        trace ("nospecific case was encountered");  
}
```

See also

`switch`, `case`, `break`

delete

Availability

Flash Player 5.

Usage

delete *reference*

Parameters

reference The name of the variable or object to eliminate.

Returns

Nothing.

Description

Operator; destroys the object or variable specified by the *reference* parameter, and returns `true` if the object was successfully deleted; otherwise returns a value of `false`. This operator is useful for freeing up memory used by scripts. Although `delete` is an operator, it is typically used as a statement, as in the following:

```
delete x;
```

The `delete` operator may fail and return `false` if the *reference* parameter does not exist, or may not be deleted. Predefined objects and properties, and variables declared with `var`, may not be deleted. You cannot use the `delete` operator to remove movie clips.

Example

The following example creates an object, uses it, and then deletes it after it is no longer needed.

```
account = new Object();
account.name = 'Jon';
account.balance = 10000;

delete account;
```

Example

The following example deletes a property of an object.

```
// create the new object "account"
account = new Object();
// assign property name to the account
account.name = 'Jon';
// delete the property
delete account.name;
```

Example

The following is another example of deleting an object property.

```
// create an Array object with length 0
array = new Array();
// Array.length is now 1
array[0] = "abc";
// add another element to the array, Array.length is now 2
array[1] = "def";
// add another element to array, Array.length is now 3
array[2] = "ghi";
// array[2] is deleted, but Array.length is not changed,
delete array[2];
```

The following example illustrates the behavior of `delete` on object references:

```
// create a new object, and assign the variable ref1
// to refer to the object
ref1 = new Object();
ref1.name = "Jody";
// copy the reference variable into a new variable
// and delete ref1
ref2 = ref1;
delete ref1;
```

If `ref1` had not been copied into `ref2`, the object would have been deleted when `ref1` was deleted, because there would be no references to it. If you delete `ref2`, there will no longer be any references to the object; it will be destroyed, and the memory it was using will be made available.

See also

`var`

do while

Availability

Flash Player 4.

Usage

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

Parameters

condition The condition to evaluate.

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

Returns

Nothing.

Description

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

See also

break, continue

duplicateMovieClip

Availability

Flash Player 4.

Usage

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

Parameters

target The target path of the movie clip to duplicate.

newname A unique identifier for the duplicated movie clip.

depth A unique depth level for the duplicated movie clip. The depth level is a stacking order for duplicated movie clips. This stacking order is much like the stacking order of layers in the Timeline; movie clips with a lower depth level are hidden under clips with a higher stacking order. You must assign each duplicated movie clip a unique depth level to prevent it from replacing movies on occupied depths.

Returns

Nothing.

Description

Action; creates an instance of a movie clip while the movie is playing. The playhead in duplicate movie clips always starts at Frame 1, regardless of where the playhead is in the original (or “parent”) movie clip. Variables in the parent movie clip are not copied into the duplicate movie clip. If the parent movie clip is deleted the duplicate movie clip is also deleted. Use the `removeMovieClip` action or method to delete a movie clip instance created with `duplicateMovieClip`.

Example

This statement duplicates the movie clip instance `flower` ten times. The variable `i` is used to create a new instance name and a unique depth for each duplicated movie clip.

```
on (release) {  
    amount = 10;  
    while (amount>0) {  
        duplicateMovieClip (_root.flower, "mc"+i, i);  
        setProperty ("mc"+i, _x, random(275));  
        setProperty ("mc"+i, _y, random(275));  
        setProperty ("mc"+i, _alpha, random(275));  
        setProperty ("mc"+i, _xscale, random(50));  
        setProperty ("mc"+i, _yscale, random(50));  
        i++;  
        amount--;  
    }  
}
```

See also

`MovieClip.duplicateMovieClip`, `removeMovieClip`, `MovieClip.removeMovieClip`

else

Availability

Flash Player 4.

Usage

```
else statement  
else {...statement(s)...}
```

Parameters

condition An expression that evaluates to true or false.

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

Returns

Nothing.

Description

Action; specifies the statements to run if the condition in the `if` statement returns false.

See also

`if`

else if

Availability

Flash Player 4.

Usage

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

Parameters

condition An expression that evaluates to true or false.

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

Returns

Nothing.

Description

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

Example

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

```
// if the object goes off bounds,  
// send it back and reverse its travel speed  
if (this._x>rightBound) {  
    this._x = rightBound;  
    xInc = -xInc;  
} else if (this._x<leftBound) {  
    this._x = leftBound;  
    xInc = -xInc;  
} else if (this._y>bottomBound) {  
    this._y = bottomBound;  
    yInc = -yInc;  
} else if (this._y<topBound) {  
    this._y = topBound;  
    yInc = -yInc;  
}
```

See also

[if](#)

#endinitclip

Availability

Flash Player 6.

Usage

```
#endinitclip
```

Parameters

None.

Returns

Nothing.

Description

Action; indicates the end of a block of component initialization actions.

Example

```
#initclip
...component initialization actions go here...
#endinitclip
```

See also

#initclip

eq (equal-string specific)

Availability

Flash Player 4. This operator has been deprecated in Flash 5 in favor of the == (equality) operator.

Usage

```
expression1 eq expression2
```

Parameters

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

Returns

Nothing.

Description

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

See also

== (equality)

escape

Availability

Flash Player 5.

Usage

`escape(expression)`

Parameters

expression The expression to convert into a string and encode in a URL-encoded format.

Returns

Nothing.

Description

Function; converts the parameter to a string and encodes it in a URL-encoded format, where all non-alphanumeric characters are escaped with % hexadecimal sequences.

Example

Running the following code gives the result, `Hello%7B%5BWorld%5D%7D`.

```
escape("Hello{[World]}");
```

See also

`unescape`

eval

Availability

Flash Player 5 or later for full functionality. You can use the `eval` function when exporting to Flash Player 4, but you must use slash notation, and can only access variables, not properties or objects.

Usage

`eval(expression);`

Parameters

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

Returns

Nothing.

Description

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

In Flash 4, the `eval` function was used to simulate arrays; in Flash 5, it is recommended that you use the `Array` object to simulate arrays.

You can also use the `eval` function to dynamically set and retrieve the value of a variable or instance name. However, you can also do this with the array access operator (`[]`).

Note: The ActionScript `eval` action is not the same as the JavaScript `eval` function, and cannot be used to evaluate statements.

Example

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

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

See also

[Array \(object\)](#)

evaluate

Availability

Flash Player 5.

Usage

statement

Parameters

None.

Returns

Nothing.

Description

Action; creates a new empty line and inserts a semicolon (;) for writing statements to be evaluated in the Actions panel.

false

Availability

Flash Player 5.

Usage

`true`

Description

A unique Boolean value that represents the opposite of `true`.

See also

`true`

FCheckBox (component)

The CheckBox component in the Flash authoring environment provides drag-and-drop functionality for adding check boxes to Flash documents; it also provides a user interface for setting basic parameters. The methods of the FCheckBox component allow you to control check boxes at runtime: you can create check boxes, control check boxes created in authoring, set or override basic parameters, and set additional runtime options. You do not need to use a constructor to access the methods of components.

The CheckBox component is supported by Flash Player 6.

Component methods do not perform error checking for type, as do other native ActionScript objects and actions; therefore, it is recommended that you validate parameters before passing them to methods.

For information on using the CheckBox component, setting parameters during authoring, and changing the color and appearance of components, see “Customizing component colors and text” and “Customizing component skins” in the “Using Components” chapter of *Using Flash*.

Method summary for the FCheckBox component

Method	Description
<code>FCheckBox.setEnabled</code>	Returns <code>true</code> if the check box is enabled, <code>false</code> if it is disabled.
<code>FCheckBox.getLabel</code>	Returns the label applied to the check box as a string.
<code>FCheckBox.getValue</code>	Returns <code>true</code> if the check box is selected, <code>false</code> if it is not selected.
<code>FCheckBox.registerSkinElement</code>	Registers a skin element to a property.
<code>FCheckBox.setChangeHandler</code>	Specifies a change handler to call when the value of the check box changes.
<code>FCheckBox.setEnabled</code>	Determines whether the check box is enabled or disabled.
<code>FCheckBox.setLabel</code>	Specifies text for the label of the check box.
<code>FCheckBox.setLabelPlacement</code>	Specifies whether the label appears to the left or right of the check box.
<code>FCheckBox.setSize</code>	Sets the width of the check box, in pixels, and redraws it.
<code>FCheckBox.setStyleProperty</code>	Sets a single style property for a component.
<code>FCheckBox.setValue</code>	Selects or deselects the check box and triggers the change handler function.

FCheckBox.setEnabled

Availability

Flash Player 6.

Usage

```
myCheckBox.setEnabled()
```

Parameters

None.

Returns

A Boolean value indicating whether the check box instance is enabled (`true`) or disabled (`false`).

Description

Method; indicates whether the check box instance is enabled or disabled.

Example

The following code returns the enabled state of `checkBox1` to the Output window.

```
trace(checkBox1.setEnabled());
```

See also

`FCheckBox.setValue`

FCheckBox.getLabel

Availability

Flash Player 6.

Usage

```
myCheckBox.getLabel()
```

Parameters

None.

Returns

A text string.

Description

Method; retrieves the label of the check box.

Example

The following code returns the label of `checkBox1`.

```
checkBox1.getLabel();
```

See also

`FCheckBox.setLabel`

FCheckBox.getValue

Availability

Flash Player 6.

Usage

```
myCheckBox.getValue()
```

Parameters

None.

Returns

A Boolean value indicating whether the check box is selected (`true`) or not (`false`).

Description

Method; indicates whether the check box is selected.

Example

The following code returns the selected value of `KowalczykBox` to the Output window.

```
trace(KowalczykBox.getValue());
```

See also

`FCheckBox.setValue`

FCheckBox.registerSkinElement

Availability

Flash Player 6.

Usage

```
myCheckBox.registerSkinElement(element, styleProperty)
```

Parameters

element A movie clip instance.

styleProperty The name of an `FStyleFormat` property.

Returns

Nothing.

Description

Method; registers a skin element to a style property. Skin elements are registered to properties in the first frame of the Read Me layer of each skin the library.

Components are made up of skins, and each skin is composed of several skin elements, each of which can be registered to a property of the `FStyleFormat` object. These properties are assigned values by the style format assigned to a component. By default, all Flash UI components are assigned the `globalStyleFormat` object, which is an instance of the `FStyleFormat` object.

Use this method to register custom skin elements and properties to Flash UI or custom component skins by editing the code in the first frame of the Read Me layer of a skin in the library.

The `FCheckBox` component uses the skins in the `FCheckBox Skins` folder once you've added the component to the Flash document.

For more information, see “Customizing component skins” in the “Using Components” chapter of *Using Flash*.

Example

The following code registers the custom skin element `customChk_mc` to the `check` property in the first frame of the Read Me layer of the `fcg_check` skin in the `FCCheckBox Skins` folder in the library.

```
check1.registerSkinElement(customChk_mc, "check");
```

FCCheckBox.setChangeHandler

Availability

Flash Player 6.

Usage

```
myCheckBox.setChangeHandler(functionName, [location])
```

Parameters

functionName A string specifying the name of the handler function to execute when the value of the check box changes. If the *location* parameter is not specified, this function must be in the same Timeline as the component instance.

location A path reference to a data object, movie clip, or Timeline that contains the specified function. This parameter is optional and defaults to the parent Timeline of the component.

Returns

Nothing.

Description

Method; specifies a change handler to call when the value of the check box changes. You can specify the same change handler function for more than one component; the function always accepts the instance of the component that has changed as a parameter. Calling this method overrides the Change Handler parameter value specified in authoring.

For more information, see “Writing change handler functions for components” in the “Using Components” chapter of *Using Flash*.

Example

The following code specifies `myHandler` as the function called when the value of `checkBox1` changes. Because the *location* parameter is not specified, `myHandler` must be in the same Timeline as the component instance.

The component parameter in `myHandler` is automatically filled in with the instance of a component (the component that has changed as the result of user input and that specifies `myHandler` as its change handler). The actions defined in `myHandler` specify that when the user selects a check box, the name of the component and “has been selected” is written to the Output window.

```
checkBox1.setChangeHandler("myHandler");
function myHandler(component){
trace(component._name + " has been selected ");
}
```

If in the preceding example `myHandler` is a function located in the great-grandparent Timeline of the component's Timeline, the first line of code would be as follows:

```
check1.setChangeHandler("myHandler", _parent._parent._parent);
```

The following code creates the function `myHandler` in an instance of `myObject` (which is of class `Object`), and then specifies `myHandler` as the function for `check1`.

```
myObject = new Object();
myObject.myHandler = function(component){
    trace(component._name + " has been selected ");
}
```

```
check1.setChangeHandler("myHandler", myObject);
```

FCheckBox.setEnabled

Availability

Flash Player 6.

Usage

```
myCheckBox.setEnabled(enable)
```

Parameters

enable A Boolean value specifying whether the check box is enabled (`true`) or disabled (`false`).

Returns

Nothing.

Description

Method; specifies whether the check box is enabled (`true`) or disabled (`false`). If a check box is disabled, it does not accept mouse or keyboard interaction from the user. If you omit this parameter, the method defaults to `true`.

Example

The following code disables `checkBox1`.

```
checkBox1.setEnabled(false);
```

FCheckBox.setLabel

Availability

Flash Player 6.

Usage

```
myCheckBox.setLabel(label)
```

Parameters

label A string specifying the text label for the check box.

Returns

Nothing.

Description

Method; specifies the text label for the check box. By default, the label appears to the right of the check box. Calling this method overrides the `label` parameter specified in authoring.

Example

The following code applies the label “Send more information” to `checkBox1`.

```
checkBox1.setLabel("Send more information");
```

See also

`FCheckBox.getLabel`, `FCheckBox.setLabelPlacement`

FCheckBox.setLabelPlacement

Availability

Flash Player 6.

Usage

```
myCheckBox.setLabelPlacement(labelPosition)
```

Parameters

labelPosition A text string; specify "left" or "right".

Returns

Nothing.

Description

Method; specifies whether the label appears to the left or right of the check box. Calling this method overrides the Label Placement parameter value set during authoring.

Example

The following code places the label for `checkBox1` to the left of the check box.

```
checkBox1.setLabelPlacement("left");
```

See also

`FCheckBox.setLabel`

FCheckBox.setSize

Availability

Flash Player 6.

Usage

```
myCheckBox.setSize(width)
```

Parameters

width An integer specifying the width of the check box, in pixels.

Returns

Nothing.

Description

Method; specifies the width of the check box and redraws it. You cannot set the height of check box components. Calling this method overrides width scaling applied during authoring.

For more information, see “Sizing CheckBox components” in the “Using Components” chapter of *Using Flash*.

Example

The following code sets the width of `checkBox1` to 200 pixels.

```
checkBox1.setSize(200);
```

FCheckBox.setStyleProperty

Availability

Flash Player 6.

Usage

```
myCheckBox.setStyleProperty(styleProperty, value)
```

Parameters

styleProperty A string specifying a property of the `FStyleFormat` object.

value The value to set for the property.

Returns

Nothing.

Description

Method; sets an `FStyleFormat` property for an individual check box instance. Calling this method to specify a property overrides the setting for this property in the style format assigned to the component. Passing `undefined` as the value for a property removes all styles for that property.

To set `FStyleFormat` properties for multiple components, create a custom style format. For more information on creating custom style formats, see “Customizing component colors and text” in the “Using Components” chapter of *Using Flash*.

Example

The following code sets the `shadow` property of `checkBox1` to `0x000000` (black).

```
checkBox1.setStyleProperty("shadow", 0x000000);
```

See also

`FStyleFormat` (object)

FCheckBox.setValue

Availability

Flash Player 6.

Usage

```
myCheckBox.setValue(select)
```

Parameters

select A Boolean value specifying whether the check box is selected (`true`) or not (`false`).

Returns

Nothing.

Description

Method; selects or deselects *myCheckBox* and triggers the change handler function specified (if any) at runtime. The default value is `true`.

Although calling this method overrides the Initial Value parameter value specified in authoring, do not use the method for this purpose because it also triggers the associated change handler function. To set the Initial Value parameter of a check box at runtime, use `FCheckBox.setStyleProperty`.

Example

The following code selects the instance of `checkBox1` and triggers any specified change handler function.

```
checkBox1.setValue(true);
```

See also

`FCheckBox.getValue`

FComboBox (component)

The `ComboBox` component in the Flash authoring environment provides drag-and-drop functionality for adding scrollable single-selection drop-down lists to Flash documents; it also provides a user interface for setting basic parameters. The methods of the `FComboBox` component allow you to control combo boxes at runtime: you can create combo boxes, control combo boxes created in authoring, set or override basic parameters, and set additional runtime options. You do not need to use a constructor to access the methods of components.

The `ComboBox` component creates both static and editable combo boxes. A static combo box is a scrollable drop-down list that allows users to select items. An editable combo box is a scrollable drop-down list with a text field at the top. You can let users enter text in the text field to make the combo box scroll to the desired item, or you can use the text field to set displayed text at runtime.

Both the static and editable versions of the ComboBox component list items from top to bottom using a zero-based indexing system. If the number of items in the combo box list creates a drop-down list that exceeds the available space below the component, the list opens upward instead of dropping down.

Component methods do not perform error checking for type, as do other native ActionScript objects and actions; therefore, it is recommended that you validate parameters before passing them to methods.

The ComboBox component is supported by Flash Player 6 and later versions of the Flash Player.

For information on using the ComboBox component, setting parameters during authoring, and changing the color and appearance of components, see “Customizing component colors and text” and “Customizing component skins” in the “Using Components” chapter of *Using Flash*.

Method summary for the FComboBox component

Method	Description
<code>FComboBox.addItem</code>	Adds a new item to the end of the combo box list.
<code>FComboBox.addItemAt</code>	Adds a new item to the combo box list at the specified index.
<code>FComboBox.setEnabled</code>	Returns <code>true</code> if the combo box is enabled, <code>false</code> if it is disabled.
<code>FComboBox.getItemAt</code>	Returns the item at the specified index as an object with the properties <code>label</code> and <code>data</code> .
<code>FComboBox.getLength</code>	Returns the number of items listed in the combo box.
<code>FComboBox.getRowCount</code>	Returns the number of rows visible in the combo box.
<code>FComboBox.getScrollPosition</code>	Returns the index of the item at the top of the combo box.
<code>FComboBox.getSelectedIndex</code>	Returns the index of the currently selected item.
<code>FComboBox.getSelectedItem</code>	Returns the currently selected item as an object with the properties <code>label</code> and <code>data</code> .
<code>FComboBox.getValue</code>	Returns the text in the input field for editable combo boxes; returns the label or data of the selected item for static combo boxes.
<code>FComboBox.registerSkinElement</code>	Registers a skin element to a property.
<code>FComboBox.removeAll</code>	Removes all items from the combo box.
<code>FComboBox.removeItemAt</code>	Removes the item at the specified index.
<code>FComboBox.replaceItemAt</code>	Replaces the label and data of an item at the specified index.
<code>FComboBox.setChangeHandler</code>	Assigns a function to call each time an item is selected or the user enters text in the text field.
<code>FComboBox.setDataProvider</code>	Registers an outside object to the component as a data source.
<code>FComboBox.setEditable</code>	Determines whether the combo box is editable (<code>true</code>) or static (<code>false</code>).
<code>FComboBox.setEnabled</code>	Specifies whether the combo box is enabled (<code>true</code>) or disabled (<code>false</code>).
<code>FComboBox.setItemSymbol</code>	Registers a symbol linkage ID to use for displaying list items in the combo box.
<code>FComboBox.setRowCount</code>	Determines the number of items displayed in the combo box without a scroll bar.
<code>FComboBox.setSelectedIndex</code>	Selects the item at the specified index.

Method	Description
<code>FComboBox.setSize</code>	Sets the pixel width of the combo box.
<code>FComboBox.setStyleProperty</code>	Sets a single style property for an instance of a component.
<code>FComboBox.setValue</code>	Specifies the text displayed in the text field at the top of an editable combo box.
<code>FComboBox.sortItemsBy</code>	Sorts the items in the list box alphabetically or numerically by label or by data.

FComboBox.addItem

Availability

Flash Player 6.

Usage

```
myComboBox.addItem( label [,data])
```

Parameters

label A text string to display in the combo box list.

data The value to associate with the list item. This parameter is optional.

Returns

Nothing.

Description

Method; adds a new item with the specified label and data to the end of the combo box list and updates the list. The *data* can be any Flash object, string, Boolean value, integer, object, or movie clip.

For best performance and load-time results, do not add more than 400 items in a single frame. This applies whether you are adding the items to a single combo box list or to several combo box lists.

Example

The following code adds the item `Kenny` with an associated value of `Keen` to the end of the list in the combo box `teacherList`.

```
teacherList.addItem("Kenny", Keen);
```

The following code adds the maximum number of items recommended in a single frame (400 items) to `comboBox1`:

```
for (i=0; i<400; i++) {
    comboBox1.addItem(i);
}
```

The following code adds the maximum number of items recommended in a single frame (400 items) to `listBox1` and `comboBox2`:

```
for (i=0; i<200; i++) {
    listBox1.addItem(i);
    comboBox2.addItem(i);
}
```

See also

`FComboBox.addItemAt`, `FComboBox.getItemAt`, `FComboBox.replaceItemAt`,
`FComboBox.setDataProvider`, `FComboBox.sortItemsBy`

FComboBox.addItemAt

Availability

Flash Player 6.

Usage

```
myComboBox.addItemAt(index, label [,data])
```

Parameters

index An integer specifying the position at which to insert the item.

label A string identifying the list item in the combo box.

data The value to associate with the list item. This parameter is optional.

Returns

Nothing.

Description

Method; adds a new item with the specified label and optional associated data to the combo box list at the specified index position. The Data parameter can be any Flash object, string, Boolean value, integer, object, or movie clip. As each item is added, the list is updated and the scroll bar is resized.

The ComboBox component uses a zero-based index, where the item at index 0 is displayed at the top of the list.

For best performance and load-time results, do not add more than 400 items in a single frame. This applies whether you are adding the items to a single combo box list or to several combo box lists.

Example

The following code adds the item `Justin` with the associated value `Ace` as the fifth item in the list of the combo box `Favorites`.

```
Favorites.addItemAt(4, "Justin", Ace);
```

For examples of loading a large number of items, see `FComboBox.addItem`.

See also

`FComboBox.getItemAt`, `FComboBox.removeItemAt`, `FComboBox.replaceItemAt`,
`FComboBox.setDataProvider`, `FComboBox.sortItemsBy`

FComboBox.setEnabled

Availability

Flash Player 6.

Usage

```
myComboBox.setEnabled()
```

Parameters

None.

Returns

A Boolean value indicating whether the combo box is enabled (`true`) or disabled (`false`).

Description

Method; indicates whether the combo box is enabled.

Example

The following code uses `getEnabled` to determine whether `comboBox1` is enabled or disabled and displays the result in the Output window.

```
trace(comboBox1.getEnabled());
```

See also

`FComboBox.setEnabled`

FComboBox.getItemAt

Availability

Flash Player 6.

Usage

```
myComboBox.getItemAt(index)
```

Parameters

index An integer specifying the position of an item in the combo box.

Returns

An object.

Description

Method; returns the item at the specified index as an object with the properties `label` and `data`.

The `ComboBox` component uses a zero-based index, where the item at index 0 is displayed at the top of the list.

Example

The following code returns the label of the item at index 4 in `comboBox1` as a string.

```
trace(comboBox1.getItemAt(4).label);
```

The following code returns the data associated with the item at index 4 in `comboBox2`. The return value depends on the type of data, and may be an object, string, movie clip reference, or other value.

```
trace(comboBox2.getItemAt(4).data);
```

The following code returns an object containing the label and the data value associated with the item at index 4 in `comboBox3`.

```
trace(comboBox3.getItemAt(4));
```

See also

`FComboBox.getSelectedItem`

FComboBox.getLength

Availability

Flash Player 6.

Usage

```
myComboBox.getLength()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the number of items in the combo box list.

Example

The following code retrieves the number of items in the list of `listMain` and stores it in the variable `len`.

```
var len = listMain.getLength();
```

See also

`FComboBox.addItem`, `FComboBox.addItemAt`

FComboBox.getRowCount

Availability

Flash Player 6.

Usage

```
myComboBox.getRowCount()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the number of rows visible in the combo box.

Example

The following code returns the number of rows visible in `toyList` and sets the value to the variable `rowCount`.

```
var rowCount = toyList.getRowCount();
```

See also

`FComboBox.setRowCount`

FComboBox.getScrollPosition

Availability

Flash Player 6.

Usage

```
myComboBox.getScrollPosition()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the index position of the item currently displayed at the top of the combo box list.

The ComboBox component uses a zero-based index, where the item at index 0 is displayed at the top of the list.

Example

The following code retrieves the index of the item currently at the top of the list in `toyList` and stores it in the variable `scrollPos`.

```
var scrollPos = toyList.getScrollPosition();
```

See also

`FComboBox.setSelectedIndex`

FComboBox.getSelectedIndex

Availability

Flash Player 6.

Usage

```
myComboBox.getSelectedIndex()
```

Parameters

None.

Returns

An integer or undefined.

Description

Method; returns the index of the item currently selected in the combo box, or returns `undefined` if no item is selected.

Items are listed in the combo box from top to bottom using a zero-based index.

Example

The following code retrieves the index of the item currently selected in `toyList` and stores it in the variable `selectedIndex`.

```
var selectedIndex = toyList.getSelectedIndex();
```

See also

`FComboBox.setSelectedIndex`

FComboBox.getSelectedItem

Availability

Flash Player 6.

Usage

```
myComboBox.getItem()
```

Parameters

None.

Returns

An object or undefined.

Description

Method; returns the currently selected item as an object with the properties `label` and `data`, or returns undefined if no item is selected.

Example

The following code retrieves the label and data of the item currently selected in `comboBox1`.

```
trace(comboBox1.getItem());
```

The following code retrieves the label of the item currently selected in `comboBox2`.

```
trace(comboBox2.getItem().label);
```

The following code retrieves the data of the item currently selected in `comboBox3`.

```
trace(comboBox3.getItem().data);
```

See also

`FComboBox.setSelectedIndex`

FComboBox.getValue

Availability

Flash Player 6.

Usage

```
myComboBox.getValue()
```

Parameters

None.

Returns

A text string.

Description

Method; returns the text in the field at the top of the combo box, if the combo box is editable. If the combo box is static (not editable), this method returns the data associated with the selected item, or the label of the item if no data is associated.

Example

The following code returns the data or label of the currently selected item in `menuMain`.

```
trace(menuMain.getValue())
```

See also

`FComboBox.setValue`

FComboBox.registerSkinElement

Availability

Flash Player 6.

Usage

```
myComboBox.registerSkinElement(element, styleProperty)
```

Parameters

element A movie clip instance.

styleProperty The name of an `FStyleFormat` property.

Returns

Nothing.

Description

Method; registers a skin element to a style property. Skin elements are registered to properties in the first frame of the Read Me layer of each skin the library.

Components are made up of skins, and each skin is composed of several skin elements, each of which can be registered to a property of the `FStyleFormat` object. These properties are assigned values by the style format assigned to a component. By default, all Flash UI components are assigned the `globalStyleFormat` object, which is an instance of the `FStyleFormat` object.

Use this method to register custom skin elements and properties to Flash UI or custom component skins by editing the code in the first frame of the Read Me layer of a skin in the library.

The `FComboBox` component uses the skins in the `FComboBox Skins` folder once you've added the component to the Flash document.

For more information, see "Customizing component skins" in the "Using Components" chapter of *Using Flash*.

Example

The following code registers the custom skin element `boundBox_mc` to the `background` property in the first frame of the Read Me layer of the `FBoundingBox` skin in the Global Skins folder.

```
toysMenu.registerSkinElement(boundBox_mc, "background");
```

FComboBox.removeAll

Availability

Flash Player 6.

Usage

```
myComboBox.removeAll()
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all the items in the combo box list, updates it, and resizes the scroll bar. Combo boxes without items are displayed without a scroll bar. This method cannot be used if the combo box is disabled.

Example

The following code removes all the items from menuMain.

```
menuMain.removeAll();
```

See also

FComboBox.removeItemAt

FComboBox.removeItemAt

Availability

Flash Player 6.

Usage

```
myComboBox.removeItemAt(index)
```

Parameters

index An integer specifying the index of the item to remove.

Returns

An object that contains the removed item.

Description

Method; returns the item removed at the specified index and updates the list. When an item is removed from the list, the indexes of the subsequent items are updated to reflect their new positions in the list. If no item exists at the specified index, this method returns *undefined*.

The ComboBox component uses a zero-based index, where the item at index 0 is displayed at the top of the list.

Example

The following code removes the fifth item from the list in menuMain.

```
menuMain.removeItemAt(4);
```

See also

FComboBox.removeAll

FComboBox.replaceItemAt

Availability

Flash Player 6.

Usage

```
myComboBox.replaceItemAt(index, label [,data])
```

Parameters

index An integer specifying the position of a list item.

label A string specifying a new label for the list item.

data The new value to associate with the list item. This parameter is optional; if you don't specify it, any data currently specified for the item remains in place.

Returns

Nothing.

Description

Method; updates the item at the specified index with the specified label and data. If the item at the specified index has an associated data value and you do not specify a value for the *data* parameter, the data value of the list item is not changed.

The ComboBox component uses a zero-based index, where the item at index 0 is displayed at the top of the list.

Example

The following code updates the fifth item in the combo box `Favorites` with the label `Nigel` and data value `7439`. If no data was specified for the list item, `Nigel` assumes the data value of the existing list item.

```
Favorites.replaceItemAt(4, "Nigel", "7439");
```

See also

`FComboBox.addItemAt`, `FComboBox.getItemAt`, `FComboBox.setDataProvider`,
`FComboBox.sortItemsBy`

FComboBox.setChangeHandler

Availability

Flash Player 6.

Usage

```
myComboBox.setChangeHandler(functionName, [location])
```

Parameters

functionName A string specifying the name of the handler function to execute when the selection in the combo box changes. If the *location* parameter is not specified, this function must be in the same Timeline as the component instance.

location A path reference to a data object, movie clip, or Timeline that contains the specified function. This parameter is optional and defaults to the parent Timeline of the component.

Returns

Nothing.

Description

Method; specifies a change handler to call when the selection in the combo box changes. You can specify the same change handler function for more than one component; the function always accepts the instance of the component that has changed as a parameter. Calling this method overrides the Change Handler parameter value specified in authoring.

For more information, see “Writing change handler functions for components” in the “Using Components” chapter of *Using Flash*.

Example

The following code specifies `myHandler` as the function called when the value of `toyList` changes. Because the `location` parameter is not specified, `myHandler` must be in the same Timeline as the component instance.

The component parameter in `myHandler` is automatically filled in with the instance of a component (the component that has changed as the result of user input and that specifies `myHandler` as its change handler). The actions defined in `myHandler` specify that when the user selects an item in the list, the label of the item is written to the Output window.

```
toyList.setChangeHandler("myHandler");
function myHandler(component){
    trace(toyList.getSelectedItem().label);
}
```

If in the preceding example `myHandler` is a function located in the great-grandparent Timeline of the component's Timeline, the first line of code would be as follows:

```
toyList.setChangeHandler("myHandler", _parent._parent._parent);
```

The following code creates the function `myHandler` in an instance of `myObject` (which is of class `Object`), and then specifies `myHandler` as the function for `toyList`.

```
myObject = new Object();
myObject.myHandler = function(component){
    trace(toyList.getSelectedItem().label);
}

toyList.setChangeHandler("myHandler", myObject);
```

FComboBox.setDataProvider

Availability

Flash Player 6.

Usage

```
myComboBox.setDataProvider(dataProvider)
```

Parameters

dataProvider An array of text strings listing the items to add, an instance of the Array object specifying the items to add, or an instance of the DataProvider class.

Returns

Nothing.

Description

Method; registers an outside object (*dataProvider*) as the data source for the combo box component. If *dataProvider* is an instance of the Array object, the object can specify *label*, *data*, or both, because object properties and the contents of the array can be copied to the combo box as labels, data, or both. If *dataProvider* is an instance of the DataProvider class, it must implement the DataProvider API defined in the DataProvider symbol in the FlashUIComponents/Core Assets/ClassTree folder in the library. The DataProvider API is for advanced users and programmers only; all other users should use an array or an Array object.

Example

The following code specifies the Array object *peopleList1* as the data provider for *comboBox1*.

```
comboBox1.setDataProvider(peopleList1);
```

The following code creates the array *peopleList* to display the labels of the items listed in *comboBox1*.

```
peopleList = new Array();
peopleList[0] = "BHall";
peopleList[1] = "CMoock";
peopleList[2] = "MWobensmith";
peopleList[3] = "MShepherd";
```

The following code creates the array *itemList1*, which specifies both the label and the data for list items. This array object could be used as an alternate data provider for *comboBox1*.

```
itemList1 = new Array();
for (i=0; i<10; i++) {

    // create a real item
    var myItem = new Object();
    myItem.label = "Item" + i;
    myItem.data = 75;

    // put it in the array
    itemList1[i] = myItem;
}
```

The following code specifies *comboData*, an instance of the DataProvider class, as the data provider for *comboBox1*.

```
comboBox1.setDataProvider(comboData);
```

The following code creates a new instance of the DataProvider class and then adds the item labels using the DataProvider *addItem* method.

Note: The *addItem* method is just one method of the DataProvider class. Programmers interested in using the DataProvider class should refer to the DataProvider symbol in the FlashUIComponents/CoreAssets/ClassTree folder in the library before attempting to use the methods.

```
comboData = new DataProviderClass();

comboData.addItem("Devra");
comboData.addItem("Delia");
comboData.addItem("Vashti");
comboData.addItem("Alicia");
```

See also

FComboBox.addItem, FComboBox.replaceItemAt, FComboBox.sortItemsBy

FComboBox.setEditable

Availability

Flash Player 6.

Usage

```
myComboBox.setEditable(editable)
```

Parameters

editable A Boolean value specifying whether the combo box is editable (`true`) or static (`false`).

Returns

Nothing.

Description

Method; determines whether the combo box is editable (`true`) or static (`false`). An editable combo box has a text field; when the user enters text, the combo box scrolls to the item with the same text. The text field can also be used to display text using `FComboBox.setValue`. Calling this method overrides the Editable parameter value set during authoring.

Example

The following code enables an input text field at the top of `menuMain`.

```
menuMain.setEditable(true);
```

See also

`FComboBox.setValue`

FComboBox.setEnabled

Availability

Flash Player 6.

Usage

```
myComboBox.setEnabled(enable)
```

Parameters

enable A Boolean value specifying whether the combo box is enabled (`true`) or disabled (`false`).

Returns

Nothing.

Description

Method; determines whether the combo box is enabled (`true`) or disabled (`false`). If the combo box is disabled, it does not accept mouse or keyboard interaction from the user. If you omit the parameter, this method defaults to `true`.

Example

The following code disables `menuMain`.

```
menuMain.setEnabled(false);
```

See also

`FListBox.setEnabled`

FComboBox.setItemSymbol

Availability

Flash Player 6.

Usage

```
myComboBox.setItemSymbol(symbolID)
```

Parameters

symbolID The symbol linkage ID of a graphic symbol to display the contents of the combo box.

Returns

Nothing.

Description

Method; registers a graphic symbol to display the combo box list items. The default value is the FComboBoxItem symbol in the library. This method is intended for advanced users and programmers.

FComboBox.setRowCount

Availability

Flash Player 6.

Usage

```
myComboBox.setRowCount(rows)
```

Parameters

rows The maximum number of rows that the drop-down list can display without scrolling.

Returns

Nothing.

Description

Method; sets the number of items that can be seen in the combo box's drop-down list without scrolling. The minimum value for the *rows* parameter is 3. Calling this method overrides the Row Count parameter value set during authoring.

Example

The following code sets the number of items displayed in the drop-down list of `menuMain` to 4.

```
menuMain.setRowCount(4);
```

See also

FComboBox.setSize

FComboBox.setSelectedIndex

Availability

Flash Player 6.

Usage

```
myComboBox.setSelectedIndex(index)
```

Parameters

index An integer specifying the index of the item to select.

Returns

Nothing.

Description

Method; selects the specified item and updates the combo box to show the item as selected. Calling this method does not affect the current open or closed state of the drop-down list. This method cannot be used if the combo box is disabled.

The ComboBox component uses a zero-based index, where the item at index 0 is displayed at the top of the list.

Example

The following code selects the fourth item in the list of `menuMain`.

```
menuMain.setSelectedIndex(3);
```

See also

`FComboBox.setRowCount`

FComboBox.setSize

Availability

Flash Player 6.

Usage

```
myComboBox.setSize(width)
```

Parameters

width An integer specifying the width of the combo box, in pixels.

Returns

Nothing.

Description

Method; resizes the combo box to the specified width. (You cannot set the height of a combo box component.) Use this method to programmatically resize the combo box and update it at runtime.

Example

The following code sets or resizes the width of `menuMain` to 100 pixels.

```
menuMain.setSize(100);
```

See also

`FComboBox.setRowCount`

FComboBox.setStyleProperty

Availability

Flash Player 6.

Usage

```
myComboBox.setStyleProperty(styleProperty, value)
```

Parameters

styleProperty A string specifying a property of the FStyleFormat object.

value The value to set for the property.

Returns

Nothing.

Description

Method; sets an FStyleFormat property for an individual combo box instance. Calling this method to specify a property overrides the setting for this property in the style format assigned to the component. Passing `undefined` as the value for a property removes all styles for that property.

To set FStyleFormat properties for multiple components, create a custom style format. For more information on creating custom style formats, see “Customizing component colors and text” in the “Using Components” chapter of *Using Flash*.

Example

The following code sets the `arrow` property of `comboBox1` to `0x000000` (black).

```
comboBox1.setStyleProperty("arrow", 0x000000);
```

See also

FStyleFormat (object)

FComboBox.setValue

Availability

Flash Player 6.

Usage

```
myComboBox.setValue(editableText)
```

Parameters

editableText A string specifying the text to appear in the text field of an editable combo box.

Returns

Nothing.

Description

Method; specifies the text displayed in the input field at the top of an editable combo box. If you call this method, the user can still input text in the field.

This method can be used only with editable combo boxes. Before calling this method, you must specify `true` for the `Editable` parameter (which defaults to `false`) during authoring or use `FComboBox.setEditable` to set the parameter to `true`.

Example

The following code enters the string `Gabino` in the top field of the combo box `surnameMenu`.

```
surnameMenu.setValue("Gabino");
```

See also

`FComboBox.getValue`

FComboBox.sortItemsBy

Availability

Flash Player 6.

Usage

```
myComboBox.sortItemsBy(fieldName, order)
```

Parameters

fieldName A string specifying the name of the field used for sorting. This will normally be "label" or "data".

order A string specifying whether to sort the items in ascending order ("ASC") or descending order ("DESC").

Returns

Nothing.

Description

Method; sorts the items in the combo box alphabetically or numerically, in the specified order, using the specified field name. If the *fieldName* items are a combination of text strings and integers, the integer items are listed first. The *fieldName* parameter is usually `label` or `data`, but you can specify any primitive data value that suits their needs.

Example

The following code sorts the items in the combo box `surnameMenu` in ascending order using the labels of the list items.

```
surnameMenu.sortItemsBy("label", "ASC");
```

See also

`FComboBox.addItemAt`, `FComboBox.replaceItemAt`, `FComboBox.setDataProvider`

FListBox (component)

The `ListBox` component in the Flash authoring environment provides drag-and-drop functionality for adding scrollable single-selection and multiple-selection list boxes to Flash documents; it also provides a user interface for setting basic parameters. The methods of the `FListBox` component allow you to control list boxes at runtime: you can create list boxes, control list boxes created in authoring, set or override basic parameters, and set additional runtime options. You do not need to use a constructor to access the methods of components.

Component methods do not perform error checking for type, as do other native `ActionScript` objects and actions; therefore, it is recommended that you validate parameters before passing them to methods.

The `ListBox` component is supported by Flash Player 6 and later versions of the Flash Player.

For information on using the `ListBox` component, setting parameters during authoring, and changing the color and appearance of components, see “Customizing component colors and text” and “Customizing component skins” in the “Using Components” chapter of *Using Flash*.

Method summary for the `FListBox` component

Method	Description
<code>FListBox.addItem</code>	Adds a new item to the end of the list box.
<code>FListBox.addItemAt</code>	Adds a new item to the list box at the specified index.
<code>FListBox.setEnabled</code>	Returns <code>true</code> if the list box is enabled, <code>false</code> if it is disabled.
<code>FListBox.getItemAt</code>	Returns the label and value of the item at the specified index.
<code>FListBox.getLength</code>	Returns the number of items in the list box.
<code>FListBox.getRowCount</code>	Returns the number of items visible in the list box.
<code>FListBox.getScrollPosition</code>	Returns the index of the item at the top of the list box.
<code>FListBox.getSelectedIndex</code>	Returns the index of the last selected item.
<code>FListBox.getSelectedIndices</code>	Returns the indexes of multiple selected items.
<code>FListBox.getSelectedItem</code>	Returns the label and value of the selected item.
<code>FListBox.getSelectedItems</code>	Returns the label and value of multiple selected items.
<code>FListBox.getSelectMultiple</code>	Returns <code>true</code> if multiple selection is allowed, <code>false</code> if single selection is allowed.
<code>FListBox.getValue</code>	Returns the label of the selected item or any associated data information.
<code>FListBox.registerSkinElement</code>	Registers a skin element to a property.
<code>FListBox.removeAll</code>	Removes all items from the list box.
<code>FListBox.removeItemAt</code>	Removes the item at the specified index.
<code>FListBox.replaceItemAt</code>	Replaces the label and data of an item at a specified index with a new label and data.
<code>FListBox.setAutoHideScrollBar</code>	Determines whether a scroll bar is hidden (<code>true</code>) or displayed (<code>false</code>) when the number of items in the list box does not require scrolling.
<code>FListBox.setChangeHandler</code>	Assigns a function to call each time the selection changes.
<code>FListBox.setDataProvider</code>	Associates an outside object with the list box.
<code>FListBox.setEnabled</code>	Specifies whether the list box is enabled (<code>true</code>) or disabled (<code>false</code>).
<code>FListBox.setItemSymbol</code>	Registers a symbol linkage ID to use for displaying items in the list box.
<code>FListBox.setRowCount</code>	Determines the number of items displayed in the list box.
<code>FListBox.setScrollPosition</code>	Causes the list box to scroll so that the item at the specified index is displayed at the top of the list.
<code>FListBox.setSelectedIndex</code>	Selects the item at the specified index and updates the list box.
<code>FListBox.setSelectedIndices</code>	Selects the items at the specified indexes and updates the list box.
<code>FListBox.setSelectMultiple</code>	Determines whether the user can select more than one item in the list (<code>true</code>) or not (<code>false</code>).
<code>FListBox.setSize</code>	Sets the width and height of the list box, in pixels.

Method	Description
<code>FListBox.setStyleProperty</code>	Sets a single style property for a component.
<code>FListBox.setWidth</code>	Sets the width of the list box, in pixels.
<code>FListBox.sortItemsBy</code>	Sorts the items in the list box alphabetically or numerically using the label or the data.

FListBox.addItem

Availability

Flash Player 6.

Usage

```
myListBox.addItem(label [, data])
```

Parameters

label A text string specifying the item to add to the list.

data A value associated with the list item. This parameter is optional.

Returns

Nothing.

Description

Method; adds a new item with the specified label and data (optional) to the end of the list box, updates the list box, and resizes the scroll bar. The Data parameter can be any Flash object, string, Boolean value, integer, object, or movie clip.

For best performance and load-time results, you should not add more than 400 items in a single frame. This applies whether you are adding the items to a single list box or to several list boxes.

Example

The following code adds *Lyvia* to the list of items displayed in the list box *coolGirls*.

```
coolGirls.addItem("Lyvia");
```

The following code adds the maximum number of items recommended in a single frame (400 items) to *listBox1*:

```
for (i=0; i<400; i++) {
    listBox1.addItem(i);
}
```

The following code adds the maximum number of items recommended in a single frame (400 items) to *listBox1* and *comboBox2*:

```
for (i=0; i<200; i++) {
    listBox1.addItem(i);
    comboBox2.addItem(i);
}
```

See also

`FListBox.addItemAt`, `FListBox.getItemAt`, `FListBox.removeItemAt`,
`FListBox.replaceItemAt`, `FListBox.sortItemsBy`

FListBox.addItemAt

Availability

Flash Player 6.

Usage

```
myListBox.addItemAt(index, label [, data])
```

Parameters

index An integer specifying the index position at which to add the item.

label A text string specifying the label of the item.

data A value associated with the list item. This parameter is optional.

Returns

Nothing.

Description

Method; adds a new item with the specified label and associated data (optional) at the specified index and updates the list box. The Data parameter can be any Flash object, string, Boolean, integer, object, or movie clip.

The ListBox component uses a zero-based index, where the item at index 0 is displayed at the top of the list.

For best performance and load-time results, do not add more than 400 items in a single frame. This applies whether you are adding the items to a single list box or to several list boxes.

Example

The following code adds the item Dave with an associated value of friend as the fifth item in list box peopleList.

```
peopleList.addItemAt(4, "Dave", friend);
```

For examples of loading a large number of items, see FListBox.addItem.

See also

FListBox.getSelectedItem, FListBox.removeItemAt, FListBox.replaceItemAt, FListBox.sortItemsBy

FListBox.setEnabled

Availability

Flash Player 6.

Usage

```
myListBox.setEnabled()
```

Parameters

None.

Returns

A Boolean value indicating whether the list box is enabled (`true`) or disabled (`false`).

Description

Method; indicates whether the list box is enabled.

Example

The following code uses `getEnabled` to determine whether `listMenu` is enabled or disabled, and displays the result in the Output window.

```
trace(listMenu.getEnabled());
```

See also

`FListBox.setEnabled`

FListBox.getItemAt

Availability

Flash Player 6.

Usage

```
myListBox.getItemAt(index)
```

Parameters

index An integer specifying the index of the item to retrieve.

Returns

An object or undefined.

Description

Method; returns the item at the specified index as an object with the properties `label` and `data`. If there is no item at the specified index, the method returns `undefined`.

The `ListBox` component uses a zero-based index, where the item at index 0 is displayed at the top of the list.

Example

The following code returns the label of the item at index 4 in `listMenu1` to the Output window.

```
trace(listMenu1.getItemAt(4).label);
```

The following code returns the data or value associated with the item at index 4 in `listMenu2` to the Output window.

```
trace(listMenu2.getItemAt(4).data);
```

The following code returns an object containing the label and data value associated with the item at index 4 in `listMenu3` to the Output window.

```
trace(listMenu3.getItemAt(4));
```

See also

`FListBox.getSelectedItem`

FListBox.getLength

Availability

Flash Player 6.

Usage

```
myListBox.getLength()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the number of items in the list box.

Example

The following code returns the number of items in `phoneList`.

```
phoneList.getLength();
```

See also

`FListBox.setSize`

FListBox.getRowCount

Availability

Flash Player 6.

Usage

```
myListBox.getRowCount()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the number of rows visible in the list box. This method is useful for determining how many rows are displayed in a list box that is scaled in pixels.

Example

The following code returns the number of rows visible in `toyList` and sets the value to the variable `rowCount`.

```
var rowCount = toyList.getRowCount();
```

See also

`FListBox.setRowCount`, `FListBox.setSize`

FListBox.getScrollPosition

Availability

Flash Player 6.

Usage

```
myListBox.getScrollPosition()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the index of the item currently at the top of the list box view.

Example

The following code returns the index of the item at the top of `staffList`.

```
staffList.getScrollPosition();
```

See also

`FListBox.setScrollPosition`

FListBox.getSelectedIndex

Availability

Flash Player 5

Usage

```
myListBox.getSelectedIndex()
```

Parameters

None.

Returns

An integer or undefined.

Description

Method; returns the index of the currently selected item in a single-selection list box, the most recently selected item in a multiple-selection list box, or undefined if no item is selected. To retrieve the indexes of all selected items in a multiple-selection list box, use `FListBox.getSelectedIndices`.

Example

The following code returns the index of the currently selected item in the single-section list box `nationList`.

```
nationList.getSelectedIndex();
```

See also

`FListBox.setSelectedIndices`, `FListBox.setSelectMultiple`

FListBox.getSelectedIndices

Availability

Flash Player 6.

Usage

```
myListBox.getSelectedIndices()
```

Parameters

None.

Returns

An array or undefined.

Description

Method; returns the indexes of the currently selected items in a multiple-selection list box as an array, or returns undefined if no items are selected.

Example

The following code returns the indexes of the currently selected items in the multiple-selection list box `groceryList` as an array.

```
groceryList.getSelectedIndices();
```

See also

`FListBox.getSelectedIndex`, `FListBox.setSelectMultiple`

FListBox.getSelectedItem

Availability

Flash Player 6.

Usage

```
myListBox.getSelectedItem()
```

Parameters

None.

Returns

An object or undefined.

Description

Method; returns the currently selected item as an object with the properties `label` and `data`. If more than one item is selected, the method returns the most recently selected item in the list; if no items are selected, the method returns undefined. To retrieve information about all selected items in a multiple-selection list box, use `FListBox.getSelectedItems`.

Example

The following code returns the label of the item currently selected in `listBox1`.

```
trace(listBox1.getSelectedItem().label);
```

The following code returns the data or value associated with item currently selected in `listBox2`.

```
trace(listBox2.getSelectedItem().data);
```

The following code returns an object containing the label and the data value associated with the item currently selected in `listBox3`.

```
trace(listBox3.getSelectedItemAt());
```

See also

`FListBox.getItemAt`

FListBox.getSelectedItems

Availability

Flash Player 6.

Usage

```
myListBox.getSelectedItems()
```

Parameters

None.

Returns

An array or undefined.

Description

Method; returns the currently selected items as an array of objects with the properties `label` and `data`, or returns undefined if no items are selected. This method can be used only to get the selected items in a multiple-selection list box. To retrieve information about the currently selected item in a single-selection list box, use `FListBox.getSelectedItem`.

Example

The following code retrieves the items currently selected in `wishList` and stores them in the variable `myObjArray`.

```
var myObjArray = wishList.getSelectedItems();
```

See also

`FListBox.getSelectedItem`, `FListBox.setSelectMultiple`

FListBox.getSelectMultiple

Availability

Flash Player 6.

Usage

```
myListBox.getSelectMultiple()
```

Parameters

None.

Returns

A Boolean value.

Description

Method; indicates whether users can select multiple items (`true`) or only a single item (`false`) in the list box.

Example

The following code tests whether `wishList` permits multiple selection.

```
if (wishList.getSelectMultiple()) {  
}
```

See also

`FListBox.setSelectMultiple`

FListBox.getValue

Availability

Flash Player 6.

Usage

```
myListBox.getValue()
```

Parameters

None.

Returns

The label or data associated with a selected item.

Description

Method; returns information about the item currently selected in the list box. If the item does not have specified data, this method returns the item's label; if the item has data associated, this method returns the data.

Example

The following code returns the label of the item selected in `nationList`.

```
trace(nationList.getValue());
```

See also

`FListBox.getItemAt`

FListBox.registerSkinElement

Availability

Flash Player 6.

Usage

```
myListBox.registerSkinElement(element, styleProperty)
```

Parameters

element A movie clip instance.

styleProperty The name of an `FStyleFormat` property.

Returns

Nothing.

Description

Method; registers a skin element to a style property. Skin elements are registered to properties in the first frame of the Read Me layer of each skin the library.

Components are made up of skins, and each skin is composed of several skin elements, each of which can be registered to a property of the `FStyleFormat` object. These properties are assigned values by the style format assigned to a component. By default, all Flash UI components are assigned the `globalStyleFormat` object, which is an instance of the `FStyleFormat` object.

Use this method to register custom skin elements and properties to Flash UI or custom component skins by editing the code in the first frame of the Read Me layer of a skin in the library.

The `FListBox` component uses the skins in the `FListBox Skins` folder once you've added the component to the Flash document.

For more information, see “Customizing component skins” in the “Using Components” chapter of *Using Flash*.

Example

The following code registers the custom skin element `boundBox_mc` to the `background` property in the first frame of the Read Me layer of the `FBoundingBox` skin in the Global Skins folder in the library.

```
toysMenu.registerSkinElement(boundBox_mc, "background");
```

See also

`FStyleFormat` (object)

FListBox.removeAll

Availability

Flash Player 6.

Usage

```
myListBox.removeAll()
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all the items from the list box, updates it, and resizes the scroll bar.

Example

The following code removes all the items from `wishList`.

```
wishList.removeAll();
```

See also

`FListBox.removeItemAt`

FListBox.removeItemAt

Availability

Flash Player 6.

Usage

```
myListBox.removeItemAt(index)
```

Parameters

index An integer specifying the index of the item to remove.

Returns

Nothing or `undefined`.

Description

Method; removes the item at the specified index, updates the indexes of the list items following the removed item to reflect their new positions in the list, and then updates the list box and resizes the scroll bar. If no item exists at the specified index, this method returns `undefined`.

The ListBox component uses a zero-based index, where the item at index 0 is displayed at the top of the list.

Example

The following code removes the fifth item in the list from `wishList`.

```
wishList.removeItemAt(4);
```

See also

`FListBox.addItemAt`

FListBox.replaceItemAt

Availability

Flash Player 6.

Usage

```
myListBox.replaceItemAt(index, label [,data])
```

Parameters

index An integer specifying the position of a list item.

label A string specifying a new label for the list item.

data The new value to associate with the list item. This parameter is optional; if you don't specify it, any data currently associated with the item remains in place.

Returns

Nothing.

Description

Method; updates the item at the specified index with the specified label and data. If the item at the specified index has an associated data value and you do not specify a value for the Data parameter, the data value of the list item is not changed.

The List Box component uses a zero-based index, where the item at index 0 is displayed at the top of the list.

Example

The following code updates the fifth item in the list box `Favorites` with the new label `Lucky` and new value `Cat`. If the data value `Cat` were not specified, and the data associated with the fifth list item were `Dog`, then the data value for `Lucky` would be `Dog` (which would be wrong because she is a cat).

```
Favorites.replaceItemAt(4, "Lucky", "Cat");
```

See also

```
FListBox.addItemAt, FListBox.getItemAt
```

FListBox.setAutoHideScrollBar

Availability

Flash Player 6.

Usage

```
myListBox.setAutoHideScrollBar(hideScroll)
```

Parameters

hideScroll A Boolean value specifying whether the scroll bar is hidden when not needed (`true`) or always displayed (`false`).

Returns

Nothing.

Description

Method; specifies whether the scroll bar is hidden when the number of items in the list box can be viewed without a scroll bar (`true`) or whether the scroll bar is always displayed (`false`). If this method is set to `false` and the number of items does not require a scroll bar, the scroll bar is displayed as disabled (dimmed).

Example

The following code hides the scroll bar for `wishList` when the number of items in the list box can be viewed without scrolling.

```
wishList.setAutoHideScrollBar(true);
```

FListBox.setChangeHandler

Availability

Flash Player 6.

Usage

```
myListBox.setChangeHandler(functionName, [location])
```

Parameters

functionName A string specifying the name of the handler function to execute when the selection in the list box changes. If the *location* parameter is not specified, this function must be in the same Timeline as the component instance.

location A path reference to a data object, movie clip, or Timeline that contains the specified function. This parameter is optional and defaults to the parent Timeline of the component.

Returns

Nothing.

Description

Method; specifies a change handler to call when the selection in the list box changes. You can specify the same change handler function for more than component; the function always accepts the instance of the component that has changed as a parameter. Calling this method overrides the Change Handler parameter value specified in authoring.

For more information, see “Writing change handler functions for components” in the “Using Components” chapter of *Using Flash*.

Example

The following code specifies `myHandler` as the function called when the value of `listBox1` changes. Because the *location* parameter is not specified, `myHandler` must be in the same Timeline as the component instance.

The component parameter in `myHandler` is automatically filled in with the instance of a component (the component that has changed as the result of user input and that specifies `myHandler` as its change handler). The actions defined in `myHandler` specify that when the user selects an item in the list, the label of the item is written to the Output window.

```
listBox1.setChangeHandler("myHandler");
function myHandler(component){
    trace(listBox1.getSelectedItem().label);
}
```

If in the preceding example `myHandler` is a function located in the great-grandparent Timeline of the component's Timeline, the first line of code would be as follows:

```
listBox1.setChangeHandler("myHandler", _parent._parent._parent);
```

The following code creates the function `myHandler` in an instance of `myObject` (which is of class `Object`), and then specifies `myHandler` as the function for `listBox1`.

```
myObject = new Object();
myObject.myHandler = function(component){
    trace(listBox1.getSelectedItem().label);
}

listBox1.setChangeHandler("myHandler", myObject);
```

FListBox.setDataProvider

Availability

Flash Player 6.

Usage

```
myListBox.setDataProvider(dataProvider)
```

Parameters

dataProvider An array of text strings listing the items to add, an instance of the Array object specifying the items to add, or an instance of the DataProvider class.

Returns

Nothing.

Description

Method; registers an outside object (*dataProvider*) as the data source for the list box component. If *dataProvider* is an instance of the Array object, the object can specify `label`, `data`, or both, because object properties and the contents of the array can be copied to the list box as labels, data, or both. If *dataProvider* is an instance of the DataProvider class, it must implement the DataProvider API defined in the DataProvider symbol in the FlashUIComponents/CoreAssets/ClassTree folder in the library. The DataProvider API is for advanced users and programmers only; all other users should use an array or an Array object.

Example

The following code specifies the Array object `writerList` as the data provider for `listBox1`.

```
listBox1.setDataProvider(writerList);
```

The following code creates the array `writerList` to display the labels of the items listed in `listBox1`.

```
writerList = new Array();
writerList[0] = "Jody";
writerList[1] = "Mary";
writerList[2] = "Marcelle";
writerList[3] = "Dale";
writerList[4] = "Stephanie";
writerList[5] = "Barbara";
```

The following code creates the array `itemList1`, which specifies both the label and the data for list items. This Array object could be used as an alternate data provider for `listBox1`.

```
itemList1 = new Array();
for (i=0; i<10; i++) {

    // create a real item
    var myItem = new Object();
    myItem.label = "Item" + i;
    myItem.data = 75;

    // put it in the array
    itemList1[i] = myItem;
}
```

The following code specifies `editorList`, an instance of the DataProvider class, as the data provider for `listBox1`.

```
listBox1.setDataProvider(editorList);
```

The following code creates a new instance of the DataProvider class and then adds the item labels using the DataProvider `addItem` method.

Note: The `addItem` method is just one method of the DataProvider class. Programmers interested in using the DataProvider class should refer to the DataProvider symbol in the FlashUIComponents/CoreAssets/ClassTree folder in the library before attempting to use the methods.

```
editorList = new DataProviderClass();
editorList.addItem("Anne");
editorList.addItem("Rosana");
editorList.addItem("Lisa");
editorList.addItem("Rebecca");
```

See also

`FListBox.addItem`, `FListBox.replaceItemAt`, `FListBox.sortItemsBy`

FListBox.setEnabled

Availability

Flash Player 6.

Usage

```
myListBox.setEnabled(enable)
```

Parameters

enable A Boolean value specifying whether the list box is enabled (`true`) or disabled (`false`).

Returns

Nothing.

Description

Method; specifies whether the list box is enabled (`true`) or disabled (`false`). If a list box is disabled, it does not accept mouse or keyboard interaction from the user. If you omit the parameter, this method defaults to `true`.

Example

The following code disables `interestList`.

```
interestList.setEnabled(false);
```

See also

`FListBox.setEnabled`

FListBox.setItemSymbol

Availability

Flash Player 6.

Usage

```
myListBox.setItemSymbol(symbolID)
```

Parameters

symbolID The symbol linkage ID of a graphic symbol to display the contents of the list box.

Returns

Nothing.

Description

Method; registers a graphic symbol to display the items in the list box. The default value is the `FListBoxItem` symbol in the library. This method is intended for advanced users and programmers.

FListBox.setRowCount

Availability

Flash Player 6.

Usage

```
myListBox.setRowCount(rows)
```

Parameters

rows The maximum number of rows displayed in the list box.

Returns

Nothing.

Description

Method; specifies the number of items displayed in the list box. If you use this method, use `FListBox.setWidth`, not `FListBox.setSize`, to set the width of the list box. Calling `FListBox.setSize` overrides the Row Count parameter value set during authoring. Therefore, if you call this method after calling `FListBox.setRowCount`, your movie will disregard the `rowCount` setting and set the height of the list box in pixels.

Example

The following code sets the number of items displayed in `toyList` to 4.

```
toyList.setRowCount(4);
```

See also

`FListBox.getRowCount`, `FListBox.setSize`

FListBox.setScrollPosition

Availability

Flash Player 6.

Usage

```
myListBox.setScrollPosition(index)
```

Parameters

index An integer specifying the index of the item to display at the top of the list box.

Returns

Nothing.

Description

Method; causes the list box to scroll so that the specified item is displayed at the top.

The ListBox component uses a zero-based index, where the item at index 0 is displayed at the top of the list.

Example

The following code displays the fifth item in `toyList` at the top of the list.

```
toyList.setScrollPosition(4);
```

See also

`FListBox.getScrollPosition`

FListBox.setSelectedIndex

Availability

Flash Player 6.

Usage

```
myListBox.setSelectedIndex(index)
```

Parameters

index An integer specifying the index of the item to select in the list box.

Returns

Nothing.

Description

Method; selects the item at the specified index and updates the list box.

The ListBox component uses a zero-based index, where the item at index 0 is displayed at the top of the list.

Example

The following code selects the fifth item in the list box.

```
toyList.setSelectedIndex(4);
```

See also

FListBox.getSelectedIndex, FListBox.getSelectedIndices

FListBox.setSelectedIndices

Availability

Flash Player 6.

Usage

```
myListBox.setSelectedIndices(indexArray)
```

Parameters

indexArray An array of item indexes to select in the list box.

Returns

Nothing.

Description

Method; selects the items specified in the array of indexes and updates the list box.

The ListBox component uses a zero-based index, where the item at index 0 is displayed at the top of the list.

Example

The following code creates an array called `myArray` specifying the items to select in `toyList` and then passes the array to the `setSelectedIndices` method.

```
var myArray = new Array (1,4,5,7);  
toyList.setSelectedIndices(myArray);
```

See also

FListBox.getSelectedIndices

FListBox.setSelectMultiple

Availability

Flash Player 6.

Usage

```
myListBox.setSelectMultiple(multipleSelect)
```

Parameters

multipleSelect A Boolean value specifying multiple-selection mode (`true`) or single-selection mode (`false`).

Returns

Nothing.

Description

Method; specifies whether users are allowed to select multiple items (`true`) or only single items (`false`) in the list box. The default setting is `false`. Calling this method overrides the Select Multiple parameter value set during authoring.

Example

The following code enables multiple selection for `toyList`.

```
toyList.setSelectMultiple(true);
```

FListBox.setSize

Availability

Flash Player 6.

Usage

```
myListBox.setSize(width, height)
```

Parameters

width An integer specifying the width of the list box, in pixels.

height An integer specifying the height of the list box, in pixels.

Returns

Nothing.

Description

Method; resizes the list box at runtime to the specified width and height. Calling this method overrides the Row Count parameter value set during authoring. Therefore, if you call this method after calling `FListBox.setRowCount`, your movie will set the height of the list box in pixels and disregard the `rowCount` setting. To set the width of a list box when you are using `setRowCount`, use `FListBox.setWidth`.

Example

The following code resizes `phoneList` to measure 200 pixels in width and 50 pixels in height.

```
phoneList.setSize(200, 50);
```

See also

`FListBox.setRowCount`, `FListBox.setWidth`

FListBox.setStyleProperty

Availability

Flash Player 6.

Usage

```
myListBox.setStyleProperty(styleProperty, value)
```

Parameters

styleProperty A string specifying a property of the FStyleFormat object.

value The value to set for the property.

Returns

Nothing.

Description

Method; sets an FStyleFormat property for an individual list box instance. Calling this method to specify a property overrides the setting for this property in the style format assigned to the component. Passing *undefined* as the value for a property removes all styles for that property.

To set FStyleFormat properties for multiple components, create a custom style format. For more information on creating custom style formats, see “Customizing component colors and text” in the “Using Components” chapter of *Using Flash*.

Example

The following code sets the shadow property of `listBox1` to 0x000000 (black).

```
listBox1.setStyleProperty("shadow", 0x000000);
```

See also

FStyleFormat (object)

FListBox.setWidth

Availability

Flash Player 6.

Usage

```
myListBox.setWidth(width)
```

Parameters

width An integer specifying the width of the list box, in pixels.

Returns

Nothing.

Description

Method; specifies the width of the list box, in pixels. This method is useful for setting the width of list box when FListBox.setRowCount is used to determine the height.

Example

The following code sets the width of the list box `toyList` to 500 pixels.

```
toyList.setWidth(500);
```

See also

`FListBox.setSize`

FListBox.sortItemsBy

Availability

Flash Player 6.

Usage

```
myListBox.sortItemsBy(fieldName, order)
```

Parameters

fieldName A string specifying the name of the field used for sorting. This will normally be "label" or "data".

order A string specifying whether to sort the items in ascending order ("ASC") or descending order ("DESC").

Returns

Nothing.

Description

Method; sorts the items in the list box alphabetically or numerically, in the specified order, using the specified field name. If the *fieldName* items are a combination of text strings and integers, the integer items are listed first. The *fieldName* parameter is usually "label" or "data", but advanced users and programmers can specify any primitive that suits their needs.

Example

The following code sorts the items in the list box `surnameMenu` in ascending order using the labels of the list items.

```
surnameMenu.sortItemsBy("label", "ASC");
```

See also

`FListBox.addItemAt`, `FListBox.replaceItemAt`

_focusrect

Availability

Flash Player 4.

Usage

```
_focusrect = Boolean;
```

Description

Property (global); specifies whether a yellow rectangle appears around the button that has keyboard focus. The default value, `true`, displays a yellow rectangle around the currently focused button or text field as the user presses the Tab key to navigate through objects in a movie. Specify `false` if you do not want to display the yellow rectangle. This is a global property that can be overridden for specific instances.

See also

`Button._focusrect`

for

Availability

Flash Player 5.

Usage

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

Parameters

init An expression to evaluate before beginning the looping sequence, typically an assignment expression. A `var` statement is also permitted for this parameter.

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

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

statement(s) An instruction or instructions to execute within the body of the loop.

Description

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

Some properties cannot be enumerated by the `for` or `for...in` actions. For example, the built-in methods of the Array object (`Array.sort` and `Array.reverse`) are not included in the enumeration of an Array object, and movie clip properties, such as `_x` and `_y`, are not enumerated.

Example

The following example uses `for` to add the elements in an array:

```
for(i=0; i<10; i++) {  
    array[i] = (i + 5)*10;  
    trace(array[i]);  
}
```

The following results are displayed in the Output window:

```
50  
60  
70  
80  
90  
100  
110  
120  
130  
140
```

The following is an example of using `for` to perform the same action repeatedly. In the code below, the `for` loop adds the numbers from 1 to 100:

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

See also

`++` (increment), `--` (decrement), `for..in`, `var`

for..in

Availability

Flash Player 5.

Usage

```
for(variableIterant in object){  
    statement(s);  
}
```

Parameters

variableIterant The name of a variable to act as the iterant, referencing each property of an object or element in an array.

object The name of an object to be repeated.

statement(s) An instruction to execute for each iteration.

Returns

Nothing.

Description

Action; loops through the properties of an object or element in an array, and executes the *statement* for each property of an object.

Some properties cannot be enumerated by the `for` or `for..in` actions. For example, the built-in methods of the Array object (`Array.sort` and `Array.reverse`) are not included in the enumeration of an Array object, and movie clip properties, such as `_x` and `_y`, are not enumerated.

The `for..in` construct iterates over properties of objects in the iterated object's prototype chain. If the child's prototype is `parent`, iterating over the properties of the child with `for..in`, will also iterate over the properties of `parent`.

The `for..in` action enumerates all objects in the prototype chain of an object. Properties of the object are enumerated first, then properties of its immediate prototype, then properties of the prototype's prototype, and so on. The `for..in` action does not enumerate the same property name twice. If the object `child` has prototype `parent` and both contain the property `prop`, the `for..in` action called on `child` enumerates `prop` from `child` but ignores the one in `parent`.

Example

The following is an example of using `for..in` to iterate over the properties of an object:

```
myObject = { name:'Tara', age:27, city:'San Francisco' };
for (name in myObject) {
    trace ("myObject." + name + " = " + myObject[name]);
}
```

The output of this example is as follows:

```
myObject.name = Tara
myObject.age = 27
myObject.city = San Francisco
```

The following is an example of using the `typeof` operator with `for..in` to iterate over a particular type of child:

```
for (name in myMovieClip) {
    if (typeof (myMovieClip[name]) = "movieclip") {
        trace ("I have a movie clip child named " + name);
    }
}
```

The following example enumerates the children of a movie clip and sends each to frame 2 in their respective Timelines. The `RadioButtonGroup` movie clip is a parent with several children, `_RedRadioButton`, `_GreenRadioButton` and `_BlueRadioButton`.

```
for (var name in RadioButtonGroup) {
    RadioButtonGroup[name].gotoAndStop(2);
}
```

FPushButton (component)

The PushButton component in the Flash authoring environment provides drag-and-drop functionality for adding buttons to Flash documents; it also provides a user interface for setting basic parameters. The methods of the FPushButton component allow you to control buttons at runtime: you can create buttons, control buttons created in authoring, set or override basic parameters, and set additional runtime options. You do not need to use a constructor to access the methods of components.

The PushButton component accepts all standard mouse and keyboard interactions. You can use the FPushButton methods to specify a handler function for push buttons, disable or enable buttons, and resize buttons without distortion at runtime.

Component methods do not perform error checking for type, as do other native ActionScript objects and actions; therefore, it is recommended that you validate parameters before passing them to methods.

The PushButton component is supported by Flash Player 6 and later versions of the Flash Player.

For information on using the PushButton component, setting parameters during authoring, and changing the color and appearance of components, see “Customizing component colors and text” and “Customizing component skins” in the “Using Components” chapter of *Using Flash*.

Method summary for the FPushButton component

Method	Description
<code>FPushButton.setEnabled</code>	Returns <code>true</code> if the button is enabled, <code>false</code> if it is disabled.
<code>FPushButton.getLabel</code>	Returns the button label as a string.
<code>FPushButton.registerSkinElement</code>	Registers a skin element to a property.
<code>FPushButton.setClickHandler</code>	Specifies the function called when the user releases the button.
<code>FPushButton.setEnabled</code>	Determines whether the button is enabled or disabled.
<code>FPushButton.setLabel</code>	Sets the button label at runtime.
<code>FPushButton.setSize</code>	Sets the height and width of the button, in pixels.
<code>FPushButton.setStyleProperty</code>	Sets a single style property for a component.

FPushButton.setEnabled

Availability

Flash Player 6.

Usage

```
myPushButton.setEnabled()
```

Parameters

None.

Returns

A Boolean value.

Description

Method; returns `true` if the push button instance is enabled, `false` if it is disabled.

Example

The following code returns the enabled state of the push button `submit` to the Output window.

```
trace(submit.setEnabled());
```

See also

`FPushButton.setEnabled`

FPushButton.getLabel

Availability

Flash Player 6.

Usage

```
myPushButton.getLabel()
```

Parameters

None.

Returns

A string.

Description

Method; returns the text label on the push button as a string.

Example

The following code returns the label for the push button `buttonPage1` to the Output window.

```
trace(buttonPage1.getLabel());
```

See also

`FPushButton.setLabel`

FPushButton.registerSkinElement

Availability

Flash Player 6.

Usage

```
myPushButton.registerSkinElement(element, styleProperty)
```

Parameters

element A movie clip instance.

styleProperty The name of an `FStyleFormat` property.

Returns

Nothing.

Description

Method; registers a skin element to a style property. Skin elements are registered to properties in the first frame of the Read Me layer of each skin the library.

Components are made up of skins, and each skin is composed of several skin elements, each of which can be registered to a property of the `FStyleFormat` object. These properties are assigned values by the style format assigned to a component. By default, all Flash UI components are assigned the `globalStyleFormat` object, which is an instance of the `FStyleFormat` object.

Use this method to register custom skin elements and properties to Flash UI or custom component skins by editing the code in the first frame of the Read Me layer of a skin in the library.

The `FPushButton` component uses the skins in the `FPushButton Skins` folder and the `FLabel` skin in the `Global Skins` folder once you've added the component to the Flash document.

For more information, see “Customizing component skins” in the “Using Components” chapter of *Using Flash*.

Example

The following code registers the custom skin element `newFace_mc` to the `face` property in the first frame of the Read Me layer of the `FLabel` skin. The `FLabel` skin is in the `Component Skins/Global Skins` folder in the library.

```
submitButton.registerSkinElement(newFace_mc, "face");
```

See also

`FStyleFormat` (object)

FPushButton.setClickHandler

Availability

Flash Player 6.

Usage

```
myPushButton.setClickHandler(functionName, [location])
```

Parameters

functionName A string specifying the name of the handler function to execute when the user releases the push button. If the *location* parameter is not specified, this function must be in the same Timeline as the component instance.

location A path reference to a data object, movie clip, or Timeline that contains the specified function. This parameter is optional and defaults to the parent Timeline of the component.

Returns

Nothing.

Description

Method; specifies a handler function to call when the user releases the push button. You can specify the same handler function for more than one component; the function always accepts the instance of the component that has changed as a parameter. Calling this method overrides the Click Handler parameter value specified in authoring.

For more information, see “Writing change handler functions for components” in the “Using Components” chapter of *Using Flash*.

Example

The following code specifies `onClick` as the function called when the value of `button1` changes. Because the *location* parameter is not specified, `onClick` must be in the same Timeline as the component instance. The `component` parameter in `onClick` is automatically filled in with the instance of a component (the component that has changed as the result of user input and that specifies `onClick` as its change handler). The actions defined in `onClick` specify that when the user releases a button, the label of the button is written to the Output window.

```
button1.setClickHandler("onClick");
```

```
function onClick(component){  
    trace(component._name);  
}
```

If in the preceding example `onClick` is a function located in the great-grandparent Timeline of the component's Timeline, the first line of code would be as follows:

```
button1.setChangeHandler("onClick", _parent._parent._parent);
```

The following code creates the function `onClick` in an instance of `myObject` (which is of class `Object`), and then specifies `onClick` as the function for `button1`.

```
myObject = new Object();  
myObject.onClick = function(component){  
    trace(component._name);  
}  
  
button1.setChangeHandler("onClick", myObject);
```

FPushButton.setEnabled

Availability

Flash Player 6.

Usage

```
myPushButton.setEnabled(enable)
```

Parameters

enable A Boolean value specifying whether the push button is enabled (`true`) or disabled (`false`).

Returns

Nothing.

Description

Method; determines whether the push button is enabled. If a push button is disabled, it does not accept mouse or keyboard interaction from the user, and the text on the button is dimmed. Omitting the parameter is the same as passing `true`.

Example

The following code disables `button1`.

```
button1.setEnabled(false);
```

See also

`FPushButton.setEnabled`

FPushButton.setLabel

Availability

Flash Player 6.

Usage

```
myPushButton.setLabel(label)
```

Parameters

label A string containing the text to appear on the push button.

Returns

Nothing.

Description

Method; applies a text label to the push button at runtime. Calling this method overrides the *label* parameter value specified in authoring.

Example

The following code applies the label `Cleveland Rocks!` to `voteButton`.

```
voteButton.setLabel("Cleveland Rocks!");
```

See also

`FPushButton.getLabel`

FPushButton.setSize

Availability

Flash Player 6.

Usage

```
myPushButton.setSize(width, height)
```

Parameters

width An integer specifying the width of the push button, in pixels.

height An integer specifying the height of the push button, in pixels.

Returns

Nothing.

Description

Method; sets the width and height of the push button at runtime. Calling this method overrides any sizing or scaling applied during authoring. For more information, see “Sizing PushButton components” of the “Using Components” chapter in *Using Flash*.

Example

The following code resizes `submitButton` to 100 x 50 pixels at runtime.

```
submitButton.setSize(100, 50);
```


FPushButton.setStyleProperty

Availability

Flash Player 6.

Usage

```
myPushButton.setStyleProperty(styleProperty, value)
```

Parameters

styleProperty A string specifying a property of the FStyleFormat object.

value The value to set for the property.

Returns

Nothing.

Description

Method; sets an FStyleFormat property for an individual push button instance. Calling this method to specify a property overrides the setting for this property in the style format assigned to the component. Passing *undefined* as the value for a property removes all styles for that property.

To set FStyleFormat properties for multiple components, create a custom style format. For more information on creating custom style formats, see “Customizing component colors and text” in the “Using Components” chapter of *Using Flash*.

Example

The following code sets the *face* property of *submitButton* to 0xffffffff (white).

```
submitButton.setStyleProperty("face", 0xffffffff);
```

See also

FStyleFormat (object)

FRadioButton (component)

Radio buttons are groups of selectable buttons from which only one button can be selected at a time. The RadioButton component in the Flash authoring environment provides drag-and-drop functionality for adding groups of radio buttons to Flash documents; it also provides a user interface for setting basic parameters. The methods of the FRadioButton component allow you to control radio buttons at runtime: you can create radio buttons, control radio buttons created in authoring, set or override basic parameters, and set additional runtime options. You do not need to use a constructor to access the methods of components.

The RadioButton component is supported by Flash Player 6 and later versions of the Flash Player.

Component methods do not perform error checking for type, as do other native ActionScript objects and actions; therefore, it is recommended that you validate parameters before passing them to methods.

For information on using the RadioButton component, setting parameters during authoring, and changing the color and appearance of components, see “Customizing component colors and text” and “Customizing component skins” in the “Using Components” chapter of *Using Flash*.

Method summary for the FRadioButton component

Method	Description
<code>FRadioButton.getData</code>	Returns a data value for a radio button instance.
<code>FRadioButton.setEnabled</code>	Returns <code>true</code> if the radio button is enabled, <code>false</code> if it is disabled.
<code>FRadioButton.getLabel</code>	Returns the radio button label as a string.
<code>FRadioButton.getState</code>	Returns the selected state of a radio button instance.
<code>FRadioButton.getValue</code>	Returns the data value of the selected radio button in a group, or returns the label if no data is specified.
<code>FRadioButton.registerSkinElement</code>	Registers a skin element to a style property.
<code>FRadioButton.setChangeHandler</code>	Specifies a function to call when the radio button selection changes.
<code>FRadioButton.setData</code>	Sets the data associated with a radio button instance.
<code>FRadioButton.setEnabled</code>	Determines whether the radio button is enabled or disabled.
<code>FRadioButton.setGroupName</code>	Specifies a group name for a radio button instance, or sets a new name for a group of radio buttons.
<code>FRadioButton.setLabel</code>	Applies a label for the radio button at runtime.
<code>FRadioButton.setLabelPlacement</code>	Specifies whether the label appears to the left or right of the radio button.
<code>FRadioButton.setSize</code>	Sets the width of the radio button, in pixels.
<code>FRadioButton.setState</code>	Sets the selected state of radio button instance.
<code>FRadioButton.setStyleProperty</code>	Sets a single style property for a component instance.
<code>FRadioButton.setValue</code>	Selects a radio button in a radio button group at runtime.

FRadioButton.getData

Availability

Flash Player 6.

Usage

```
myRadioButton.getData()
```

Parameters

None.

Returns

A string.

Description

Method; returns the data associated with the specified radio button instance. Use `FRadioButton.getValue` to get the data associated with the selected radio button in a group of radio buttons.

Example

The following code returns the data associated with the radio button `flashRadio` to the Output window.

```
trace(flashRadio.getData());
```

See also

`FRadioButton.setData`

FRadioButton.setEnabled

Availability

Flash Player 6.

Usage

```
myRadioButton.setEnabled()
```

```
myRadioButtonGroup.setEnabled()
```

Parameters

None.

Returns

A Boolean value or `undefined`.

Description

Method; indicates whether a radio button instance or radio button group is enabled.

Usage 1: Indicates whether *myRadioButton* is enabled (`true`) or disabled (`false`).

Usage 2: Indicates whether the buttons in *myRadioButtonGroup* are enabled (`true`) or disabled (`false`). If some of the buttons in the group are enabled and some are disabled, the method returns `undefined`.

Example

The following code returns the enabled state of `radio1` to the Output window.

```
trace(radio1.setEnabled());
```

See also

`FRadioButton.setEnabled`

FRadioButton.getLabel

Availability

Flash Player 6.

Usage

```
myRadioButton.getLabel()
```

Parameters

None.

Returns

A string.

Description

Method; returns the label of the specified radio button as a string. You cannot use this method to get the labels for a group of radio buttons; the syntax *radioButtonGroup*.getLabel is not valid.

Example

The following code returns the label of the instance *radio2* to the Output window.

```
trace(radio2.getLabel());
```

See also

FRadioButton.setLabel

FRadioButton.getState

Availability

Flash Player 6.

Usage

```
myRadioButton.getState()
```

Parameters

None.

Returns

A Boolean value indicating the selected state of the radio button.

Description

Method; returns a Boolean value indicating whether *myRadioButton* is selected (true) or not (false).

Example

The following code returns the selected state of the radio button *radio1* to the Output window.

```
trace(radio1.getState());
```

See also

FRadioButton.setState

FRadioButton.getValue

Availability

Flash Player 6.

Usage

```
myRadioButtonGroup.getValue()
```

Parameters

None.

Returns

A string or undefined.

Description

Method; returns the data associated with the selected radio button in *myRadioButtonGroup*, or the label of the radio button if no data is specified. If no button is selected, the method returns undefined.

Example

The following code returns the data associated with the selected radio button in the group `radioGroup1` to the Output window.

```
trace(radioGroup1.getValue());
```

See also

`FRadioButton.setValue`

FRadioButton.registerSkinElement

Availability

Flash Player 6.

Usage

```
myRadioButton.registerSkinElement(element, styleProperty)
```

Parameters

element A movie clip instance.

styleProperty The name of an `FStyleFormat` property.

Returns

Nothing.

Description

Method; registers a skin element to a style property. Skin elements are registered to properties in the first frame of the Read Me layer of each skin the library.

Components are made up of skins, and each skin is composed of several skin elements, each of which can be registered to a property of the `FStyleFormat` object. These properties are assigned values by the style format assigned to a component. By default, all Flash UI components are assigned the `globalStyleFormat` object, which is an instance of the `FStyleFormat` object.

Use this method to register custom skin elements and properties to Flash UI or custom component skins by editing the code in the first frame of the Read Me layer of a skin in the library.

The FRadioButton component uses the skins in the FRadioButton Skins folder and the FLabel skin in the Global Skins folder once you've added the component to the Flash document.

For more information, see “Customizing component skins” in the “Using Components” chapter of *Using Flash*.

Example

The following code registers the custom skin element `myDot_mc` to the `FStyleFormat` property `radioDot` in the ReadMe file for the `frb_dot` skin located in the FRadioButton Skins folder in the library.

```
radiol.registerSkinElement(myDot_mc, "radioDot");
```

See also

`FStyleFormat` (object)

FRadioButton.setChangeHandler

Availability

Flash Player 6.

Usage

```
myRadioButton.setChangeHandler(functionName, [location])
```

```
myRadioButtonGroup.setChangeHandler(functionName, [location])
```

Parameters

functionName A string specifying the name of the handler function to execute when the value of a radio button changes. If the *location* parameter is not specified, this function must be in the same Timeline as the component instance.

location A reference to a data object, movie clip, or Timeline that contains the specified function. This parameter is optional and defaults to the parent Timeline of the component.

Returns

Nothing.

Description

Method; specifies a change handler function to call when the radio button selection changes. You can specify the same change handler function for more than one component; the function always accepts the instance of the component that has changed as a parameter. Calling this method overrides the Change Handler parameter value specified in authoring.

Usage 1: Specifies the function to call if the radio button instance *myRadioButton* is selected or deselected.

Usage 2: Specifies the function to call if the selected radio button in the group *radioButtonGroup* changes. Specifying a function for a group of radio buttons is the equivalent of specifying the same function for all the radio buttons in that group individually with *myRadioButton.setChangeHandler*.

For more information, see “Writing change handler functions for components” in the “Using Components” chapter of *Using Flash*.

Example

Usage 1: The following code specifies `myHandler` as the function called when `radio1` is selected.

```
radio1.setChangeHandler("myHandler");
```

Usage 2: The following code specifies `onChange` as the function called when a radio button in the group `radioGroup1` is selected.

```
radioGroup1.setChangeHandler("onChange");
```

The following code specifies `onChange` as the function called when the user selects a radio button in `radioGroup1`. Because the *location* parameter is not specified, `onChange` must be in the same Timeline as the component instance. The `component` parameter in `onChange` is automatically set with the component (the component that has changed as the result of user input and that specifies `onChange` as its change handler)—in this case, a radio button in the group. The actions defined in `onChange` specify that when the user selects a radio button, the instance name is written to the Output window.

```
radioGroup1.setChangeHandler("onChange");
```

```
function onChange(component){  
    trace(component._name);  
}
```

If in the preceding example `onChange` is a function located in the great-grandparent Timeline of the component’s Timeline, the first line of code would be as follows:

```
radioGroup1.setChangeHandler("onChange", _parent._parent._parent);
```

The following code creates the function `onChange` in an instance of `myObject` (which is of class `Object`), and then specifies `onChange` as the function for `radioGroup1`.

```
myObject = new Object();  
myObject.onChange = function(component){  
    trace(component._name);  
}
```

```
radioGroup1.setChangeHandler("onChange", myObject);
```

FRadioButton.setData

Availability

Flash Player 6.

Usage

```
myRadioButton.setData("data")
```

Parameters

data The data to associate with the radio button instance.

Returns

Nothing.

Description

Method; specifies the data to associate with the radio button instance. Calling this method overrides the Data parameter value set during authoring.

Example

The following code specifies the data ActionScript for the radio button instance flashRadio.

```
flashRadio.setData("ActionScript");
```

See also

FRadioButton.getData, FRadioButton.setValue

FRadioButton.setEnabled

Availability

Flash Player 6.

Usage

```
myRadioButton.setEnabled(enable)
myRadioButtonGroup.setEnabled(enable)
```

Parameters

enable A Boolean value specifying whether an individual radio button or all the buttons in a group are enabled (*true*) or disabled (*false*).

Returns

Nothing.

Description

Method; enables and disables radio buttons at runtime.

Usage 1: Specifies whether *myRadioButton* is enabled (*true*) or disabled (*false*).

Usage 2: Specifies whether all the radio buttons with the group name *radioButtonGroup* are enabled (*true*) or disabled (*false*). Calling this method without passing a parameter is the same as passing the parameter *true*.

Example

Usage 1: The following code disables the single radio button `radio1` without disabling the other buttons in the group.

```
radio1.setEnabled(false);
```

Usage 2: The following code disables all the radio buttons in the group `radioGroup1`.

```
radioGroup1.setEnabled(false);
```

See also

`FRadioButton.setEnabled`

FRadioButton.setGroupName

Availability

Flash Player 6.

Usage

```
myRadioButton.setGroupName(groupName)
```

```
myRadioButtonGroup.setGroupName(groupName)
```

Parameters

groupName A string specifying the name of a radio button group.

Returns

Nothing.

Description

Method; applies a group name to a radio button instance or group of radio buttons at runtime. Calling this method overrides the Group Name parameter value set during authoring.

Usage 1: Specifies *myRadioButton* as a member of the radio button group *groupName*.

Usage2: Specifies a new group name for all of the radio buttons in *myRadioButtonGroup*.

Example

Usage 1: The following code specifies `Colors` as the group name for the radio button instance `radioRed`.

```
radioRed.setGroupName("Colors");
```

Usage2: The following code specifies `radioGroupToys` as the new group name for all the radio buttons in `radioGroupGames`.

```
radioGroupGames.setGroupName("radioGroupToys");
```

FRadioButton.setLabel

Availability

Flash Player 6.

Usage

```
myRadioButton.setLabel(label)
```

Parameters

label A text string specifying the label that appears to the right of the radio button.

Returns

Nothing.

Description

Method; applies a label to the radio button instance *myRadioButton* at runtime. Calling this method overrides the *label* parameter value specified in authoring. You cannot use this method to set labels for groups of radio buttons; the syntax *radioButtonGroup.setLabel* is not valid.

Example

The following code applies the label `Brown eyes` to `radio1`.

```
radio1.setLabel("Brown eyes");
```

See also

`FRadioButton.getLabel`

FRadioButton.setLabelPlacement

Availability

Flash Player 6.

Usage

```
myRadioButton.setLabelPlacement(labelPosition)
```

```
myRadioButtonGroup.setLabelPlacement(labelPosition)
```

Parameters

labelPosition A text string; specify "left" or "right".

Description

Method; specifies whether the label appears to the left or right of the radio button. Calling this method overrides the Label Placement parameter value set during authoring.

Usage 1: specifies the placement of the label for a single radio button.

Usage 2: specifies the placement of the labels for all the radio buttons in a group.

Example

Usage 1: The following code places the label for `radio1` to the left of the radio button.

```
radio1.setLabelPlacement("left");
```

Usage 2: The following code places the labels for the radio buttons in the group `Colors` to the right of the buttons.

```
Colors.setLabelPlacement("right");
```

See also

`FRadioButton.setLabel`, `FRadioButton.setLabelPlacement`

FRadioButton.setSize

Availability

Flash Player 6.

Usage

```
myRadioButton.setSize(width)
```

```
myRadioButtonGroup.setSize(width)
```

Parameters

width An integer specifying the size of the radio button, in pixels.

Returns

Nothing.

Description

Method; specifies the width of the radio button in pixels and redraws the radio button. (You cannot set the height of radio button components.) Calling this method overrides width scaling applied during authoring.

Usage 1: Sets the size of an individual radio button.

Usage 2: Sets the size of all of the radio buttons in a group.

For more information, see “Sizing RadioButton components” in the “Using Components” chapter of *Using Flash*.

Example

The following code sets the width of `radio1` to 200 pixels.

```
radio1.setSize(200);
```

FRadioButton.setState

Availability

Flash Player 6.

Usage

```
myRadioButton.setState("select")
```

Parameters

select A Boolean value indicating whether the radio button is selected (`true`) or not (`false`).

Returns

Nothing.

Description

Method; specifies whether *myRadioButton* is selected (`true`) or unselected (`false`). Only one radio button in a group (all having the same Group Name parameter) can have an initial state of `true` (selected). If more than one radio button has `true` specified for this parameter, the last radio button with an initial state parameter of `true` is selected. The default value for this parameter is `false`.

Calling this method overrides the Initial State parameter value set during authoring. If you call this method and also call `FRadioButton.setValue` to select a radio button at runtime, and the radio buttons are different buttons in the same group, the radio button specified in the last method called is selected.

Example

The following code selects the radio button `radio1` at runtime.

```
radio1.setState(true));
```

See also

`FRadioButton.getState`, `FRadioButton.getValue`, `FRadioButton.setValue`

FRadioButton.setStyleProperty

Availability

Flash Player 6.

Usage

```
myRadioButton.setStyleProperty(styleProperty, value)  
myRadioButtonGroup.setStyleProperty(styleProperty, value)
```

Parameters

styleProperty A string specifying a property of the `FStyleFormat` object.

value The value to set for the property.

Returns

Nothing.

Description

Method; sets an `FStyleFormat` property for an individual radio button instance. Calling this method to specify a property overrides the setting for this property in the style format assigned to the component. Passing `undefined` as the value for a property removes all styles for that property.

To set `FStyleFormat` properties for multiple components, create a custom style format. For more information on creating custom style formats, see “Customizing component colors and text” in the “Using Components” chapter of *Using Flash*.

Example

The following code sets the `radioDot` property for `radioButton1` to `0xFF12AC` (pink).

```
radioButton1.setStyleProperty("radioDot", 0xFF12AC);
```

The following code sets the `radioDot` property for all the buttons in `radioGroup1` to `0xFF12AC` (pink).

```
radioGroup1.setStyleProperty("radioDot", 0xFF12AC);
```

See also

`FStyleFormat` (object)

FRadioButton.setValue

Availability

Flash Player 6.

Usage

```
myRadioButtonGroup.setValue("data")
```

Parameters

data The data associated with the radio button to select.

Returns

Nothing.

Description

Method; selects the radio button associated with the specified data and deselects any currently selected button is the same group.

Calling this method overrides the Initial Value parameter value set during authoring. If you call this method and also call `FRadioButton.setState` to select a radio button at runtime, and the radio buttons are different buttons in the same group, the radio button specified in the last method called is selected.

Example

The following code selects the radio button with the associated data `red` in the radio button group called `Colors`.

```
Colors.setValue("red");
```

See also

`FRadioButton.getData`, `FRadioButton.getValue`, `FRadioButton.setState`

FScrollBar (component)

The ScrollBar component in the Flash authoring environment provides drag-and-drop functionality for adding scroll bars to dynamic and input text fields in Flash documents; it also provides a user interface for setting basic parameters. The methods of the FScrollBar component allow you to control scroll bars at runtime: you can create scroll bars, control scroll bars created in authoring, set or override basic parameters, and set additional runtime options. You do not need to use a constructor to access the methods of components. Some of the methods of the FScrollBar component are not recommended for use with scroll bars attached to text fields. See individual method entries for details.

Advanced users and programmers can use the ScrollBar component with other Flash elements to create custom user interfaces.

Component methods do not perform error checking for type, as do other native ActionScript objects and actions; therefore, it is recommended that you validate parameters before passing them to methods.

The ScrollBar component is supported by Flash Player 6 and later versions of the Flash Player.

For information on using the ScrollBar component, setting parameters during authoring, and changing the color and appearance of components, see “Customizing component colors and text” and “Customizing component skins” in the “Using Components” chapter in *Using Flash*.

Method summary for the FScrollBar component

Method	Description
FScrollBar.setEnabled	Returns true if the scroll bar is enabled, false if it is disabled.
FScrollBar.getScrollPosition	Returns an integer representing the current position of the scroll box (thumb).
FScrollBar.registerSkinElement	Registers a skin element to a property defined for a skin in the Read Me layer located on Frame 1 of a skin movie clip in the library.
FScrollBar.setChangeHandler	Specifies a function to call every time the scroll position changes. (You cannot use this method with text fields.)
FScrollBar.setEnabled	Specifies whether the scroll bar is enabled (true) or disabled (false).
FScrollBar.setHorizontal	Specifies whether the scroll bar is horizontal (true) or vertical (false).
FScrollBar.setLargeScroll	Specifies the number of positions scrolled when the user clicks the track.
FScrollBar.setScrollContent	Specifies the text field instance to which the scroll bar applies.
FScrollBar.setScrollPosition	Sets the position of the scroll box as an integer between minPos and maxPos.
FScrollBar.setScrollProperties	Sets the pageSize, minPos, and maxPos properties for the scroll bar. (You cannot use this method with text fields.)
FScrollBar.setScrollTarget	Specifies a text field as the target for the scroll bar.
FScrollBar.setSize	Sets the length of the scroll bar, in pixels.
FScrollBar.setSmallScroll	Specifies the number of positions scrolled when the user clicks a scroll arrow.
FScrollBar.setStyleProperty	Sets a single style property for a component.

FScrollBar.setEnabled

Availability

Flash Player 6.

Usage

```
myScrollBar.setEnabled()
```

Parameters

None.

Returns

A Boolean value.

Description

Method; indicates whether the scroll bar is enabled (`true`) or disabled (`false`).

Example

The following code returns a value to the Output window indicating whether `scroll1` is enabled (`true`) or disabled (`false`).

```
trace(scroll1.setEnabled());
```

See also

`FScrollBar.setEnabled`

FScrollBar.getScrollPosition

Availability

Flash Player 6.

Usage

```
myScrollBar.getScrollPosition()
```

Parameters

None.

Returns

An integer.

Description

Method; returns an integer specifying the position of the scroll box (thumb). The returned value is in the range defined by the `minPos` and `maxPos` properties that determine the scrolling boundaries of the scroll bar. To set the `minPos` and `maxPos` parameters, use `FScrollBar.setScrollProperties`.

Example

The following code returns the current position of the scroll box for the scroll bar `scroll2` to the Output window. If `scroll2` has a `minPos` setting of 2 and a `maxPos` setting of 25, a return value of 12 indicates that the scroll box is in the middle of the scroll bar.

```
trace(scroll2.getPosition());
```

See `FScrollBar.setChangeHandler` for another example that uses this method.

See also

`FScrollBar.setChangeHandler`, `FScrollBar.setScrollPosition`

FScrollBar.registerSkinElement

Availability

Flash Player 6.

Usage

```
myScrollBar.registerSkinElement(element, styleProperty)
```

Parameters

element A movie clip instance.

styleProperty A string specifying an FStyleFormat property.

Returns

Nothing.

Description

Method; registers a skin element to a style property. Skin elements are registered to properties in the first frame of the Read Me layer of each skin the library.

Components are made up of skins, and each skin is composed of several skin elements, each of which can be registered to a property of the FStyleFormat object. These properties are assigned values by the style format assigned to a component. By default, all Flash UI components are assigned the `globalStyleFormat` object, which is an instance of the FStyleFormat object.

Use this method to register custom skin elements and properties to Flash UI or custom component skins by editing the code in the first frame of the Read Me layer of a skin in the library.

The FScrollBar component uses the skins in the FScrollBar Skins folder and the FLabel skin in the Global Skins folder once you've added the component to the Flash document. Editing any of the skins in the FScrollBar Skins folder affects all the components that use scroll bars (ComboBox, ListBox, ScrollBar, and ScrollPane).

For more information, see "Customizing component skins" in the "Using Components" chapter of *Using Flash*.

Example

The following code registers the custom skin element `NewArrow_mc` to the `arrow` property in the first frame of the Read Me layer of the `fsb_downArrow` skin in the FScrollBar Skins folder in the library.

```
Scroll11.registerSkinElement(NewArrow_mc, "arrow");
```

See also

FStyleFormat (object)

FScrollBar.setChangeHandler

Availability

Flash Player 6.

Usage

```
myScrollBar.setChangeHandler(functionName, [location])
```

Parameters

functionName A string specifying the name of the handler function to execute when the user moves the scroll box. If the *location* parameter is not specified, this function must be in the same Timeline as the component instance.

location A path reference to a data object, movie clip, or Timeline that contains the specified function. This parameter is optional and defaults to the parent Timeline of the component.

Returns

Nothing.

Description

Method; specifies a change handler to call when the user moves the scroll bar's scroll box (thumb). You can specify the same change handler function for more than one component; the function always accepts the instance of the component that has changed as a parameter. Calling this method overrides the Change Handler parameter value specified in authoring.

This method is for advanced users and programmers who create applications and custom components using the Flash UI ScrollBar component; the method cannot be used with scroll bars attached to text fields.

Example

The following code creates a filled box on the Stage, then applies a horizontal scroll bar, sets the scroll properties, and specifies the function `mover` as the change handler. The `mover` change handler uses the scroll bar's scroll position to change the movie clip's `_x` position between 50 and 250.

```
root.createEmptyMovieClip("square", 1);
_root.square._x = 50;
_root.square._y = 50;

with (_root.square) {
    moveTo(0, 0);
    beginFill(0x0066CC);
    lineTo(20, 0);
    lineTo(20, 20);
    lineTo(0, 20);
    lineTo(0, 0);
    endFill();
}

scrollBar._x = 50;
scrollBar.setHorizontal (true);
scrollBar.setScrollProperties (1, 50, 250);
scrollBar.setChangeHandler ("mover");

function mover () {
    _root.square._x = scrollBar.getScrollPosition();
}
```

The following code specifies a change handler function for an instance of the scroll bar component attached to a custom list box component. The change handler sets up `scroll1` to get the current scroll position using `FScrollBar.getScrollPosition`, and then `customListBox` uses `FScrollBar.setScrollPosition` to reset the scroll position so that the item at the current scroll position is displayed at the top of the custom list box view. The component parameter is automatically filled in with the instance of a component (the component that has changed as the result of user input and that specifies `myHandler` as its change handler).

```
scroll1.setChangeHandler("myHandler");

function myHandler(component)
{
    customListBox.setScrollPosition(component.getScrollPosition());
}
```

If in the preceding example `myHandler` is a function located in the great-grandparent Timeline of the component's Timeline, the first line of code would be as follows:

```
scroll1.setChangeHandler("myHandler", _parent._parent._parent);
```

The following code creates the function `myHandler` in an instance of `myObject` (which is of class `Object`), and then specifies `myHandler` as the function for `scroll1`.

```
myObject = new Object();
myObject.myHandler = function(component){
    customListBox.setScrollPosition(component.getScrollPosition());
}

scroll1.setChangeHandler("myHandler", myObject);
```

See also

`FScrollBar.getScrollPosition`, `FScrollBar.setScrollPosition`,
`FScrollBar.setScrollProperties`

FScrollBar.setEnabled

Availability

Flash Player 6.

Usage

```
myScrollBar.setEnabled(enable)
```

Parameters

enable A Boolean value specifying whether the scroll bar is enabled (`true`) or disabled (`false`).

Returns

Nothing.

Description

Method; determines whether the scroll bar is enabled (`true`) or disabled (`false`). If a scroll bar is disabled, it does not accept mouse or keyboard interaction from the user, and it is dimmed (unavailable to the user). Calling this method without passing a parameter is the same as passing a parameter of `true`.

Example

The following code disables the scroll bar `scroll2`.

```
scroll2.setEnabled(false);
```

See also

`FScrollBar.setEnabled`

FScrollBar.setHorizontal

Availability

Flash Player 6.

Usage

```
myScrollBar.setHorizontal(horizontalScroll)
```

Parameters

horizontalScroll A Boolean value specifying whether the scroll bar is horizontal (`true`) or vertical (`false`).

Returns

Nothing.

Description

Method; specifies whether the scroll bar is applied to the target horizontally (`true`) or vertically (`false`). This method defaults to `false`.

Example

The following code specifies that the scroll bar `scrollText` is applied horizontally to its target.

```
scrollText.setHorizontal(true);
```

See also

`FScrollBar.setSize`

FScrollBar.setLargeScroll

Availability

Flash Player 6.

Usage

```
myScrollBar.setLargeScroll(largeScroll)
```

Parameters

largeScroll An integer specifying the number of positions to scroll when the user clicks the track once. The default value is the value set for `pageSize` with `FScrollBar.setScrollProperties`.

Returns

Nothing.

Description

Method; sets the `largeScroll` property of the scroll bar instance at runtime. When the user clicks the scroll track once, the scroll box (thumb) moves the distance specified for one `largeScroll` position.

Example

The following code specifies that when the user clicks the track, `scrollText1` is scrolled 20 positions.

```
scrollText1.setLargeScroll(20);
```

See also

`FScrollBar.setSmallScroll`

FScrollBar.setScrollContent

Availability

Flash Player 6.

Usage

```
myScrollBar.setScrollContent(target)
```

Parameters

target A reference to the text field for the scroll bar.

Returns

Nothing.

Description

Method; specifies the text field instance to which the scroll bar applies. This instance must be defined in the same Timeline and on the same level as the scroll bar. Calling this method overrides the Target Text Field parameter value set during authoring. Passing `undefined` for the *target* parameter disassociates the scroll bar from the text field.

Example

The following code attaches `scrollText1` to the text field with the instance name `textField1`.

```
scrollText1.setScrollContent("textField1");
```

FScrollBar.setScrollPosition

Availability

Flash Player 6.

Usage

```
myScrollBar.setScrollPosition(position)
```

Parameters

position An integer between the `minPos` and `maxPos` settings of the scroll bar. See `FScrollBar.setScrollProperties` for more information on setting the `minPos` and `maxPos` properties.

Returns

Nothing.

Description

Method; specifies the position of the scroll box (thumb) on the scroll bar and executes the change handler function specified with `FScrollBar.setChangeHandler`.

Example

The following code sets the position of the scroll box for `scrollText1` to 5.

```
scrollText1.setScrollPosition(5);
```

See `FScrollBar.setChangeHandler` for another example that uses this method.

See also

`FScrollBar.setChangeHandler`, `FScrollBar.setScrollProperties`

FScrollBar.setScrollProperties

Availability

Flash Player 6.

Usage

```
myScrollBar.setScrollProperties(pageSize, minPos, maxPos)
```

Parameters

pageSize An integer representing the number of positions displayed in the page view.

minPos An integer representing the minimum scrolled position.

maxPos An integer representing the maximum scrolled position.

Returns

Nothing.

Description

Method; specifies the `pageSize`, `minPos`, and `maxPos` properties of the scroll bar and sets the scroll bar's scroll box (thumb) to the proper size.

This method is for advanced users and programmers who create custom components; it cannot be used with scroll bars attached to text fields. When a scroll bar is attached to a text field, the scroll properties are automatically set according to the properties of the text field, and calling this method breaks the scroll bar for the text field.

Example

The following code sets the `pageSize`, `minPos`, and `maxPos` properties for a scroll bar attached to a custom list box component. The list box has 5 visible rows and a total of 20 items in the list. Given that the box is indexed from 0 to 19, the `maxPos` property is the total number of items in the box minus the number of items visible.

```
scrollBar.setScrollProperties(5, 0, 15);
```

See also

`FScrollBar.setScrollPosition`

FScrollBar.setScrollTarget

Availability

Flash Player 6.

Usage

```
myScrollBar.setScrollTarget(target)
```

Parameters

target A reference to the text field for the scroll bar.

Returns

Nothing.

Description

Method; specifies the text field instance to which the scroll bar applies. This instance must be defined in the same Timeline and on the same level as the scroll bar. Calling this method overrides the Target Text Field parameter value set during authoring. Passing *undefined* for the *target* parameter disassociates the scroll bar from the text field.

Example

The following code attaches `scrollText1` to the text field with the instance name `textField1`.

```
scrollText1.setScrollTarget("textField1");
```

FScrollBar.setSize

Availability

Flash Player 6.

Usage

```
myScrollBar.setSize(length)
```

Parameters

length An integer specifying the length of the scroll bar, in pixels.

Returns

Nothing.

Description

Method; sets the length, in pixels, of the scroll bar at runtime. (You cannot set the width of scroll bar components.) Calling this method overrides any scaling and sizing applied during authoring.

This method should not be used with scroll bars attached to text fields; the scroll bar automatically snaps to the size of the text field during authoring.

For more information, see “Sizing ScrollBar components” in the “Using Components” chapter of *Using Flash*.

Example

The following code sets the length of `scrollText1` to 200 pixels.

```
scrollText1.setSize(200);
```

FScrollBar.setSmallScroll

Availability

Flash Player 6.

Usage

```
myScrollBar.setSmallScroll(smallScroll)
```

Parameters

smallScroll An integer specifying the number of positions to scroll when the user clicks a scroll arrow. The default value is 1.

Returns

Nothing.

Description

Method; sets the `smallScroll` property of the scroll bar instance at runtime, if the text field has the focus. When the user clicks the scroll bar's arrows, or an arrow key on the keyboard, the scroll box (thumb) moves the distance specified for one *smallScroll* position.

Example

The following code specifies that when the user clicks a scroll arrow, `scrollText1` is scrolled 5 positions.

```
scrollText1.setSmallScroll(5);
```

See also

`FScrollBar.setLargeScroll`

FScrollBar.setStyleProperty

Availability

Flash Player 6.

Usage

```
myScrollBar.setStyleProperty(styleProperty, value)
```

Parameters

styleProperty A string specifying a property of the `FStyleFormat` object.

value The value to set for the property.

Returns

Nothing.

Description

Method; sets an `FStyleFormat` property for an individual scroll bar instance. Calling this method to specify a property overrides the setting for this property in the style format assigned to the component. Passing `undefined` as the value for a property removes all styles for that property.

To set `FStyleFormat` properties for multiple components, create a custom style format. For more information on creating custom style formats, see “Customizing component colors and text” in the “Using Components” chapter of *Using Flash*.

Example

The following code sets the arrow property for `scrollBar1` to `0x000000` (black).

```
scrollBar1.setStyleProperty("arrow", 0x000000);
```

See also

`FStyleFormat` (object)

FScrollPane (component)

The `ScrollPane` component in the Flash authoring environment provides drag-and-drop functionality for adding scroll panes to display movie clips in Flash documents; it also provides a user interface for setting basic parameters. The methods of the `FScrollPane` component allow you to control scroll panes at runtime: you can create scroll panes, control scroll panes created in authoring, set or override basic parameters, and set additional runtime options. You do not need to use a constructor to access the methods of components.

The `ScrollPane` component provides vertical and horizontal scroll bars that let you display large movie clips without taking up much Stage space. Standard mouse and keyboard controls are built in.

Note: The `ScrollPane` component only displays movie clips; to add scroll bars to dynamic and input text fields, use the `ScrollBar` component. The `ScrollPane` component cannot display any content that uses device fonts.

Component methods do not perform error checking for type, as do other native `ActionScript` objects and actions; therefore, it is recommended that you validate parameters before passing them to methods.

The `ScrollPane` component is supported by Flash Player 6 and later versions of the Flash Player.

For information on using the `ScrollPane` component, setting parameters during authoring, and changing the color and appearance of components, see “Customizing component colors and text” and “Customizing component skins” in the “Using Components” chapter of *Using Flash*.

Method summary for the FScrollPane component

Method	Description
<code>FScrollPane.getPaneHeight</code>	Returns the height of the scroll pane.
<code>FScrollPane.getPaneWidth</code>	Returns the width of the scroll pane.
<code>FScrollPane.getScrollContent</code>	Returns an instance of the content displayed in the scroll pane.
<code>FScrollPane.getScrollPosition</code>	Returns the x and y coordinates of the current scroll position.
<code>FScrollPane.loadScrollContent</code>	Loads a SWF or JPEG into the scroll pane.
<code>FScrollPane.refreshPane</code>	Resizes the scroll bars on the scroll pane when the content changes size.
<code>FScrollPane.registerSkinElement</code>	Registers a skin element to a property defined for a skin in the Read Me layer located on Frame 1 of a skin movie clip in the library.
<code>FScrollPane.setDragContent</code>	Sets the content of the scroll pane as draggable.
<code>FScrollPane.setHScroll</code>	Sets the horizontal scrolling style for the scroll pane.
<code>FScrollPane.setScrollContent</code>	Sets a movie clip as the scroll pane target.

Method	Description
<code>FScrollPane.setScrollPosition</code>	Causes the pane to scroll to the specified x, y coordinates.
<code>FScrollPane.setSize</code>	Sets the width and height of the scroll pane, in pixels.
<code>FScrollPane.setStyleProperty</code>	Sets a single style property for a component.
<code>FScrollPane.setVScroll</code>	Sets the vertical scrolling style of the scroll pane.

FScrollPane.getPaneHeight

Availability

Flash Player 6.

Usage

```
myScrollPane.getPaneHeight()
```

Parameters

None.

Returns

An integer specifying the height of the scroll pane view.

Description

Method; returns the height of the scroll pane view. You can only use this method to get the height of a scroll pane that was sized with `FScrollPane.setSize`. This method works only if the scroll pane was sized with `FScrollPane.setSize`, not if you set the size using the `_width` and `_height` properties.

Example

The following code gets the height and width of the scroll pane `display1` and uses the returned values to resize the scroll pane with `FScrollPane.setSize`.

```
var h = display1.getPaneHeight();
var w = display1.getPaneWidth();
display1.setSize(w+10, h+10);
```

See also

`FScrollPane.getPaneWidth`, `FScrollPane.setSize`

FScrollPane.getPaneWidth

Availability

Flash Player 6.

Usage

```
myScrollPane.getPaneWidth()
```

Parameters

None.

Returns

An integer specifying the width of the scroll pane view.

Description

Method; returns the width of the scroll pane view. You can only use this method to get the width of a scroll pane that was sized with `FScrollPane.setSize`. This method works only if the scroll pane was sized with `FScrollPane.setSize`, not if you set the size using the `_width` and `_height` properties.

Example

The following code gets the height and width of the scroll pane `display1` and uses the returned values to resize the scroll pane with `FScrollPane.setSize`.

```
var h = display1.getPaneHeight();
var w = display1.getPaneWidth();
display1.setSize(w+10, h+10);
```

See also

`FScrollPane.getPaneHeight`, `FScrollPane.setSize`

FScrollPane.getScrollContent

Availability

Flash Player 6.

Usage

```
myScrollPane.getScrollContent()
```

Parameters

None.

Returns

A reference to the movie clip in the scroll pane.

Description

Method; returns an instance of the content displayed in the scroll pane.

Example

The following code retrieves a reference to the movie clip inside `display1`, stores it in a variable, and then tells the movie clip to go to frame 4.

```
var content = display1.getScrollContent();
content.gotoAndStop(4);
```

See also

`FScrollPane.setScrollContent`

FScrollPane.getScrollPosition

Availability

Flash Player 6.

Usage

```
myScrollPane.getScrollPosition()
```

Parameters

None.

Returns

An object.

Description

Method; returns an object with the fields `.x` or `.y` specifying the current vertical or horizontal scroll position of the scroll pane view.

Example

The following code returns the current scroll position of the scroll pane `scroll2` to the Output window.

```
trace(scroll2.getScrollPosition());
```

See also

`FScrollPane.setScrollPosition`

FScrollPane.loadScrollContent

Availability

Flash Player 6.

Usage

```
myScrollPane.loadScrollContent(URL [, funcName, location])
```

Parameters

URL A string specifying the URL of a SWF or JPEG file to load into the scroll pane.

funcName A string specifying the name of the handler function to execute when the contents of the scroll pane loads. If the *location* parameter is not specified, this function must be in the same Timeline as the component instance.

location A path reference to a data object, movie clip, or Timeline that contains the specified function. This parameter is optional and defaults to the parent Timeline of the component.

Returns

Nothing.

Description

Method; specifies the URL of a SWF or JPEG file to display in the scroll pane. The optional *funcName* and *location* parameters allow you to specify a change handler function to call when the content loads.

The URL must be in the same subdomain as the URL where the Flash movie currently resides. To use SWF or JPEG files in the Flash Player or test the movie in the Flash authoring environment, you must store all SWF or JPEG files in the same folder, and their filenames cannot include folder or disk drive specifications.

Calling this method overrides the Scroll Content parameter value set in authoring.

See `FScrollBar.setChangeHandler` for more information and examples of using change handler functions.

Example

The following code loads a JPEG located on a server into `display1`.

```
display1.loadScrollContent("http://www.YourWebServer.com/Nice.jpg");
```

The following code loads a JPEG located on a server and specifies the change handler function `load` located in the grandparent Timeline of the component `display1`.

```
display1.loadScrollContent("http://www.YourWebServer.com/Nice.jpg" , "load" ,
    _parent._parent);

function load(component){

    //content is loaded
    component.setScrollPosition(10,10);
}
```

See also

`FScrollPane.getPaneHeight`, `FScrollPane.setScrollContent`

FScrollPane.refreshPane

Availability

Flash Player 6.

Usage

```
myScrollPane.refreshPane()
```

Parameters

None.

Returns

Nothing.

Description

Method; resizes the scroll bars of the scroll pane when the contents inside the scroll pane changes. Call this method if you resize the content in the scroll pane window using `_width` or `_height`.

Example

The following code updates the scroll bars of `moviePane` after increasing the size of the movie clip `myContent` in the scroll pane.

```
var myContent = moviePane.getScrollContent();
myContent._width = 400;
moviePane.refreshPane();
```

See also

`FScrollPane.getScrollContent`

FScrollPane.registerSkinElement

Availability

Flash Player 6.

Usage

```
myScrollPane.registerSkinElement(element, styleProperty)
```

Parameters

element A movie clip instance.

styleProperty The name of an FStyleFormat property.

Returns

Nothing.

Description

Method; registers a skin element to a style property. Skin elements are registered to properties in the first frame of the Read Me layer of each skin in the library.

Components are made up of skins, and each skin is composed of several skin elements, each of which can be registered to a property of the FStyleFormat object. These properties are assigned values by the style format assigned to a component. By default, all Flash UI components are assigned the `globalStyleFormat` object, which is an instance of the FStyleFormat object.

Use this method to register custom skin elements and properties to Flash UI or custom component skins by editing the code in the first frame of the Read Me layer of a skin in the library.

The FScrollPane component uses the skins in the FScrollBar Skins folder and the FLabel skin in the Global Skins folder once you've added the component to the Flash document. Editing any of the skins in the FScrollBar Skins folder affects all the components that use scroll bars (ComboBox, ListBox, ScrollBar, and ScrollPane).

For more information, see "Customizing component skins" in the "Using Components" chapter of *Using Flash*.

Example

The following code registers the custom skin element `NewArrow_mc` to the `arrow` property in the first frame of the Read Me layer of the `fsb_downArrow` skin in the FScrollBar Skins folder in the library.

```
Panel1.registerSkinElement(NewArrow_mc, "arrow");
```

See also

FStyleFormat (object)

FScrollPane.setDragContent

Availability

Flash Player 6.

Usage

```
myScrollPane.setDragContent(drag)
```

Parameters

drag A Boolean value; `true` specifies that the user can change the view by dragging the content in the scroll pane; `false` specifies that the user can change the view only by using the scroll bars.

Returns

Nothing.

Description

Method; specifies whether the user can change the scroll pane view by dragging its content in addition to using the scroll bars. Calling this method overrides the Drag Content parameter value set during authoring.

Example

The following example specifies that the content in the scroll pane `display1` is draggable.

```
display1.setDragContent(true);
```

FScrollPane.setHScroll

Availability

Flash Player 6.

Usage

```
myScrollPane.setHScroll(display)
```

Parameters

display A Boolean value specifying whether the scroll bar is always displayed (`true`) or never displayed (`false`), or a string specifying that the scroll bar is displayed only when necessary (`"auto"`).

Calling this method overrides the Horizontal Scroll parameter value set during authoring.

Returns

Nothing.

Description

Method; determines whether a horizontal scroll bar is always displayed (`true`), never displayed (`false`), or only displayed when necessary (`"auto"`). The default value is `auto`.

Example

The following code hides the horizontal scroll bar for `display1`.

```
display1.setHScroll(false);
```

See also

`FScrollPane.setVScroll`

FScrollPane.setScrollContent

Availability

Flash Player 6.

Usage

```
myScrollPane.setScrollContent(target)
```

Parameters

target A text string specifying the symbol linkage ID of a movie clip in the library or an instance of a movie clip.

Returns

Nothing.

Description

Method; specifies a movie clip to display in the scroll pane. Calling this method overrides the Scroll Content parameter value set in authoring.

Example

The following example specifies the movie clip instance `BetsyTacy` as the `target` for `display1`.

```
display1.setScrollContent("BetsyTacy");
```

See also

`FScrollPane.getPaneHeight`, `FScrollPane.loadScrollContent`

FScrollPane.setScrollPosition

Availability

Flash Player 6.

Usage

```
myScrollPane.setScrollPosition(x, y)
```

Parameters

x An integer specifying the number of pixels (from 0) to scroll to the right.

y An integer specifying the number of pixels (from 0) to scroll down.

Returns

Nothing.

Description

Method; sets the scroll position to the specified *x*, *y* coordinate positions.

Example

The following example causes the contents of `display1` to scroll 14 pixels down and 40 pixels to the right.

```
display1.setScrollPosition(14,40);
```

See also

`FScrollPane.getScrollPosition`

FScrollPane.setSize

Availability

Flash Player 6.

Usage

```
myScrollPane.setSize(width, height)
```

Parameters

width An integer specifying the width of the scroll pane, in pixels.

height An integer specifying the height of the scroll pane, in pixels.

Returns

Nothing.

Description

Method; sets the width and height, in pixels, of the scroll pane view at runtime. Calling this method overrides sizing applied during authoring.

For more information, see “Customizing component skins” in the “Using Components” chapter of *Using Flash*.

Example

The following code sets the width and height of `display1` to 500 x 300 pixels.

```
display1.setSize(500, 300);
```

See also

`FScrollPane.getPaneHeight`, `FScrollPane.getPaneWidth`

FScrollPane.setStyleProperty

Availability

Flash Player 6.

Usage

```
myScrollPane.setStyleProperty(styleProperty, value)
```

Parameters

styleProperty A string specifying a property of the `FStyleFormat` object.

value The value to set for the property.

Returns

Nothing.

Description

Method; sets an `FStyleFormat` property for an individual scroll pane instance. Calling this method to specify a property overrides the setting for this property in the style format assigned to the component. Passing `undefined` as the value for a property removes all styles for that property.

To set `FStyleFormat` properties for multiple components, create a custom style format. For more information on creating custom style formats, see “Customizing component colors and text” in the “Using Components” chapter of *Using Flash*.

Example

The following code sets the arrow property of `ScrollPane2` to `0x000000` (black).

```
ScrollPane2.setStyleProperty("arrow", 0x000000);
```

See also

`FStyleFormat` (object)

FScrollPane.setVScroll

Availability

Flash Player 6.

Usage

```
myScrollBar.setVScroll(display)
```

Parameters

display A Boolean value specifying whether a vertical scroll bar is always displayed (`true`) or never displayed (`false`), or a string specifying that the scroll bar is displayed only when necessary (`"auto"`).

Returns

Nothing.

Description

Method; determines whether a vertical scroll bar is always displayed (`true`), never displayed (`false`), or only displayed when necessary (`"auto"`). The default value is `auto`.

Calling this method overrides the Vertical Scroll parameter value set during authoring.

Example

The following code specifies that the vertical scroll bar for `display1` is always displayed.

```
display1.setVScroll(true);
```

See also

`FScrollPane.setHScroll`

FStyleFormat (object)

The `FStyleFormat` object lets you set or change properties in the global style format assigned to all Flash UI components by default, or create new custom style formats to use with Flash UI components or custom components that you create or acquire from other sources. The global style format, or `globalStyleFormat` object, is an instance of the `FStyleFormat` object that defines the color and text formatting properties used to display all Flash UI components.

To create a new custom style format, you create a new instance of the `FStyleFormat` object using the new `FStyleFormat()` constructor, set the `FStyleFormat` properties that you want to include in your style format, and then use the `FStyleFormat.addListener` method to register component instances to the new style format. A component instance can “listen” to more than one style format, but can only take the value of one style format for a specific property. If you add a component as a listener to a style format, it uses the new style format for properties specified in the format, and it uses the old style format for all other properties.

You do not need to use the `FStyleFormat` constructor to add or remove listeners or set or change properties in the global style format because the `globalStyleFormat` object exists once any Flash UI component is placed on the Stage.

You can set any of the `FStyleFormat` properties for a single instance of a component using the `setStyleProperty` method available to all Flash UI components. Using `setStyleProperty` allows you to set a property for a component without creating an instance of the `FStyleFormat` object. Using `setStyleProperty` overrides the setting for a specific style format property assigned to the component without changing the other property settings. For more information, see the `setStyleProperty` entries for individual components.

When assigning a color value to an `FStyleFormat` property, specify an RGB color in the format `0xRRGGBB`.

For more information on the global style format and creating custom style formats, see “Customizing component colors and text” in the “Using Components” chapter of *Using Flash*.

Method summary for the `FStyleFormat` object

Method	Description
<code>FStyleFormat.addListener</code>	Registers a component to a style format.
<code>FStyleFormat.applyChanges</code>	Applies the changes made to property values in a style format.
<code>FStyleFormat.removeListener</code>	Removes a component as a listener to a style format.

Property summary for the `FStyleFormat` object

The following tables lists the property summaries for the `FStyleFormat` object.

Property	Description
<code>FStyleFormat.arrow</code>	The color of the arrow used in scroll bars and drop-down lists.
<code>FStyleFormat.background</code>	The color of the background portion of a component.
<code>FStyleFormat.backgroundDisabled</code>	The color of the background portion of a disabled component.
<code>FStyleFormat.check</code>	The color of the check mark in a selected check box.
<code>FStyleFormat.darkshadow</code>	The color of the inner border or darker shadow portion of a component.
<code>FStyleFormat.face</code>	The main color of the component.
<code>FStyleFormat.foregroundDisabled</code>	The foreground color of a disabled component.
<code>FStyleFormat.highlight</code>	The color for the inner border or darker shadow portion of a component when it is selected.
<code>FStyleFormat.highlight3D</code>	The color for the outer border or light shadow portion of a component when it is selected.
<code>FStyleFormat.radioDot</code>	The color of the dot in a selected radio button.
<code>FStyleFormat.scrollTrack</code>	The color of the track in a scroll bar.
<code>FStyleFormat.selection</code>	The color of the selection bar highlighting a list item in a component.
<code>FStyleFormat.selectionDisabled</code>	The color of the selection bar that highlights a list item in a disabled component.
<code>FStyleFormat.selectionUnfocused</code>	The color of the selection bar when the component does not have keyboard focus.

Property	Description
<code>FStyleFormat.shadow</code>	The color of the outer border or light shadow portion of a component.
<code>FStyleFormat.textAlign</code>	The alignment (left, right, or center) for text displayed in or on a component.
<code>FStyleFormat.textBold</code>	Specifies whether text is bold (<code>true</code>) or not (<code>false</code>).
<code>FStyleFormat.textColor</code>	The default text color in all components assigned to the style format.
<code>FStyleFormat.textDisabled</code>	The color of the text in a disabled component.
<code>FStyleFormat.textFont</code>	The name of the font to display text.
<code>FStyleFormat.textIndent</code>	The indentation of the text from the left margin to the first text character, in pixels.
<code>FStyleFormat.textItalic</code>	Specifies whether text is italic (<code>true</code>) or not (<code>false</code>).
<code>FStyleFormat.textLeftMargin</code>	The left paragraph margin for text, in pixels.
<code>FStyleFormat.textRightMargin</code>	The right paragraph margin for text, in pixels.
<code>FStyleFormat.textSelected</code>	The color of a selected list item in a component.
<code>FStyleFormat.textSize</code>	The size of the text, in points.
<code>FStyleFormat.textUnderline</code>	Specifies whether text is underlined (<code>true</code>) or not (<code>false</code>).

Constructor for the FStyleFormat object

Availability

Flash Player 6.

Usage

```
new FStyleFormat()
```

Parameters

None.

Returns

An instance of the FStyleFormat object.

Description

Method; creates a new FStyleFormat object. You create new FStyleFormat objects to define text and color properties for custom style formats used with custom components or with the Flash UI components. All Flash UI components are assigned by default to `globalStyleFormat`, which is an instance of the FStyleFormat object. You do not need to create a new instance of the FStyleFormat object to change properties in the global style format, because it already exists. You can also use `setStyleProperty` to change properties for specific component instances without using a constructor.

For more information, see the `setStyleProperty` method available to each component—`FCheckBox.setStyleProperty`, `FComboBox.setStyleProperty`, and so on. See also the “Customizing component colors and text” of the “Using Components” chapter of *Using Flash*.

Example

The following example creates the new style format `formStyleFormat`.

```
formStyleFormat = new StyleFormat();
```

FStyleFormat.addListener

Availability

Flash Player 6.

Usage

```
myStyleFormat.addListener(component1 [, component2, ...componentN])
```

Parameters

component1 ... componentN The component instances to register to *myStyleFormat*.

Returns

Nothing.

Description

Method; registers the specified components to *myStyleFormat*. Use this method to register instances of Flash UI components or custom components to a custom style format. You can also use this method with the following syntax to register a custom component to the global style format used by all Flash UI components by default:

```
globalStyleFormat.addListener(customComponent);
```

Example

The following code registers *formStyleFormat* with the components *myListBox*, *myComboBox*, and *myScrollBar*.

```
formStyleFormat.addListener(myListBox, myComboBox, myScrollBar);
```

See also

FStyleFormat.applyChanges, FStyleFormat.removeListener

FStyleFormat.applyChanges

Availability

Flash Player 6.

Usage

```
myStyleFormat.applyChanges([propertyName1, ...propertyNameN])  
myStyleFormat.applyChanges()
```

Parameters

propertyName1...propertyNameN A series of text strings specifying the properties to update for all components assigned to *myStyleFormat*.

Returns

Nothing.

Description

Method; updates the instance of the specified style format object and applies the changes to all the components assigned to the format. You must call this method when adding or removing listeners and setting or changing properties. When updating properties, it is recommended that you use the first syntax usage to update only those properties for which you are specifying a new value.

Usage 1: Updates only the properties specified in the parameters.

Usage 2: Updates all of the information in the style format (that is, assigned components and properties) whether they have changed or not.

Example

Usage 1: The following example updates the `arrow` and `background` properties, but not the `check` and `highlight` properties, in `formStyleFormat`.

```
formStyleFormat.arrow = 0x00ffaa;  
formStyleFormat.background = 0xaabbcc;  
formStyleFormat.check = 0x000000;  
formStyleFormat.highlight = 0xffffffff;  
formStyleFormat.applyChanges("arrow", "background");
```

Usage 2: The following example updates all the properties in `formStyleFormat`—`arrow`, `background`, `check`, and `highlight`.

```
formStyleFormat.arrow = 0x00ffaa;  
formStyleFormat.background = 0xaabbcc;  
formStyleFormat.check = 0x000000;  
formStyleFormat.highlight = 0xffffffff;  
formStyleFormat.applyChanges();
```

See also

`FStyleFormat.addListener`, `FStyleFormat.removeListener`

FStyleFormat.arrow

Availability

Flash Player 6.

Usage

myStyleFormat.arrow

Description

Property; the RGB color value for the `arrow` property used in scroll bars and drop-down lists in components such as scroll bars, list boxes, and combo boxes. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `0x800080` to the `arrow` property in `formStyleFormat`, producing a purple arrow.

```
formStyleFormat.arrow = 0x800080;
```

FStyleFormat.background

Availability

Flash Player 6.

Usage

myStyleFormat.background

Description

Property; the RGB color value for the background portion of a component. For example, in a radio button or check box, the background portion is the space inside the selection area; in a list box or combo box, the background portion is the display area. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `0xFFE4E1` to the `background` property for `formStyleFormat`, producing a pale pink background when the component is enabled.

```
formStyleFormat.background = 0xFFE4E1;
```

See also

`FStyleFormat.face`

FStyleFormat.backgroundDisabled

Availability

Flash Player 6.

Usage

myStyleFormat.backgroundDisabled

Description

Property; the RGB color value for the background portion of a disabled component. The background color of disabled user interface elements is usually light gray. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `0x808080` to the `backgroundDisabled` property for `formStyleFormat`, producing a gray background when the component is disabled.

```
formStyleFormat.backgroundDisabled = 0x808080;
```

See also

`FStyleFormat.foregroundDisabled`

FStyleFormat.check

Availability

Flash Player 6.

Usage

myStyleFormat.check

Description

Property; the RGB color value for the check mark in a selected check box. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `0x228B22` to the `check` property for `formStyleFormat`, producing a dark green arrow.

```
formStyleFormat.check = 0x228B22;
```

FStyleFormat.darkshadow

Availability

Flash Player 6.

Usage

```
myStyleFormat.darkshadow
```

Description

Property; the RGB color value for the inner border or darker shadow portion of a component—for example, the inner edge of an unselected radio button circle or unselected check box. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `0x0000CD` to the `darkshadow` property for `formStyleFormat`, producing a medium-blue inner-border color.

```
formStyleFormat.darkshadow = 0x0000CD;
```

See also

`FStyleFormat.highlight`, `FStyleFormat.shadow`

FStyleFormat.face

Availability

Flash Player 6.

Usage

```
myStyleFormat.face
```

Description

Property; the RGB color value for the main color of a component—for example, the gray used for the `PushButton` or `ScrollBar` component. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `0x32CD32` to the `face` property for `formStyleFormat`, producing lime-green push buttons and scroll bars.

```
formStyleFormat.face = 0x32CD32;
```

FStyleFormat.foregroundDisabled

Availability

Flash Player 6.

Usage

myStyleFormat.foregroundDisabled

Description

Property; the RGB color value for the foreground of a disabled component. The foreground color of disabled user interface elements is usually medium gray. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `0x708090` to the `foregroundDisabled` property for `formStyleFormat`, producing a gray foreground for disabled components.

```
formStyleFormat.foregroundDisabled = 0x708090;
```

See also

`FStyleFormat.backgroundDisabled`

FStyleFormat.highlight

Availability

Flash Player 6.

Usage

myStyleFormat.highlight

Description

Property; the RGB color value for the inner border or darker shadow portion of a component when it is selected—for example, the inner edge of a radio button circle or check box. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `0xFF00FF` to the `highlight` property for `formStyleFormat`, producing a bright-yellow inner border when the component is selected.

```
formStyleFormat.highlight = 0xFF00FF;
```

See also

`FStyleFormat.darkshadow`

FStyleFormat.highlight3D

Availability

Flash Player 6.

Usage

```
myStyleFormat.highlight3D
```

Description

Property; the RGB color value for the outer border or light-shadow portion of a component when it is selected—for example, the outer edge of a radio button circle or check box. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `0x40E0D0` to the `highlight3D` property for `formStyleFormat`, producing a bright-turquoise outer border when the component is selected.

```
formStyleFormat.highlight3D = 0x40E0D0;
```

See also

`FStyleFormat.shadow`

FStyleFormat.radioDot

Availability

Flash Player 6.

Usage

```
myStyleFormat.radioDot
```

Description

Property; the RGB color value for the selection dot in a radio button in a component. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `0xFF12AC` to the `radioDot` property for `formStyleFormat`, producing a pink selection dot.

```
formStyleFormat.radioDot = 0xFF12AC;
```

FStyleFormat.removeListener

Availability

Flash Player 6.

Usage

```
myStyleFormat.removeListener(component)
```

Parameters

component The component to remove from the style format.

Description

Method; removes a component assigned to the style format.

- If you remove a Flash UI component as a listener from the global style format and do not assign it (add it as a listener) to a custom style format, the skin element movie clips are displayed as they were originally authored by the component designer without a property value assigned.
- If you remove a Flash UI component as a listener from a custom style format, the component no longer uses the property values in the custom style format and instead uses the values specified for those properties in the global style format object.
- If you remove a custom component as a listener from a custom style format without adding it to a new custom style format, the component uses the values set for the properties in the global style format where possible, and otherwise displays the skin elements without a property value.

Example

The following example removes the component `check1` from `globalStyleFormat`.

```
globalStyleFormat.removeListener(check1);
```

See also

`FStyleFormat.addListener`, `FStyleFormat.applyChanges`

FStyleFormat.scrollTrack

Availability

Flash Player 6.

Usage

```
myStyleFormat.scrollTrack
```

Description

Property; the RGB color value for the track portion of a scroll bar. The `ScrollBar` component is used by the `ScrollPane`, `ListBox`, and `ComboBox` components, and changing the value of the `scrollTrack` property in the global style format changes the color of the scroll track in all components that use scroll bars. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `0xA0522D` to the `scrollTrack` property in `formStyleFormat`, producing a maroon scroll track.

```
formStyleFormat.scrollTrack = 0xA0522D;
```

FStyleFormat.selection

Availability

Flash Player 6.

Usage

myStyleFormat.selection

Description

Property; the RGB color value for the bar used to highlight the selected item in a component's list. This property works with the `FStyleFormat.textSelected` property to display selected items, and you should coordinate the colors to make text easy to read. For example, the global style format assigns a blue color value to the `selection` property used to display the selection bar in the `ListBox` and `ComboBox` components, and assigns a white color value to the `textSelected` property; this color combination allows the user to view selected text easily. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `0x87CEEB` to the `selection` property in `formStyleFormat`, producing a sky-blue selection bar.

```
formStyleFormat.selection = 0x87CEEB;
```

See also

`FStyleFormat.textSelected`

FStyleFormat.selectionDisabled

Availability

Flash Player 6.

Usage

myStyleFormat.selectionDisabled

Description

Property; the RGB color value for the selection bar used to highlight a list item in a disabled component. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `0x708090` to the `selectionDisabled` property for `formStyleFormat`, producing a slate-gray selection bar when the component is disabled.

```
formStyleFormat.selectionDisabled = 0x708090;
```

See also

`FStyleFormat.selection`

FStyleFormat.selectionUnfocused

Availability

Flash Player 6.

Usage

myStyleFormat.selectionUnfocused

Description

Property; the RGB color value for the selection bar (highlighting) in a component's list when the component doesn't have the keyboard focus. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `0xaabbcc` to the `selectionUnfocused` property for `formStyleFormat`.

```
formStyleFormat.selectionUnfocused = 0xaabbcc;
```

See also

`FStyleFormat.selection`

FStyleFormat.shadow

Availability

Flash Player 6.

Usage

myStyleFormat.shadow

Description

Property; the RGB color value for the outer border or light shadow portion of a component—for example, the outer edge of an unselected radio button circle or unselected check box. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `0x008080` to the `shadow` property for `formStyleFormat`, producing a teal outer border for unselected radio button and check box components.

```
formStyleFormat.shadow = 0x008080;
```

See also

`FStyleFormat.check`

FStyleFormat.textAlign

Availability

Flash Player 6.

Usage

```
myStyleFormat.textAlign
```

Description

Property; a text string specifying right, left, or center alignment for text displayed in all components assigned to the style format. The default setting is `left`.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code right-aligns all text in components using `formStyleFormat`.

```
formStyleFormat.textAlign = "right";
```

See also

`FStyleFormat.textIndent`, `FStyleFormat.textLeftMargin`,
`FStyleFormat.textRightMargin`

FStyleFormat.textBold

Availability

Flash Player 6.

Usage

```
myStyleFormat.textBold
```

Description

Property; a Boolean value specifying whether all text displayed in components using the style format is bold (`true`) or not (`false`). The default setting is `false`.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code specifies that all text displayed in components assigned to `formStyleFormat` is bold.

```
formStyleFormat.textBold = true;
```

See also

`FStyleFormat.textItalic`, `FStyleFormat.textUnderline`

FStyleFormat.textColor

Availability

Flash Player 6.

Usage

myStyleFormat.textColor

Description

Property; the RGB color value for the default text color in all components assigned to the style format. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of 0x000000 to the `textColor` property in `formStyleFormat`, producing black text.

```
formStyleFormat.textColor = 0x000000;
```

See also

`FStyleFormat.textDisabled`, `FStyleFormat.textSelected`

FStyleFormat.textDisabled

Availability

Flash Player 6.

Usage

myStyleFormat.textDisabled

Description

Property; the RGB color value for the default text color used to display text in disabled components assigned to the style format. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of 0xC0C0C0 to the `textDisabled` property for `formStyleFormat`, producing silver text when the component is disabled.

```
formStyleFormat.textDisabled = 0xC0C0C0;
```

See also

`FStyleFormat.textAlign`, `FStyleFormat.textSelected`

FStyleFormat.textFont

Availability

Flash Player 6.

Usage

myStyleFormat.textFont

Description

Property; a text string specifying the font used to display text in all components assigned to the style format.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `Arial` to the `textFont` property for `formStyleFormat`.

```
formStyleFormat.textFont = "Arial";
```

FStyleFormat.textIndent

Availability

Flash Player 6.

Usage

```
myStyleFormat.textIndent
```

Description

Property; an integer specifying the indentation, in pixels, from the left margin to the first text character for all text displayed using the style format.

Example

The following code indents all text displayed by `formStyleFormat` 5 pixels.

```
formStyleFormat.textIndent = 5;
```

See also

`FStyleFormat.textAlign`, `FStyleFormat.textLeftMargin`

FStyleFormat.textItalic

Availability

Flash Player 6.

Usage

```
myStyleFormat.textItalic
```

Description

Property; a Boolean value specifying whether all text displayed in components using the style format is italic (`true`) or not (`false`). The default setting is `false`.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code specifies that all text displayed in components assigned to `formStyleFormat` is italic.

```
formStyleFormat.textItalic = true;
```

See also

`FStyleFormat.textBold`

FStyleFormat.textLeftMargin

Availability

Flash Player 6.

Usage

myStyleFormat.textLeftMargin

Description

Property; an integer specifying the left paragraph margin, in pixels, for all text displayed in components assigned to the style format.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code specifies a value of 4 pixels for the `textLeftMargin` property of `formStyleFormat`.

```
formStyleFormat.textLeftMargin = 4;
```

See also

`FStyleFormat.textRightMargin`

FStyleFormat.textRightMargin

Availability

Flash Player 6.

Usage

myStyleFormat.textRightMargin

Property; an integer specifying the right paragraph margin, in pixels, for all text displayed in components assigned to the style format.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code specifies a value of 4 pixels for the `textRightMargin` property of `formStyleFormat`.

```
formStyleFormat.textRightMargin = 4;
```

See also

`FStyleFormat.textLeftMargin`

FStyleFormat.textSelected

Availability

Flash Player 6.

Usage

myStyleFormat.textSelected

Description

Property; an RGB color value specifying the color of selected text in components assigned to the style format. This property works with the `FStyleFormat.selection` property to display selected list items, and you should coordinate the colors to make the text easy to read. For example, the global style format assigns a blue color value to the `selection` property used to display the selection bar in the `ListBox` and `ComboBox` components, and assigns a white color value to the `textSelected` property; this color combination allows the user to view selected text easily. The color value must be in the format *0xRRGGBB*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code assigns a value of `0xffffffff` to the `textSelected` property for `formStyleFormat`, producing white text when the component is selected.

```
formStyleFormat.textSelected = 0xffffffff;
```

See also

`FStyleFormat.selection`, `FStyleFormat.textDisabled`

FStyleFormat.textSize

Availability

Flash Player 6.

Usage

myStyleFormat.textSize

Description

Property; an integer specifying the point size of text displayed in components assigned to the style format. The default setting for this property is 12-point text.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code specifies 10-point text for all components assigned to `formStyleFormat`.

```
formStyleFormat.textSize = 10;
```

See also

`FStyleFormat.textFont`

FStyleFormat.textUnderline

Availability

Flash Player 6.

Usage

myStyleFormat.textUnderline

Description

Property; specifies whether text displayed in components using the specified style format is underlined (*true*) or not (*false*). The default setting is *false*.

You must use `FStyleFormat.applyChanges` when updating properties with a new value.

Example

The following code specifies that all text displayed in components assigned to `formStyleFormat` is underlined.

```
formStyleFormat.textUnderline = true;
```

See also

`FStyleFormat.textBold`, `FStyleFormat.textItalic`

Function (object)

The Function object is available in Flash MX.

Property summary for the Function object

Method	Description
<code>Function.prototype</code>	Refers to an object that is the prototype for a class.

Method summary for the Function object

Method	Description
<code>Function.apply</code>	Enables ActionScript code to call a function.
<code>Function.call</code>	Invokes the function represented by a Function object.

Function.apply

Availability

Flash Player 6.

Usage

myFunction.apply(thisObject, argumentsObject)

Parameters

thisObject The object that *myFunction* is applied to.

argumentsObject An array whose elements are passed to *myFunction* as parameters.

Returns

Any value that the called function specifies.

Description

Method; specifies the value of `this` to be used within any function that ActionScript calls. This method also specifies the parameters to be passed to any called function. Because `apply` is a method of the Function object, it is also a method of every function object in ActionScript.

The parameters are specified as an Array object. This is often useful when the number of parameters to be passed is not known until the script actually executes.

Example

The following function invocations are equivalent:

```
Math.atan2(1, 0)
Math.atan2.apply(null, [1, 0])
```

You could construct a Flash movie that contains input entry fields that permit the user to enter the name of a function to invoke, and zero or more parameters to pass to the function. Pressing a “Call” button would then use the `apply` method to call the function, specifying the parameters.

In this example, the user specifies a function name in an input text field called `functionName`. The number of parameters is specified in an input text field called `numParameters`. Up to 10 parameters are specified in text fields called `parameter1`, `parameter2`, up to `parameter10`.

```
on (release) {
    callTheFunction();
}
...
function callTheFunction()
{
    var theFunction = eval(functionName.text);
    var n = Number(numParameters);
    var parameters = [];
    for (var i = 0; i < n; i++) {
        parameters.push(eval("parameter" + i));
    }
    theFunction.apply(null, parameters);
}
```

Function.call

Availability

Flash Player 6.

Usage

```
myFunction.call(thisObject, parameter1, ..., parameterN)
```

Parameters

thisObject Specifies the value of `this` within the function body.

parameter1 A parameter to be passed to the *myFunction*. You can specify zero or more parameters.

parameterN

Returns

Nothing.

Description

Method; invokes the function represented by a Function object. Every function in ActionScript is represented by a Function object, so all functions support the `call` method.

In almost all cases, the function call operator (`()`) may be used instead of the `call` method. The function call operator makes for code that is concise and readable. The `call` method is primarily useful when the `this` parameter of the function invocation needs to be explicitly controlled.

Normally, if a function is invoked as a method of an object, within the body of the function, `this` is set to `myObject` as in the following:

```
myObject.myMethod(1, 2, 3);
```

In some situations, you may want `this` to point somewhere else; for example, if a function must be invoked as a method of an object, but is not actually stored as a method of that object.

```
myObject.myMethod.call(myOtherObject, 1, 2, 3);
```

You can pass the value `null` for the *thisObject* parameter to invoke a function as a regular function and not as a method of an object. For example, the following function invocations are equivalent:

```
Math.sin(Math.PI / 4)
Math.sin.call(null, Math.PI / 4)
```

Example

This example uses the `call` method to make a function behave as a method of another object, without storing the function in the object.

```
function MyObject() {
}
function MyMethod(obj) {
    trace("this == obj? " + (this == obj));
}
var obj = new MyObject();
MyMethod.call(obj, obj);
```

The trace action sends the following code to the Output window:

```
this == obj? true
```

Function.prototype

Availability

Flash Player 6.

Usage

```
myFunction.prototype
```

Description

Property; in a constructor function, the `prototype` property refers to an object that is the prototype of the constructed class. Each instance of the class that is created by the constructor function inherits all the properties and methods from the prototype object.

fscommand

Availability

Flash Player 3.

Usage

```
fscommand("command", "parameters")
```

Parameters

command A string passed to the host application for any use or a command passed to the stand-alone Flash Player.

parameters A string passed to the host application for any use or a value passed to the Flash Player.

Returns

Nothing.

Description

Action; allows the Flash movie to communicate with either the Flash Player, or the program hosting the Flash Player, such as a Web browser. You can also use the `fscommand` action to pass messages to Macromedia Director, or to Visual Basic, Visual C++, and other programs that can host ActiveX controls.

Usage 1: To send a message to the Flash Player, you must use predefined commands and parameters. The following table shows the values you can specify for the *command* and *parameters* parameters of the `fscommand` action to control a movie playing in the stand-alone Flash player (including projectors):

Command	Parameters	Purpose
quit	None	Closes the projector.
fullscreen	true or false	Specifying true sets the Flash Player to full-screen mode. Specifying false returns the player to normal menu view.
allowscale	true or false	Specifying false sets the player so that the movie is always drawn at its original size and never scaled. Specifying true forces the movie to scale to 100% of the player.
showmenu	true or false	Specifying true enables the full set of context menu items. Specifying false dims all the context menu items except About Flash Player.
exec	Path to application	Executes an application from within the projector.
trapallkeys	true or false	Specifying true sends all key events, including accelerator keys, to the <code>onClipEvent(keyDown/keyUp)</code> handler in the Flash Player.

Usage 2: To use the `fscommand` action to send a message to a scripting language such as JavaScript in a Web browser, you can pass any two parameters in the *command* and *parameters* parameters. These parameters can be strings or expressions and are used in a JavaScript function that “catches,” or handles, the `fscommand` action.

In a Web browser, the `fscommand` action calls the JavaScript function `myMovie_DoFSCommand` in the HTML page containing the Flash movie. The `myMovie` is the name of the Flash Player as assigned by the `NAME` attribute of the `EMBED` tag or the `ID` property of the `OBJECT` tag. If you assign the Flash Player the name `myMovie`, the JavaScript function called is `myMovie_DoFSCommand`.

Usage 3: The `fscommand` action can send messages to Macromedia Director that are interpreted by Lingo as strings, events, or executable Lingo code. If the message is a string or an event, you must write the Lingo code to receive the message from the `fscommand` action and carry out an action in Director.

Usage 4: In Visual Basic, Visual C++, and other programs that can host ActiveX controls, `fscommand` sends a VB event with two strings that can be handled in the environment's programming language. For more information, use the keywords `Flash` method to search the Flash Support Center.

Example

Usage 1: In the following example, the `fscommand` action sets the Flash Player to scale the movie to the full monitor screen size when the button is released.

```
on(release){  
    fscommand("fullscreen", true);  
}
```

Usage 2: The following example uses the `fscommand` action applied to a button in Flash to open a JavaScript message box in an HTML page. The message itself is sent to JavaScript as the `fscommand` parameter.

You must add a function to the HTML page that contains the Flash movie. This function, `myMovie_DoFSCommand` sits in the HTML page and waits for an `fscommand` action in Flash. When an `fscommand` is triggered in Flash (for example, when a user presses the button), the command and parameter strings are passed to the `myMovie_DoFSCommand` function. You can use the passed strings in your JavaScript or VBScript code in any way you like. In this example, the function contains a conditional `if` statement that checks to see if the command string is "messagebox". If it is, a JavaScript alert box (or "messagebox") opens and displays the contents of the parameters string.

```
function myMovie_DoFSCommand(command, args) {  
    if (command == "messagebox") {  
        alert(args);  
    }  
}
```

In the Flash document, add the `fscommand` action to a button:

```
fscommand("messagebox", "This is a message box called from within Flash.")
```

You can also use expressions for the `fscommand` action and parameters, as in the following example:

```
fscommand("messagebox", "Hello, " + name + ", welcome to our Web site!")
```

To test the movie, choose **File > Publish Preview > HTML**.

Note: If you publish your movie using the Flash with `FSCCommand` template in the HTML Publish Settings, the `myMovie_DoFSCommand` function is inserted automatically. The movie's `NAME` and `ID` attributes will be the filename. For example, for the file `myMovie.fl`, the attributes would be set to `myMovie`.

function

Availability

Flash Player 5.

Usage

```
function functionname ([parameter0, parameter1,...parameterN]){  
    statement(s)  
}  
function ([parameter0, parameter1,...parameterN]){  
    statement(s)  
}
```

Parameters

functionname The name of the new function.

parameter An identifier that represents a parameter to pass to the function. These parameters are optional.

statement(s) Any ActionScript instruction you have defined for the body of the function.

Returns

Nothing.

Description

Action; a set of statements that you define to perform a certain task. You can *declare*, or define, a function in one location and call, or invoke, it from different scripts in a movie. When you define a function, you can also specify parameters for the function. Parameters are placeholders for values on which the function operates. You can pass different parameters to a function each time you call it. This lets you reuse one function in many different situations.

Use the `return` action in a function's *statement(s)* to cause a function to return, or generate, a value.

Usage 1: Declares a function with the specified *functionname*, *parameters*, and *statement(s)*. When a function is called, the function declaration is invoked. Forward referencing is permitted; within the same Action list, a function may be declared after it is called. A function declaration replaces any prior declaration of the same function. You can use this syntax wherever a statement is permitted.

Usage 2: Creates an anonymous function and returns it. This syntax is used in expressions, and is particularly useful for installing methods in objects.

Example

Usage 1: The following example defines the function `sqr`, which accepts one parameter and returns the `square(x*x)` of the parameter. Note that if the function is declared and used in the same script, the function declaration may appear after using the function.

```
y=sqr(3);  
  
function sqr(x) {  
    return x*x;  
}
```

Usage 2: The following function defines a Circle object:

```
function Circle(radius) {  
    this.radius = radius;  
}
```

The following statement defines an anonymous function that calculates the area of a circle and attaches it to the object Circle as a method:

```
Circle.prototype.area = function () {return Math.PI * this.radius *  
    this.radius}
```

ge (greater than or equal to—string specific)

Availability

Flash Player 4. This operator has been deprecated in Flash 5 in favor of the >= (greater than or equal to) operator.

Usage

expression1 ge *expression2*

Parameters

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

Returns

Nothing.

Description

Operator (comparison); compares the string representation of *expression1* to the string representation of *expression2* and returns *true* if *expression1* is greater than or equal to *expression2*; otherwise, returns *false*.

See also

>= (greater than or equal to)

getProperty

Availability

Flash Player 4.

Usage

getProperty(*instancename* , *property*)

Parameters

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

property A property of a movie clip.

Returns

Nothing.

Description

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

Example

The following example retrieves the horizontal axis coordinate (`_x`) for the movie clip `myMovie` and assigns it to the variable `myMovieX`:

```
myMovieX = getProperty(_root.myMovie, _x);
```

getTimer

Availability

Flash Player 4.

Usage

```
getTimer()
```

Parameters

None.

Returns

Nothing.

Description

Function; returns the number of milliseconds that have elapsed since the movie started playing.

getURL

Availability

Flash 2. The `GET` and `POST` options are only available to Flash Player 4 and later versions of the Player.

Usage

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

Parameters

url The URL from which to obtain the document.

window An optional parameter specifying the window or HTML frame that the document should load into. You can enter the name of a specific window or choose from the following reserved target names:

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

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

Returns

Nothing.

Description

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

Example

This example loads a new URL into a blank browser window. The `getURL` action targets the variable `incomingAd` as the `url` parameter so that you can change the loaded URL without having to edit the Flash movie. The `incomingAd` variable's value is passed into Flash earlier in the movie using a `loadVariables` action.

```
on(release) {  
    getURL(incomingAd, "_blank");  
}
```

See also

`loadVariables`, `XML.send`, `XML.sendAndLoad`, `XMLSocket.send`

getVersion

Availability

Flash Player 5.

Usage

```
getVersion()
```

Parameters

None.

Returns

Nothing.

Description

Function; returns a string containing Flash Player version and platform information.

The `getVersion` function only returns information for Flash Player 5 or later versions of the Player.

Example

The following is an example of a string returned by the `getVersion` function.

```
WIN 5,0,17,0
```

This indicates that the platform is Windows, and the version number of the Flash Player is major version 5, minor version 17(5.0r17).

_global

Availability

Flash Player 6.

Usage

`_global.identifier`

Parameters

None.

Returns

A reference to the global object that holds the core ActionScript classes, such as String, Object, Math, and Array.

Description

Identifier; creates global variables, objects, or classes. For example, you could create a library that is exposed as a global ActionScript object, much like the Math or Date object. Unlike Timeline-declared or locally-declared variables and functions, global variables and functions are visible to every Timeline and scope in the Flash movie, provided they are not obscured by identifiers with the same names in inner scopes.

Example

The following example creates a top-level function `factorial` that is available to every Timeline and scope in a Flash movie:

```
_global.factorial = function (n) {  
    if (n <= 1) {  
        return 1;  
    } else {  
        return n * factorial(n-1);  
    }  
}
```

See also

`var`, `set variable`

globalStyleFormat

Availability

Flash Player 6.

Usage

`globalStyleFormat.styleProperty`

Parameters

styleProperty A property of the FStyleFormat object.

Returns

Nothing.

Description

Object instance; an instance of the `FStyleFormat` object that defines the style format properties for Flash UI components. The `globalStyleFormat` instance is available once a Flash UI component is placed on the Stage. You set or change style format properties for Flash UI components by editing the properties in the `globalStyleFormat` object instance. For more information, see “Customizing component colors and text” in the “Using Components” chapter of *Using Flash*.

Example

The following code sets the `arrow` property of the `FStyleFormat` property for the `globalStyleFormat` instance.

```
globalStyleFormat.arrow = 0x800080;
```

See also

`FStyleFormat` (object)

gotoAndPlay

Availability

Flash 2.

Usage

```
gotoAndPlay(scene, frame)
```

Parameters

scene The scene name to which the playhead is sent.

frame The frame number or label to which the playhead is sent.

Returns

Nothing.

Description

Action; sends the playhead to the specified frame in a scene and plays from that frame. If no scene is specified, the playhead goes to the specified frame in the current scene.

Example

When the user clicks a button to which the `gotoAndPlay` action is assigned, the playhead is sent to Frame 16 and starts to play.

```
on(release) {  
    gotoAndPlay(16);  
}
```

gotoAndStop

Availability

Flash 2.

Usage

```
gotoAndStop(scene, frame)
```

Parameters

scene The scene name to which the playhead is sent.

frame The frame number or label to which the playhead is sent.

Returns

Nothing.

Description

Action; sends the playhead to the specified frame in a scene and stops it. If no scene is specified, the playhead is sent to the frame in the current scene.

Example

When the user clicks a button that the `gotoAndStop` action is assigned to, the playhead is sent to frame 5 and the movie stops playing.

```
on(release) {  
    gotoAndStop(5);  
}
```

gt (greater than –string specific)

Availability

Flash Player 4. This operator has been deprecated in Flash 5 in favor of the new `>` (greater than) operator.

Usage

```
expression1 gt expression2
```

Parameters

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

Description

Operator (comparison); compares the string representation of *expression1* to the string representation of *expression2* and returns `true` if *expression1* is greater than *expression2*; otherwise, returns `false`.

See also

`>` (greater than)

_highquality

Availability

Flash Player 4.

Usage

`_highquality`

Description

Property (global); specifies the level of anti-aliasing applied to the current movie. Specify 2 (BEST) to apply high quality with bitmap smoothing always on. Specify 1 (high quality) to apply anti-aliasing; this will smooth bitmaps if the movie does not contain animation. Specify 0 (low quality) to prevent anti-aliasing.

Example

```
_highquality = 1;
```

See also

`_quality`, `toggleHighQuality`

if

Availability

Flash Player 4.

Usage

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

Parameters

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

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

Returns

Nothing.

Description

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

Example

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

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

Example

The following example uses an `if` action to evaluate when a draggable object in the movie is released by the user. If the object was released less than 300 milliseconds after dragging it, the condition evaluates to `true` and the statements inside the curly brackets run. Those statements set variables to store the new location of the object, how hard it was thrown, and the speed at which it was thrown. The `timePressed` variable is also reset. If the object was released more than 300 milliseconds after it was dragged, the condition evaluates to `false` and none of the statements run.

```
if (getTimer()<timePressed+300) {  
    // if the condition is true,  
    // the object was thrown.  
    // what is the new location of this object?  
    xNewLoc = this._x;  
    yNewLoc = this._y;  
    // how hard did they throw it?  
    xTravel = xNewLoc-xLoc;  
    yTravel = yNewLoc-yLoc;  
    // setting the speed of the object depending on  
    // how far they travelled with it  
    xInc = xTravel/2;  
    yInc = yTravel/2;  
    timePressed = 0;  
}
```

See also

`else`

ifFrameLoaded

Availability

Flash Player 3. The `ifFrameLoaded` action is deprecated in Flash 5; use of the `MovieClip._framesloaded` action is encouraged.

Usage

```
ifFrameLoaded(scene, frame) {  
    statement;  
}  
  
ifFrameLoaded(frame) {  
    statement;  
}
```

Parameters

scene The scene that must be loaded.

frame The frame number or frame label that must be loaded before the next statement is executed.

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

Returns

Nothing.

Description

Action; checks whether the contents of a specific frame are available locally. Use `ifFrameLoaded` to start playing a simple animation while the rest of the movie downloads to the local computer. The difference between using `_framesloaded` and `ifFrameLoaded` is that `_framesloaded` allows you to add your own `if` or `else` statements.

See also

`MovieClip._framesloaded`

#include

Availability

N/A

Usage

```
#include "filename.as"
```

Parameters

filename.as The filename for the script to add to the Actions panel; `.as` is the recommended file extension.

Returns

Nothing.

Description

Action; includes the contents of the file specified in the parameter when the movie is tested, published, or exported. The `#include` action is invoked when you test, publish, or export. The `#include` action is checked when a syntax check occurs.

#initclip

Availability

Flash Player 6.

Usage

```
#initclip order
```

Parameters

order An integer that specifies the execution order of blocks of `#initclip` code. This is an optional parameter.

Description

Action; indicates the start of a block of component initialization actions. When multiple clips are initialized at the same time you can use the *order* parameter to specify which initialization occurs first. Component initialization actions execute when a movie clip symbol is defined. If the movie clip is an exported symbol, the component initialization actions execute before the actions on Frame 1 of the SWF file. Otherwise, they execute immediately before the frame actions of the frame that contains the first instance of the associated movie clip symbol.

Component initialization actions execute only once during the playback of a movie and you should use them for one-time initializations, such as class definition and registration.

Example

The following example code is assigned to the first frame of a movie that is a check box component. The `#initclip` and `#endinitclip` actions designate the block of statements they enclose as component initialization actions. The enclosed statements register the class and store methods in a prototype object.

```
#initclip
if (typeof(CheckBox) == "undefined") {
    // Define constructor for (and thus define) CheckBox class
    function CheckBox() {
        // Set up our data bindings
        this.watch('value', function (id, oldval, newval) { ... });
        this.watch('label', function (id, oldval, newval) { ... });
    }
    // Set CheckBox prototype chain to inherit from MovieClip
    CheckBox.prototype = new MovieClip();
    // Register CheckBox as the class for symbol "Check Box"
    Object.registerClass("Check Box", CheckBox);
    // Set up some methods
    CheckBox.prototype.enable = function () { ... };
    CheckBox.prototype.show = function () { ... };
    CheckBox.prototype.hide = function () { ... };
    // Set up a convenience function to instantiate
    // check boxes
    CheckBox.create = function (parentMovieClip, instanceName, depth) {
        parentMovieClip.attachMovie("CheckBox", instanceName, depth);
    };
}
#endinitclip
```

Note: If you copy and paste this code into the Actions panel, it will generate an error when the script is compiled because of the undefined functions (`{ ... }`)

See also

```
#endinitclip
```

instanceof

Availability

Flash Player 6.

Usage

object instanceof *class*

Parameters

object An ActionScript object.

class A reference to an ActionScript constructor function, such as `String` or `Date`.

Returns

If *object* is an instance of *class*, `instanceof` returns `true`; otherwise, `instanceof` returns `false`.

Description

Operator; determines whether an object belongs to a specified class. Tests if *object* is an instance of *class*.

An ActionScript object is said to be an instance of a class if the constructor function's prototype object is in the ActionScript object's prototype chain.

The `instanceof` operator does not convert primitive types to wrapper objects. For example, the following code returns `true`:

```
new String("Hello") instanceof String
```

Whereas the following code returns `false`:

```
"Hello" instanceof String
```

Example

To illustrate the algorithm of the `instanceof` operator, the following example shows how the `instanceof` operator might look if it was coded as an ActionScript function.

```
function instanceof (theObject, theClass){
    while ((theObject = theObject.__proto__) != null) {
        if (theObject == theClass.prototype) {
            return true;
        }
    }
    return false;
}
```

See also

`typeof`

int

Availability

Flash Player 4. This function has been deprecated in Flash 5 in favor of the `Math.round` method.

Usage

```
int(value)
```

Parameters

value A number to be rounded to an integer.

Returns

Nothing.

Description

Function; converts a decimal number to the closest integer value.

See also

`Math.floor`

isFinite

Availability

Flash Player 5.

Usage

```
isFinite(expression)
```

Parameters

expression A Boolean, variable, or other expression to be evaluated.

Returns

Nothing.

Description

Top-level function; evaluates the *expression* and returns `true` if it is a finite number, and `false` if it is infinity or negative infinity. The presence of infinity or negative infinity indicates a mathematical error condition such as division by 0.

Example

The following are examples of return values for `isFinite`:

```
isFinite(56)
// returns true

isFinite(Number.POSITIVE_INFINITY)
// returns false
```

isNaN

Availability

Flash Player 5.

Usage

```
isNaN(expression)
```

Parameters

expression A Boolean, variable, or other expression to be evaluated.

Returns

Nothing.

Description

Top-level function; evaluates the parameter and returns `true` if the value is not a number (NaN), indicating the presence of mathematical errors.

Example

The following code illustrates return values for the `isNaN` function.

```
isNaN("Tree")
// returns true

isNaN(56)
// returns false

isNaN(Number.POSITIVE_INFINITY)
// returns false
```

Key (object)

The Key object is a top-level object that you can access without using a constructor. Use the methods of the Key object to build an interface that can be controlled by a user with a standard keyboard. The properties of the Key object are constants representing the keys most commonly used to control games. For a complete list of key code values, see the appendix “Keyboard Keys and Key Code Values” in *Using Flash*.

Example

The following script uses the Key object to identify keys on any keyboard so that the user can control a movie clip.

```
onClipEvent (enterFrame) {  
    if(Key.isDown(Key.RIGHT)) {  
        this._x=_x+10;  
    } else if (Key.isDown(Key.DOWN)) {  
        this._y=_y+10;  
    }  
}
```

Method summary for the Key object

Method	Description
Key.addListener	Registers an object to receive notification when the onKeyDown and onKeyUp methods are invoked.
Key.getAscii	Returns the ASCII value of the last key pressed.
Key.getCode	Returns the virtual key code of the last key pressed.
Key.isDown	Returns true if the key specified in the parameter is pressed.
Key.isToggled	Returns true if the Num Lock or Caps Lock key is activated.
Key.removeListener	Removes an object that was previously registered with addListener.

Property summary for the Key object

All of the properties for the Key object are constants.

Property	Description
Key.BACKSPACE	Constant associated with the key code value for the Backspace key (8).
Key.CAPSLock	Constant associated with the key code value for the Caps Lock key (20).
Key.CONTROL	Constant associated with the key code value for the Control key (17).
Key.DELETEKEY	Constant associated with the key code value for the Delete key (46).
Key.DOWN	Constant associated with the key code value for the Down Arrow key (40).
Key.END	Constant associated with the key code value for the End key (35).
Key.ENTER	Constant associated with the key code value for the Enter key (13).
Key.ESCAPE	Constant associated with the key code value for the Escape key (27).
Key.HOME	Constant associated with the key code value for the Home key (36).
Key.INSERT	Constant associated with the key code value for the Insert key (45).
Key.LEFT	Constant associated with the key code value for the Left Arrow key (37).

Property	Description
Key.PGDN	Constant associated with the key code value for the Page Down key (34).
Key.PGUP	Constant associated with the key code value for the Page Up key (33).
Key.RIGHT	Constant associated with the key code value for the Right Arrow key (39).
Key.SHIFT	Constant associated with the key code value for the Shift key (16).
Key.SPACE	Constant associated with the key code value for the Spacebar (32).
Key.TAB	Constant associated with the key code value for the Tab key (9).
Key.UP	Constant associated with the key code value for the Up Arrow key (38).

Listener summary for the Key object

Method	Description
Key.onKeyDown	Notified when a key is pressed.
Key.onKeyUp	Notified when a key is released.

Key.addListener

Availability

Flash Player 6.

Usage

Key.addListener (*newListener*)

Parameters

newListener An object with methods `onKeyDown` and `onKeyUp`.

Returns

Nothing.

Description

Method; registers an object to receive `onKeyDown` and `onKeyUp` notification. When a key is pressed or released, regardless of the input focus, all listening objects registered with `addListener` have either their `onKeyDown` method or `onKeyUp` method invoked. Multiple objects can listen for keyboard notifications. If the listener *newListener* is already registered, no change occurs.

Example

This example creates a new listener object and defines a function for `onKeyDown` and `onKeyUp`. The last line uses the `addListener` method to register the listener with the Key object so that it can receive notification from the key down and key up events.

```
myListener = new Object();
myListener.onKeyDown = function () {
    trace ("You pressed a key.");
}
myListener.onKeyUp = function () {
    trace ("You released a key.");
}
Key.addListener(myListener);
```

Key.BACKSPACE

Availability

Flash Player 5.

Usage

Key.BACKSPACE

Description

Property; constant associated with the key code value for the Backspace key (8).

Key.CAPSLOCK

Availability

Flash Player 5.

Usage

Key.CAPSLOCK

Description

Property; constant associated with the key code value for the Caps Lock key (20).

Key.CONTROL

Availability

Flash Player 5.

Usage

Key.CONTROL

Description

Property; constant associated with the key code value for the Control key (17).

Key.DELETEKEY

Availability

Flash Player 5.

Usage

Key.DELETEKEY

Description

Property; constant associated with the key code value for the Delete key (46).

Key.DOWN

Availability

Flash Player 5.

Usage

Key.DOWN

Description

Property; constant associated with the key code value for the Down Arrow key (40).

Key.END

Availability

Flash Player 5.

Usage

`Key.END`

Description

Property; constant associated with the key code value for the End key (35).

Key.ENTER

Availability

Flash Player 5.

Usage

`Key.ENTER`

Description

Property; constant associated with the key code value for the Enter key (13).

Key.ESCAPE

Availability

Flash Player 5.

Usage

`Key.ESCAPE`

Description

Property; constant associated with the key code value for the Escape key (27).

Key.getAscii

Availability

Flash Player 5.

Usage

`Key.getAscii();`

Parameters

None.

Returns

Nothing.

Description

Method; returns the ASCII code of the last key pressed or released. The ASCII values returned are English keyboard values. For example, if you press Shift+2, `Key.getAscii` returns @ on a Japanese keyboard, just as it does on an English keyboard.

Key.getCode

Availability

Flash Player 5.

Usage

```
Key.getCode();
```

Parameters

None.

Returns

Nothing.

Description

Method; returns the key code value of the last key pressed. To match the returned key code value with the virtual key on a standard keyboard, use the information in Appendix B, “Keyboard Keys and Key Code Values” of *Using Flash*.

Key.HOME

Availability

Flash Player 5.

Usage

```
Key.HOME
```

Description

Property; constant associated with the key code value for the Home key (36).

Key.INSERT

Availability

Flash Player 5.

Usage

```
Key.INSERT
```

Description

Property; constant associated with the key code value for the Insert key (45).

Key.isDown

Availability

Flash Player 5.

Usage

```
Key.isDown(keycode);
```

Parameters

keycode The key code value assigned to a specific key, or a Key object property associated with a specific key. For a list of all of the key codes associated with the keys on a standard keyboard, see Appendix B, “Keyboard Keys and Key Code Values” of *Using Flash*.

Returns

Nothing.

Description

Method; returns `true` if the key specified in *keycode* is pressed. On the Macintosh, the key code values for the Caps Lock and Num Lock keys are identical.

Key.isToggled

Availability

Flash Player 5.

Usage

```
Key.isToggled(keycode)
```

Parameters

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

Returns

Nothing.

Description

Method; returns `true` if the Caps Lock or Num Lock key is activated (toggled). On the Macintosh, the key code values for these keys are identical.

Key.LEFT

Availability

Flash Player 5.

Usage

```
Key.LEFT
```

Description

Property; constant associated with the key code value for the Left Arrow key (37).

Key.onKeyDown

Availability

Flash Player 6.

Usage

```
someListener.onKeyDown
```

Description

Listener; notified when a key is pressed. To use `onKeyDown` you must create a listener object. You can then define a function for `onKeyDown` and use the `addListener` method to register the listener with the `Key` object, as in the following:

```
someListener = new Object();
someListener.onKeyDown = function () { ... };
Key.addListener(someListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

See also

`Key.addListener`

Key.onKeyUp

Availability

Flash Player 6.

Usage

```
someListener.onKeyUp
```

Description

Listener; notified when a key is released. To use `onKeyUp` you must create a listener object. You can then define a function for `onKeyUp` and use the `addListener` method to register the listener with the `Key` object, as in the following:

```
someListener = new Object();
someListener.onKeyUp = function () { ... };
Key.addListener(someListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

See also

`Key.addListener`

Key.PGDN

Availability

Flash Player 5.

Usage

```
Key.PGDN
```

Description

Property; constant associated with the key code value for the Page Down key (34).

Key.PGUP

Availability

Flash Player 5.

Usage

`Key.PGUP`

Description

Property; constant associated with the key code value for the Page Up key (33).

Key.removeListener

Availability

Flash Player 6.

Usage

`Key.removeListener (listener)`

Parameters

`listener` An object.

Returns

If the *listener* was successfully removed, the method returns `true`. If the *listener* was not successfully removed, for example if the *listener* was not on the Key object's listener list, the method returns `false`.

Description

Method; removes an object previously registered with the `addListener` method.

Key.RIGHT

Availability

Flash Player 5.

Usage

`Key.RIGHT`

Description

Property; constant associated with the key code value for the Right Arrow key (39).

Key.SHIFT

Availability

Flash Player 5.

Usage

`Key.SHIFT`

Description

Property; constant associated with the key code value for the Shift key (16).

Key.SPACE

Availability

Flash Player 5.

Usage

`Key.SPACE`

Description

Property; constant associated with the key code value for the Spacebar (32).

Key.TAB

Availability

Flash Player 5.

Usage

`Key.TAB`

Description

Property; constant associated with the key code value for the Tab key (9).

Key.UP

Availability

Flash Player 5.

Usage

`Key.UP`

Description

Property; constant associated with the key code value for the Up Arrow key (38).

le (less than or equal to – string specific)

Availability

Flash Player 4. This operator has been deprecated in Flash 5 in favor of the `<=` (less than or equal to) operator.

Usage

expression1 le expression2

Parameters

expression1, expression2 Numbers, strings, or variables.

Returns

Nothing.

Description

Operator (comparison); compares *expression1* to *expression2* and returns a value of `true` if *expression1* is less than or equal to *expression2*; otherwise, it returns a `false` value.

See also

`<=` (less than or equal to)

length

Availability

Flash Player 4. This function, along with all of the string functions, has been deprecated in Flash 5. It is recommended that you use the methods and `length` property of the `String` object to perform the same operations.

Usage

`length(expression)`

`length(variable)`

Parameters

expression A string.

variable The name of a variable.

Returns

Nothing.

Description

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

Example

The following example returns the value of the string "Hello".

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

The result is 5.

See also

" " (string delimiter), `String.length`

_level

Availability

Flash Player 4.

Usage

`_levelN`

Description

Property; a reference to the root movie Timeline of `_levelN`. You must use the `loadMovieNum` action to load movies into the Flash Player before you use the `_level` property to target them. You can also use `_levelN` to target a loaded movie at the level assigned by *N*.

The initial movie loaded into an instance of the Flash Player is automatically loaded into `_level0`. The movie in `_level0` sets the frame rate, background color, and frame size for all subsequently loaded movies. Movies are then stacked in higher-numbered levels above the movie in `_level0`.

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

Example

The following example stops the playhead in the main Timeline of the movie in `_level9`.

```
_level9.stop();
```

The following example sends the playhead in the main Timeline of the movie in `_level4` to frame 5. The movie in `_level4` must have previously been loaded with a `loadMovieNum` action.

```
_level4.gotoAndStop(5);
```

See also

`loadMovie`, `MovieClip.swapDepths`

loadMovie

Availability

Flash Player 3.

Usage

```
loadMovie("url",level|target[, variables])
```

Parameters

url The absolute or relative URL of the SWF file or JPEG file to be loaded. A relative path must be relative to the SWF file at level 0. The URL must be in the same subdomain as the URL where the movie currently resides. For use in the Flash Player or for testing in test mode in the Flash authoring application, all SWF files must be stored in the same folder, and the filenames cannot include folder or disk drive specifications.

target A path to a target movie clip. The target movie clip will be replaced by the loaded movie or image. You must specify either a *target* movie clip or a *level* of a target movie; you can't specify both.

level An integer specifying the level in the Flash Player into which the movie will be loaded. When you load a movie or image into a level, the `loadMovie` action in the Actions panel in normal mode switches to `loadMovieNum`; in expert mode, you must specify `loadMovieNum` or choose it from the Actions toolbox.

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

Returns

Nothing.

Description

Action; load a SWF or JPEG file into the Flash Player while the original movie is playing. The `loadMovie` action lets you display several movies at once and switch between movies without loading another HTML document. Without the `loadMovie` action, the Flash Player displays a single movie (SWF file) and then closes.

When you use the `loadMovie` action, you must specify either a level in the Flash Player or a target movie clip, into which the movie will load. If you specify a level, the action changes to `loadMovieNum`. If a movie is loaded into a target movie clip, you can use the target path of that movie clip to target the loaded movie.

A movie or image loaded into a target inherits the position, rotation, and scale properties of the targeted movie clip. The upper left corner of the loaded image or movie aligns with the registration point of the targeted movie clip. Alternatively, if the target is the `_root` Timeline, the upper left corner of the image or movie aligns with the upper left corner of the Stage.

Use the `unloadMovie` action to remove movies loaded with the `loadMovie` action.

Example

The following `loadMovie` statement is attached to a navigation button labeled Products. There is an invisible movie clip on the Stage with the instance name `dropZone`. The `loadMovie` action uses this movie clip as the target parameter to load the products in the SWF file, into the correct position on the Stage.

```
on(release) {  
    loadMovie("products.swf",_root.dropZone);  
}
```

The following example loads a JPEG image from the same directory as the SWF File that calls the `loadMovie` action:

```
loadMovie("image45.jpeg", "ourMovieClip");
```

See also

`loadMovieNum`, `unloadMovie`, `unloadMovieNum`, `_level`

loadMovieNum

Availability

Flash Player 4. Flash 4 files opened in Flash 5 will be converted to use the correct syntax.

Usage

```
loadMovieNum("url",level[, variables])
```

Parameters

url The absolute or relative URL of the SWF or JPEG file to be loaded. A relative path must be relative to the SWF file at level 0. The URL must be in the same subdomain as the URL where the movie currently resides. For use in the stand-alone Flash Player or for testing in test-movie mode in the Flash authoring application, all SWF files must be stored in the same folder; and the filenames cannot include folder or disk drive specifications.

level An integer specifying the level in the Flash Player into which the movie will be loaded.

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

Returns

Nothing.

Description

Action; loads a SWF or JPEG file into a level in the Flash Player while the originally loaded movie is playing. When you load a movie into a level instead of a target, the `loadMovie` action in the Actions panel in normal mode switches to `loadMovieNum`; in expert mode, you must specify `loadMovieNum` or choose it from the Actions toolbox. Normally, the Flash Player displays a single movie (SWF file) and then closes. The `loadMovieNum` action lets you display several movies at once and switch between movies without loading another HTML document.

The Flash Player has a stacking order of levels starting with level 0. These levels are like layers of acetate; they are transparent except for the objects on each level. When you use the `loadMovieNum` action, you must specify a level in the Flash Player into which the movie will load. Once a movie is loaded into a level, you can use the syntax, `_levelN`, where *N* is the level number, to target the movie.

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

The `loadMovieNum` action also allows you to load JPEG files into a movie while it plays. For both images and SWF files, the upper left corner of the image aligns with the upper left corner of the Stage when the file loads. Also in both cases, the loaded file inherits rotation and scaling, and the original content is over written.

Use the `unloadMovieNum` action to remove movies or images loaded with the `loadMovieNum` action.

Example

This example loads the JPEG image “image45.jpg” into level 2 of the Flash Player.

```
loadMovieNum("http://www.blog.com/image45.jpg", 2); //
```

See also

`loadMovie`, `unloadMovie`, `unloadMovieNum`, `_level`

loadVariables

Availability

Flash Player 4.

Usage

```
loadVariables ("url" ,level/"target" [, variables])
```

Parameters

url An absolute or relative URL where the variables are located. If you access the movie using a Web browser, the host for the URL must be in the same subdomain as the movie itself.

level An integer specifying the level in the Flash Player to receive the variables. When you load variables into a level, the action in the Actions panel in normal mode becomes `loadVariablesNum`; in expert mode, you must specify `loadVariablesNum` or choose it from the Actions toolbox.

target The target path to a movie clip that receives the loaded variables. You must specify either a *target* movie clip or a *level* (level) in the Flash Player; you can't specify both.

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

Returns

Nothing.

Description

Action; reads data from an external file, such as a text file or text generated by a CGI script, Active Server Pages (ASP), or PHP, or Perl script, and sets the values for variables in a Flash Player level or a target movie clip. This action can also be used to update variables in the active movie with new values.

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

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

The first movie to open in an instance of the Flash Player loads into the bottom level (identified in code as `_level0`). When you use the `loadMovie` or `loadMovieNum` action to load subsequent movies into the Flash Player, you must assign a level number in the Flash Player or a target movie clip into which each movie will load. When you use the `loadVariables` action, you must specify either a Flash Player level or a movie clip target into which the variables will load.

Example

This example loads information from a text file into text fields into the `varTarget` movie clip on the main Timeline. The variable names of the text fields must match the variable names in the `data.txt` file.

```
on(release) {  
    loadVariables("data.txt", "_root.varTarget");  
}
```

See also

`loadVariablesNum`, `loadMovie`, `loadMovieNum`, `getURL`, `MovieClip.loadMovie`, `MovieClip.loadVariables`

loadVariablesNum

Availability

Flash Player 4. Flash 4 files opened in Flash 5 will be converted to use the correct syntax.

Usage

```
loadVariables ("url" ,level [, variables])
```

Parameters

url An absolute or relative URL where the variables are located. If you access the movie using a Web browser, the host for the URL must be in the same subdomain as the movie itself.

level An integer specifying the level in the Flash Player to receive the variables.

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

Returns

Nothing.

Description

Action; reads data from an external file, such as a text file or text generated by a CGI script, Active Server Pages (ASP), or PHP, or Perl script, and sets the values for variables in a Flash Player level. This action can also be used to update variables in the active movie with new values. When you load variables into a level, the action in the Actions panel in normal mode becomes `loadVariablesNum`; in expert mode, you must specify `loadVariablesNum` or choose it from the Actions toolbox.

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

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

The first movie to open in an instance of the Flash Player loads into the bottom level (identified in code as `_level0`). When you use the `loadMovie` or `loadMovieNum` action to load subsequent movies into the Flash Player, you must assign a level number in the Flash Player or a target movie clip into which each movie will load. When you use the `loadVariablesNum` action, you must specify a Flash Player level into which the variables will load.

Example

This example loads information from a text file into text fields in the main Timeline of the movie at level 0 in the Flash Player. The variable names of the text fields must match the variable names in the `data.txt` file.

```
on(release) {  
    loadVariablesNum("data.txt", 0);  
}
```

See also

`getURL`, `loadMovie`, `loadMovieNum`, `loadVariables`, `MovieClip.loadMovie`, `MovieClip.loadVariables`

LoadVars (object)

The LoadVars object is an alternative to the `loadVariables` action for transferring variables between a Flash movie and a server.

You can use the LoadVars object to obtain error information, progress indications, and stream data while it downloads. The LoadVars object works much like the XML object; it uses the methods `load`, `send`, and `sendAndLoad` to communicate with a server. The main difference between the LoadVars object and the XML object is that LoadVars transfers ActionScript name and value pairs, rather than an XML DOM tree stored in the XML object.

The LoadVars object follows the same security restrictions as the XML object.

You must use the constructor `new LoadVars()` to create an instance of the LoadVars object before calling its methods.

The LoadVars object is supported by Flash Player 6 and later.

Method summary for the LoadVars object

Method	Description
<code>LoadVars.load</code>	Downloads variables from a specified URL.
<code>LoadVars.getBytesTotal</code>	Returns the number of bytes loaded from a <code>load</code> or <code>sendAndLoad</code> method.
<code>LoadVars.getBytesTotal</code>	Returns the total number of bytes that will be downloaded by a <code>load</code> or <code>sendAndLoad</code> method.
<code>LoadVars.send</code>	Posts variables from a LoadVars object to a URL.
<code>LoadVars.sendAndLoad</code>	Posts variables from a LoadVars object to a URL and downloads the server's response to a target object.
<code>LoadVars.toString</code>	Returns a URL encoded string that contains all the enumerable variables in the LoadVars object.

Property summary for the LoadVars object

All of the properties for the Key object are constants.

Property	Description
<code>LoadVars.contentType</code>	Indicates the MIME type of the data.
<code>LoadVars.load</code>	A Boolean value that indicates whether a <code>load</code> or <code>sendAndLoad</code> operation has completed.

Event summary for the LoadVars object

Method	Description
<code>LoadVars.onLoad</code>	Invoked when a <code>load</code> or <code>sendAndLoad</code> operation has completed.

Constructor for the LoadVars object

Availability

Flash Player 6.

Usage

```
new LoadVars()
```

Parameters

None.

Returns

Nothing.

Description

Constructor; creates an instance of the LoadVars object. You can then use the methods of that LoadVars object to send and load data.

Example

The following example creates an instance of the LoadVars object called `myLoadVars`:

```
myLoadVars = new LoadVars();
```

LoadVars.contentType

Availability

Flash Player 6.

Usage

```
myLoadVars.contentType
```

Description

Property; the MIME type that is sent to the server when you call the `LoadVars.send` or `LoadVars.sendAndLoad` method. The default is `application/x-www-urlform encoded`.

See also

`LoadVars.send`, `LoadVars.sendAndLoad`

LoadVars.getBytesLoaded

Availability

Flash Player 6.

Usage

```
myLoadVars.getBytesLoaded()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the number of bytes downloaded by a `load` or `sendAndLoad` method. The `getBytesLoaded` method returns undefined if no load operation is in progress, or if a load operation has not yet initiated.

LoadVars.getBytesTotal

Availability

Flash Player 6.

Usage

```
myLoadVars.getBytesTotal()
```

Parameters

None.

Returns

An integer.

Description

Method; returns the number of total bytes that are downloaded by a `load` or `sendAndLoad` operation. The `getBytesTotal` method returns undefined if no load operation is in progress or if a load operation has not yet initiated. The `getBytesTotal` method also returns undefined if the number of total bytes can't be determined; for example, if the download was initiated but the server did not transmit an HTTP content-length.

LoadVars.load

Availability

Flash Player 6.

Usage

```
myLoadVars.load(url)
```

Parameters

url The URL from which to download the variables.

Returns

A string.

Description

Method; downloads variables from the specified URL, parses the variable data, and places the resulting variables into `loadVarsObject`. Any properties in `loadVarsObject` with the same names as downloaded variables are overwritten. Any properties in `loadVarsObject` with different names than downloaded variables are not deleted. This is an asynchronous action.

The downloaded data must be in the MIME content type *application/x-www-urlform-encoded*. This is the same format used by `loadVariables`.

This method is similar to the `XML.load` method of the XML object.

LoadVars.loaded

Availability

Flash Player 6.

Usage

```
myLoadVars.loaded
```

Description

Property; undefined by default. When a `load` or `sendAndLoad` operation is started, the `loaded` property is set to `false`. When the `load` or `sendAndLoad` operation completes, the `loaded` property is set to `true`. If the load operation has not yet completed or failed with an error, the `loaded` property remains set to `false`.

The `LoadVars.loaded` property is similar to the `XML.loaded` property of the XML object.

LoadVars.onLoad

Availability

Flash Player 6.

Usage

```
myLoadVars.onLoad(success)
```

Parameters

success The parameter indicates whether the load operation ended in success (`true`) or failure (`false`).

Returns

A Boolean value.

Description

Event handler; invoked when a `load` or `sendAndLoad` operation has ended. If the operation was successful, the *loadVarsObject* is populated with variables downloaded by the `load` or `sendAndLoad` operation, and these variables are available when `onLoad` is invoked.

This method is undefined by default, you can define it by assigning it a callback function.

This method is similar to the `XML.onLoad` method of the XML object.

LoadVars.send

Availability

Flash Player 6.

Usage

```
loadVarsObject.send(url[,target,method])
```

Parameters

loadVarsObject The LoadVars object to upload variables from.

url The URL to upload variables to.

target The browser frame window in which any response will be displayed.

method The "GET" or "POST" method of the HTTP protocol.

Returns

A string.

Description

Method; sends the variables in the *myLoadVars* object to the specified URL. All enumerable variables in the *myLoadVars* object are concatenated into a string in the *application/x-www-urlform-encoded* format by default, and the string is posted to the URL using the HTTP POST method. This is the same format used by the `loadVariables` action. The MIME content type sent in the HTTP request headers is the value of `myLoadVars.contentType`, or the default *application/x-www-urlform-encoded*. The method "POST" is used unless "GET" is specified.

If the *target* parameter is specified, the server's response is displayed in the browser frame window named *target*. If the *target* parameter is omitted, the server's response is discarded.

This method is similar to the `XML.send` method of the XML object.

LoadVars.sendAndLoad

Availability

Flash Player 6.

Usage

```
myLoadVars.sendAndLoad(url, targetObject[,method])
```

Parameters

loadVarsObject The LoadVars object to upload variables from.

url The URL to upload variables to.

targetObject The LoadVars object that receives the downloaded variables.

method The "GET" or "POST" method of the HTTP protocol.

Returns

A string.

Description

Method; posts variables in the *myLoadVars* object to the specified URL. The server response is downloaded, parsed as variable data, and the resulting variables are placed in the *targetObject* object.

Variables are posted in the same manner as `LoadVars.send`. Variables are downloaded into *targetObject* in the same manner as `LoadVars.load`.

This method is similar to the `XML.sendAndLoad` method of the XML object.

LoadVars.toString

Availability

Flash Player 6.

Usage

loadVarsObject.toString()

Parameters

None.

Returns

A string.

Description

Method; returns a string containing all enumerable variables in the LoadVars object, in the MIME content encoding *application/x-www-urlform-encoded*.

Example

```
var myVars = new LoadVars();
myVars.name = "Gary";
myVars.age = 26;
trace (myVars.toString());
would output
name=Gary&age=26
```

lt (less than – string specific)

Availability

Flash Player 4. This operator has been deprecated in Flash 5 in favor of the new < (less than) operator.

Usage

expression1 lt expression2

Parameters

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

Description

Operator (comparison); compares *expression1* to *expression2* and returns true if *expression1* is less than *expression2*; otherwise, it returns false.

See also

< (less than)

Math (object)

The Math object is a top-level object that you can access without using a constructor.

Use the methods and properties of this object to access and manipulate mathematical constants and functions. All of the properties and methods of the Math object are static, and must be called using the syntax `Math.method(parameter)` or `Math.constant`. In ActionScript, constants are defined with the maximum precision of double-precision IEEE-754 floating-point numbers.

Several of the Math object methods take the radian of an angle as an parameter. You can use the equation below to calculate radian values, or simply pass the equation (entering a value for degrees) for the radian parameter.

To calculate a radian value, use this formula:

```
radian = Math.PI/180 * degree
```

The following is an example of passing the equation as an parameter to calculate the sine of a 45-degree angle:

```
Math.SIN(Math.PI/180 * 45) is the same as Math.SIN(.7854)
```

The Math object is fully supported in Flash Player 5. In Flash Player 4, you can use methods of the Math object, but they are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Method summary for the Math object

Method	Description
<code>Math.abs</code>	Computes an absolute value.
<code>Math.acos</code>	Computes an arc cosine.
<code>Math.asin</code>	Computes an arc sine.
<code>Math.atan</code>	Computes an arc tangent.
<code>Math.atan2</code>	Computes an angle from the x-axis to the point.
<code>Math.ceil</code>	Rounds a number up to the nearest integer.
<code>Math.cos</code>	Computes a cosine.
<code>Math.exp</code>	Computes an exponential value.
<code>Math.floor</code>	Rounds a number down to the nearest integer.
<code>Math.log</code>	Computes a natural logarithm.
<code>Math.max</code>	Returns the larger of the two integers.
<code>Math.min</code>	Returns the smaller of the two integers.
<code>Math.pow</code>	Computes <i>x</i> raised to the power of the <i>y</i> .
<code>Math.random</code>	Returns a pseudo-random number between 0.0 and 1.0.
<code>Math.round</code>	Rounds to the nearest integer.
<code>Math.sin</code>	Computes a sine.
<code>Math.sqrt</code>	Computes a square root.
<code>Math.tan</code>	Computes a tangent.

Property summary for the Math object

All of the properties for the Math object are constants.

Property	Description
<code>Math.E</code>	Euler's constant and the base of natural logarithms (approximately 2.718).
<code>Math.LN2</code>	The natural logarithm of 2 (approximately 0.693).
<code>Math.LOG2E</code>	The base 2 logarithm of e (approximately 1.442).
<code>Math.LN10</code>	The natural logarithm of 10 (approximately 2.302).
<code>Math.LOG10E</code>	The base 10 logarithm of e (approximately 0.434).
<code>Math.PI</code>	The ratio of the circumference of a circle to its diameter (approximately 3.14159).
<code>Math.SQRT1_2</code>	The reciprocal of the square root of 1/2 (approximately 0.707).
<code>Math.SQRT2</code>	The square root of 2 (approximately 1.414).

Math.abs

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

`Math.abs(x)`

Parameters

x A number.

Returns

A number.

Description

Method; computes and returns an absolute value for the number specified by the parameter x .

Math.acos

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

`Math.acos(x)`

Parameters

x A number from -1.0 to 1.0.

Returns

Nothing.

Description

Method; computes and returns the arc cosine of the number specified in the parameter *x*, in radians.

Math.asin

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

```
Math.asin(x);
```

Parameters

x A number from -1.0 to 1.0.

Returns

A number.

Description

Method; computes and returns the arc sine for the number specified in the parameter *x*, in radians.

Math.atan

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

```
Math.atan(x)
```

Parameters

x A number.

Returns

A number.

Description

Method; computes and returns the arc tangent for the number specified in the parameter *x*. The return value is between negative pi divided by 2, and positive pi divided by 2.

Math.atan2

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

```
Math.atan2(y, x)
```

Parameters

x A number specifying the *x* coordinate of the point.

y A number specifying the *y* coordinate of the point.

Returns

A number.

Description

Method; computes and returns the arc tangent of y/x in radians. The return value represents the angle opposite the opposite angle of a right triangle, where *x* is the adjacent side length and *y* is the opposite side length.

Math.ceil

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

```
Math.ceil(x)
```

Parameters

x A number or expression.

Returns

A number.

Description

Method; returns the ceiling of the specified number or expression. The ceiling of a number is the closest integer that is greater than or equal to the number.

Math.cos

Usage

```
Math.cos(x)
```

Parameters

x An angle measured in radians.

Returns

A number.

Description

Method; returns the cosine (a value from -1.0 to 1.0) of the angle specified by the parameter x . The angle x must be specified in radians. Use the information outlined in the introduction to the Math object to calculate a radian.

Math.E

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

`Math.E`

Parameters

None.

Returns

Nothing.

Description

Constant; a mathematical constant for the base of natural logarithms, expressed as e . The approximate value of e is 2.71828.

Math.exp

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

`Math.exp(x)`

Parameters

x The exponent; a number or expression.

Returns

A number.

Description

Method; returns the value of the base of the natural logarithm (e), to the power of the exponent specified in the parameter x . The constant `Math.E` can provide the value of e .

Math.floor

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

```
Math.floor(x)
```

Parameters

x A number or expression.

Returns

A number.

Description

Method; returns the floor of the number or expression specified in the parameter *x*. The floor is the closest integer that is less than or equal to the specified number or expression.

Example

The following code returns a value of 12:

```
Math.floor(12.5);
```

Math.log

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

```
Math.log(x)
```

Parameters

x A number or expression with a value greater than 0.

Returns

A number.

Description

Method; returns the natural logarithm of the parameter *x*.

Math.LOG2E

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash Player 5.

Usage

```
Math.LOG2E
```

Parameters

None.

Returns

Nothing.

Description

Constant; a mathematical constant for the base-2 logarithm of the constant e (`Math.E`), expressed as $\log_2 e$, with an approximate value of 1.442695040888963387.

Math.LOG10E

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

```
Math.LOG10E
```

Parameters

None.

Returns

Nothing.

Description

Constant; a mathematical constant for the base-10 logarithm of the constant e (`Math.E`), expressed as $\log_{10} e$, with an approximate value of 0.43429448190325181667.

Math.LN2

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

```
Math.LN2
```

Parameters

None.

Returns

Nothing.

Description

Constant; a mathematical constant for the natural logarithm of 2, expressed as $\log_e 2$, with an approximate value of 0.69314718055994528623.

Math.LN10

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

`Math.LN10`

Parameters

None.

Returns

Nothing.

Description

Constant; a mathematical constant for the natural logarithm of 10, expressed as $\log_e 10$, with an approximate value of 2.3025850929940459011.

Math.max

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

`Math.max(x , y)`

Parameters

x A number or expression.

y A number or expression.

Returns

A number.

Description

Method; evaluates *x* and *y* and returns the larger value.

Math.min

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

`Math.min(x , y)`

Parameters

x A number or expression.

y A number or expression.

Returns

Nothing.

Description

Method; evaluates *x* and *y* and returns the smaller value.

Math.PI

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

`Math.PI`

Parameters

None.

Returns

Nothing.

Description

Constant; a mathematical constant for the ratio of the circumference of a circle to its diameter, expressed as pi, with a value of 3.14159265358979.

Math.pow

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

`Math.pow(x , y)`

Parameters

x A number to be raised to a power.

y A number specifying a power the parameter *x* is raised to.

Returns

A number.

Description

Method; computes and returns *x* to the power of *y*, x^y .

Math.random

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

```
Math.random()
```

Parameters

None.

Returns

A number.

Description

Method; returns n , where $0 \leq n < 1$.

See also

[random](#)

Math.round

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

```
Math.round(x)
```

Parameters

x A number.

Returns

A number.

Description

Method; rounds the value of the parameter x up or down to the nearest integer and returns the value.

Math.sin

Availability

Flash Player 5. In the Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by the Flash Player 5.

Usage

```
Math.sin(x)
```

Parameters

x An angle measured in radians.

Returns

Nothing.

Description

Method; computes and returns the sine of the specified angle in radians. Use the information outlined in the introduction to the Math object to calculate a radian.

Math.sqrt

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

```
Math.sqrt(x)
```

Parameters

x A number or expression greater than or equal to 0.

Returns

A number.

Description

Method; computes and returns the square root of the specified number.

Math.SQRT1_2

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

```
Math.SQRT1_2
```

Parameters

None.

Returns

Nothing.

Description

Constant; a mathematical constant for the reciprocal of the square root of one half ($1/2$), with an approximate value of 0.707106781186.

Math.SQRT2

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

`Math.SQRT2`

Parameters

None.

Description

Constant; a mathematical constant for the square root of 2, with an approximate value of 1.414213562373.

Math.tan

Availability

Flash Player 5. In Flash Player 4, the methods and properties of the Math object are emulated using approximations and may not be as accurate as the non-emulated math functions supported by Flash Player 5.

Usage

`Math.tan(x)`

Parameters

x An angle measured in radians.

Returns

A number.

Description

Method; computes and returns the tangent of the specified angle. To calculate a radian, use the information outlined in the introduction to the Math (object).

maxscroll

Availability

Flash Player 4.

Usage

variable_name.maxscroll

Description

Property (read-only); a deprecated property that indicates the line number of the topmost visible line of text in a text field when the bottommost line in the field is also visible. The `maxscroll` property works with the `scroll` property to control the display of information in a text field. This property can be retrieved, but not modified.

See also

`TextField.maxscroll`, `TextField.scroll`

mbchr

Availability

Flash Player 4. This function has been deprecated in favor of the `String.fromCharCode` method.

Usage

```
mbchr(number)
```

Parameters

number The number to convert to a multibyte character.

Returns

A string.

Description

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

See also

`String.fromCharCode`

mblength

Availability

Flash Player 4. This function has been deprecated in favor of the `String` (object).

Usage

```
mblength(string)
```

Parameters

string A string.

Returns

A number.

Description

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

mbord

Availability

Flash Player 4. This function has been deprecated in Flash 5 in favor of the `String.charCodeAt` method.

Usage

```
mbord(character)
```

Parameters

character The character to convert to a multibyte number.

Returns

A number.

Description

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

See also

`String.fromCharCode`

mbsubstring

Availability

Flash Player 4. This function is deprecated in Flash 5 in favor of the `String.substr` method.

Usage

```
mbsubstring(value, index, count)
```

Parameters

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

index The number of the first character to extract.

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

Returns

A string.

Description

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

See also

`String.substr`

method

Availability

Flash Player 6.

Usage

```
object.method = function ([parameters]) {  
    ...body of function...  
};
```

Parameters

object An identifier for an object.

method An identifier for a method.

parameters Parameters to pass to the function. An optional parameter.

Returns

Nothing.

Description

Action (normal mode only); allows you to define methods for objects using the Actions panel in normal mode. For more information about defining methods for objects, see the *Using Flash*.

Mouse (object)

The Mouse object is a top-level object that you can access without using a constructor. You can use the methods of the Mouse object to hide and show the cursor in the movie. The mouse pointer is visible by default, but you can hide it and implement a custom pointer that you create using a movie clip.

Mouse method summary

Method	Description
<code>Mouse.addListener</code>	Registers an object to receive <code>onMouseDown</code> , <code>onMouseMove</code> , and <code>onMouseUp</code> notification.
<code>Mouse.hide</code>	Hides the mouse pointer in the movie.
<code>Mouse.removeListener</code>	Removes an object that was registered with the <code>addListener</code> method.
<code>Mouse.show</code>	Displays the mouse pointer in the movie.

Mouse listener summary

Method	Description
<code>MovieClip.onMouseDown</code>	Notified when the mouse button is pressed down.
<code>MovieClip.onMouseMove</code>	Notified when the mouse button is moved.
<code>MovieClip.onMouseUp</code>	Notified when the mouse button is released.

Mouse.addListener

Availability

Flash Player 6.

Usage

`Mouse.addListener (newListener)`

Parameters

newListener An object.

Returns

Nothing.

Description

Method; registers an object to receive notifications of the `onMouseDown`, `onMouseMove` and `onMouseUp` callback handlers.

The *newListener* parameter should contain an object with defined methods for the `onMouseDown`, `onMouseMove`, and `onMouseUp` events.

When the mouse is pressed, moved, or released, regardless of the input focus, all listening objects that are registered with the `addListener` method have either their `onMouseDown` method, `onMouseMove` method or `onMouseUp` method invoked. Multiple objects may listen for keyboard notifications. If the listener *newListener* is already registered, no change occurs.

Mouse.hide

Availability

Flash Player 5.

Usage

`Mouse.hide()`

Parameters

None.

Returns

Nothing.

Description

Method; hides the cursor in a movie. The cursor is visible by default.

Example

The following code, attached to a movie clip on the main Timeline, hides the standard cursor, and sets the *x* and *y* positions of the `customCursor` movie clip instance to the *x* and *y* mouse positions in the main Timeline.

```
onClipEvent(enterFrame){
    Mouse.hide();
    customCursorMC._x = _root._xmouse;
    customCursorMC._y = _root._ymouse;
}
```

See also

`Mouse.show`, `MovieClip._xmouse`, `MovieClip._ymouse`

Mouse.onMouseDown

Availability

Flash Player 6.

Usage

someListener.onMouseDown

Description

Listener; notified when the mouse is pressed. To use the `onMouseDown` listener, you must create a listener object. You can then define a function for `onMouseDown` and use the `addListener` method to register the listener with the `Mouse` object, as in the following code:

```
someListener = new Object();
someListener.onMouseDown = function () { ... };
Mouse.addListener(someListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

See also

`Mouse.addListener`

Mouse.onMouseMove

Availability

Flash Player 6.

Usage

someListener.onMouseMove

Description

Listener; notified when the mouse moves. To use the `onMouseMove` listener, you must create a listener object. You can then define a function for `onMouseMove` and use the `addListener` method to register the listener with the `Mouse` object, as in the following code:

```
someListener = new Object();
someListener.onMouseMove = function () { ... };
Mouse.addListener(someListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

See also

`Mouse.addListener`

Mouse.onMouseUp

Availability

Flash Player 6.

Usage

```
someListener.onMouseUp
```

Description

Listener; notified when the mouse is released. To use the `onMouseUp` listener, you must create a listener object. You can then define a function for `onMouseUp` and use the `addListener` method to register the listener with the `Mouse` object, as in the following code:

```
someListener = new Object();
someListener.onMouseUp = function () { ... };
Mouse.addListener(someListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

See also

`Mouse.addListener`

Mouse.removeListener

Availability

Flash Player 6.

Usage

```
Mouse.removeListener (listener)
```

Parameters

listener An object.

Returns

If the *listener* object was successfully removed, the method returns `true`; if the *listener* was not successfully removed (for example, if the *listener* was not on the `Mouse` object's listener list), the method returns `false`.

Description

Method; removes an object that was previously registered with the `addListener` method.

Mouse.show

Availability

Flash Player 5.

Usage

`Mouse.show()`

Parameters

None.

Returns

Nothing.

Description

Method; displays the cursor in a movie. The cursor is visible by default.

See also

`Mouse.show`, `MovieClip._xmouse`, `MovieClip._ymouse`

MovieClip (object)

The methods for the `MovieClip` object provide the same functionality as actions that target movie clips. There are also additional methods that do not have equivalent actions in the Actions toolbox in the Actions panel.

You do not need to use a constructor method to call the methods of the `MovieClip` object; instead, you reference movie clip instances by name, using the following syntax:

```
myMovieClip.play();  
myMovieClip.gotoAndPlay(3);
```

Method summary for the MovieClip object

Method	Description
<code>MovieClip.attachMovie</code>	Attaches a movie in the library.
<code>MovieClip.createEmptyMovieClip</code>	Creates an empty movie clip.
<code>MovieClip.createTextField</code>	Creates an empty text field.
<code>MovieClip.duplicateMovieClip</code>	Duplicates the specified movie clip.
<code>MovieClip.getBounds</code>	Returns the minimum and maximum x and y coordinates of a movie in a specified coordinate space.
<code>MovieClip.getBytesLoaded</code>	Returns the number of bytes loaded for the specified movie clip.
<code>MovieClip.getBytesTotal</code>	Returns the size of the movie clip in bytes.
<code>MovieClip.getDepth</code>	Returns the depth of a movie clip.
<code>MovieClip.getURL</code>	Retrieves a document from a URL.
<code>MovieClip.globalToLocal</code>	Converts the point object from Stage coordinates to the local coordinates of the specified movie clip.
<code>MovieClip.gotoAndPlay</code>	Sends the playhead to a specific frame in the movie clip and plays the movie.

Method	Description
<code>MovieClip.gotoAndStop</code>	Sends the playhead to a specific frame in the movie clip and stops the movie.
<code>MovieClip.hitTest</code>	Returns <code>true</code> if bounding box of the specified movie clip intersects the bounding box of the target movie clip.
<code>MovieClip.loadMovie</code>	Loads the specified movie into the movie clip.
<code>MovieClip.loadVariables</code>	Loads variables from a URL or other location into the movie clip.
<code>MovieClip.localToGlobal</code>	Converts a <code>Point</code> object from the local coordinates of the movie clip to the global Stage coordinates.
<code>MovieClip.nextFrame</code>	Sends the playhead to the next frame of the movie clip.
<code>MovieClip.play</code>	Plays the specified movie clip.
<code>MovieClip.prevFrame</code>	Sends the playhead to the previous frame of the movie clip.
<code>MovieClip.removeMovieClip</code>	Removes the movie clip from the Timeline if it was created with a <code>duplicateMovieClip</code> action or method or the <code>attachMovie</code> method.
<code>MovieClip.setMask</code>	Specifies a movie clip as a mask for another movie clip.
<code>MovieClip.startDrag</code>	Specifies a movie clip as draggable and begins dragging the movie clip.
<code>MovieClip.stop</code>	Stops the currently playing movie.
<code>MovieClip.stopDrag</code>	Stops the dragging of any movie clip that is being dragged.
<code>MovieClip.swapDepths</code>	Swaps the depth level of two movies.
<code>MovieClip.unloadMovie</code>	Removes a movie that was loaded with the <code>loadMovie</code> action.

Drawing method summary for the MovieClip

Method	Description
<code>MovieClip.beginFill</code>	Begins drawing a fill on the Stage.
<code>MovieClip.beginGradientFill</code>	Begins drawing a gradient fill on the Stage.
<code>MovieClip.clear</code>	Removes all the drawing commands associated with a movie clip instance.
<code>MovieClip.curveTo</code>	Draws a curve using the latest line style.
<code>MovieClip.endFill</code>	Ends the fill specified by <code>beginFill</code> or <code>beginGradientFill</code> .
<code>MovieClip.lineStyle</code>	Defines the stroke of lines created with the <code>lineTo</code> and <code>curveTo</code> methods.
<code>MovieClip.lineTo</code>	Draws a line using the current line style.
<code>MovieClip.moveTo</code>	Moves the current drawing position to specified coordinates.

Property summary for the MovieClip object

Property	Description
MovieClip._alpha	The transparency value of a movie clip instance.
MovieClip._currentframe	The frame number in which the playhead is currently located.
MovieClip._droptarget	The absolute path in slash syntax notation of the movie clip instance on which a draggable movie clip was dropped.
MovieClip.enabled	Indicates whether a button movie clip is enabled.
MovieClip.focusEnabled	Enables a movie clip to receive focus.
MovieClip._focusrect	Indicates whether a focused movie clip has a yellow rectangle around it.
MovieClip._framesloaded	The number of frames that have been loaded from a streaming movie.
MovieClip._height	The height of a movie clip instance, in pixels.
MovieClip.hitArea	Designates another movie clip to serve as the hit area for a button movie clip.
MovieClip._highquality	Sets the rendering quality of a movie.
MovieClip._name	The instance name of a movie clip instance.
MovieClip._parent	A reference to the movie clip that encloses the movie clip.
MovieClip._rotation	The degree of rotation of a movie clip instance.
MovieClip._soundbuftime	The number of seconds before a sound starts to stream.
MovieClip.tabChildren	Indicates whether the children of a movie clip are included in automatic tab ordering.
MovieClip.tabEnabled	Indicates whether a movie clip is included in tab ordering.
MovieClip.tabIndex	Indicates the tab order of an object.
MovieClip._target	The target path of a movie clip instance.
MovieClip._totalframes	The total number of frames in a movie clip instance.
MovieClip.trackAsMenu	Indicates whether other buttons can receive mouse release events.
MovieClip._url	The URL of the SWF file from which a movie clip was downloaded.
MovieClip.useHandCursor	Determines whether the hand is displayed when a user rolls over a button movie clip.
MovieClip._visible	A Boolean value that determines whether a movie clip instance is hidden or visible.
MovieClip._width	The width of a movie clip instance, in pixels.
MovieClip._x	The x coordinate of a movie clip instance
MovieClip._xmouse	The x coordinate of the cursor within a movie clip instance.
MovieClip._xscale	The value specifying the percentage for horizontally scaling a movie clip.
MovieClip._y	The y coordinate of a movie clip instance.
MovieClip._ymouse	The y coordinate of the cursor within a movie clip instance.
MovieClip._yscale	The value specifying the percentage for vertically scaling a movie clip.

Event handler summary for the MovieClip object

Property	Description
<code>MovieClip.onData</code>	Invoked when all the data is loaded into a movie clip.
<code>MovieClip.onDragOut</code>	Invoked while the pointer is outside the button; the mouse button is pressed inside, and then rolls outside the button area.
<code>MovieClip.onDragOver</code>	Invoked while the pointer is over the button; the mouse button has been pressed then rolled outside the button, and then rolled back over the button.
<code>MovieClip.onEnterFrame</code>	Invoked continually at the frame rate of the movie. The actions associated with the <code>enterFrame</code> clip event are processed before any frame actions that are attached to the affected frames.
<code>MovieClip.onKeyDown</code>	Invoked when a key is pressed. Use the <code>Key.getCode</code> and <code>Key.getAscii</code> methods to retrieve information about the last key pressed.
<code>MovieClip.onKeyUp</code>	Invoked when a key is released.
<code>MovieClip.onKillFocus</code>	Invoked when focus is removed from a button.
<code>MovieClip.onLoad</code>	Invoked when the movie clip is instantiated and appears in the Timeline.
<code>MovieClip.onMouseDown</code>	Invoked when the left mouse button is pressed.
<code>MovieClip.onMouseMove</code>	Invoked every time the mouse is moved.
<code>MovieClip.onMouseUp</code>	Invoked when the left mouse button is released.
<code>MovieClip.onPress</code>	Invoked when the mouse is pressed while the pointer is over a button.
<code>MovieClip.onRelease</code>	Invoked when the mouse is released while the pointer is over a button.
<code>MovieClip.onReleaseOutside</code>	Invoked when the mouse is released while the pointer is outside the button after the button is pressed while the pointer is inside the button.
<code>MovieClip.onRollOut</code>	Invoked when the pointer rolls outside of a button area.
<code>MovieClip.onRollOver</code>	Invoked when the mouse pointer rolls over a button.
<code>MovieClip.onSetFocus</code>	Invoked when a button has input focus and a key is released.
<code>MovieClip.onUnload</code>	Invokes in the first frame after the movie clip is removed from the Timeline. The actions associated with the <code>Unload</code> movie clip event are processed before any actions are attached to the affected frame.

MovieClip._alpha

Availability

Flash Player 4.

Usage

myMovieClip._alpha

Description

Property; sets or retrieves the alpha transparency (*value*) of the movie clip specified by *MovieClip*. Valid values are 0 (fully transparent) to 100 (fully opaque). 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 with the `_alpha` property set to 0.

Example

The following statements set the `_alpha` property of a movie clip named `star` to 30% when the button is clicked:

```
on(release) {  
    star._alpha = 30;  
}
```

MovieClip.attachMovie

Availability

Flash Player 5.

Usage

```
myMovieClip.attachMovie( idName, newName, depth [, initObject] )
```

Parameters

idName The linkage name of the movie clip symbol in the library to attach to a movie clip on the Stage. This is the name entered in the Identifier field in the Symbol 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 movie is placed.

initObject An object containing properties with which to populate the newly attached movie clip. This parameter allows dynamically created movie clips to receive clip parameters. If *initObject* is not an object, it is ignored. All properties of *initObject* are copied into the new instance. The properties specified with *initObject* are available to the constructor function. This parameter is optional.

Returns

Nothing.

Description

Method; takes a symbol from the library and attaches it to the movie on the Stage specified by *MovieClip*. Use the `removeMovieClip` or `unloadMovie` action or method to remove a movie attached with `attachMovie`.

Example

The following example attaches the symbol with the Linkage Identifier “circle” to the movie clip instance, which is on the Stage in the movie.

```
on (release) {  
    thing.attachMovie( "circle", "circle1", 2 );  
}
```

See also

`MovieClip.removeMovieClip`, `MovieClip.unloadMovie`, `Object.registerClass`, `removeMovieClip`

MovieClip.beginFill

Availability

Flash Player 6.

Usage

```
myMovieClip.beginFill ([rgb[], alpha]])
```

Parameter

rgb A hex color value (for example, red is 0xFF0000, blue is 0x0000FF, and so on). If this value is not provided or is undefined, a fill is not created.

alpha An integer between 0–100 that specifies the alpha value of the fill. If this value is not provided, 100 (solid) is used. If the value is less than 0, Flash uses 0. If the value is greater than 100, Flash uses 100.

Returns

Nothing.

Description

Method; indicates the beginning of a new drawing path. If an open path exists (that is, if the current drawing position does not equal the previous position specified in a `moveTo` method) and it has a fill associated with it, that path is closed with a line and then filled. This is similar to what happens when the `endFill` method is called.

See also

`MovieClip.beginGradientFill`, `MovieClip.endFill`

MovieClip.beginGradientFill

Availability

Flash Player 6.

Usage

```
myMovieClip.beginGradientFill (fillType, colors, alphas, ratios, matrix)
```

Parameter

fillType Either the string "linear" or the string "radial".

colors An array of RGB hex color values to be used in the gradient (for example, red is 0xFF0000, blue is 0x0000FF, and so on).

alphas An array of alpha values for the corresponding colors in the *colors* array; valid values are 0–100. If the value is less than 0, Flash uses 0. If the value is greater than 100, Flash uses 100.

ratios An array of color distribution ratios; valid values are 0–255. This value defines the percentage of the width where the color is sampled at 100 percent.

matrix A transformation matrix that is an object with either of the following two sets of properties.

- *a, b, c, d, e, f, g, h, i*, which can be used to describe a 3 x 3 matrix of the following form:

```
a b c
d e f
g h i
```

The following example uses a `beginGradientFill` method with a *matrix* parameter that is an object with these properties.

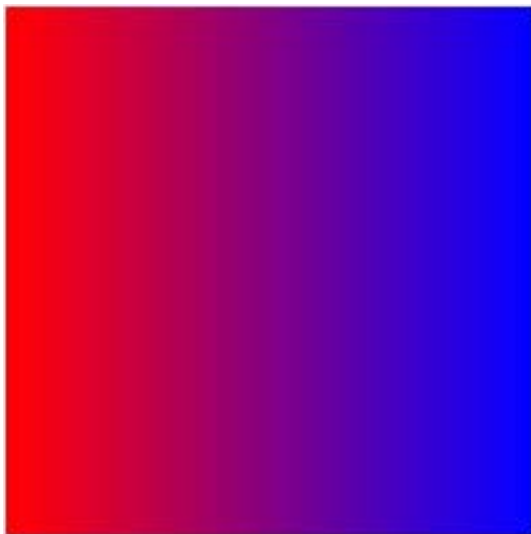
```
_root.createEmptyMovieClip( "grad", 1 );
    with ( _root.grad )

    {

        colors = [ 0xFF0000, 0x0000FF ];
        alphas = [ 100, 100 ];
        ratios = [ 0, 0xFF ];
        matrix = { a:200, b:0, c:0, d:0, e:200, f:0, g:200, h:200, i:1 };
        beginGradientFill( "linear", colors, alphas, ratios, matrix );
        moveto(100,100);
        lineto(100,300);
        lineto(300,300);
        lineto(300,100);
        lineto(100,100);
        endFill();

    }
```

If a *matrixType* property does not exist then the remaining parameters are all required; the function fails if any of them are missing. This matrix scales, translates, rotates, and skews the unit gradient which is defined at (-1,-1) and (1,1).<



- *matrixType*, *x*, *y*, *w*, *h*, *r*.

The properties indicate the following: *matrixType* is the string "box", *x* is the horizontal position relative to the registration point of the parent clip for the upper left corner of the gradient, *y* is the vertical position relative to the registration point of the parent clip for the upper left corner of the gradient, *w* is the width of the gradient, *h* is the height of the gradient, and *r* is the rotation in radians of the gradient.

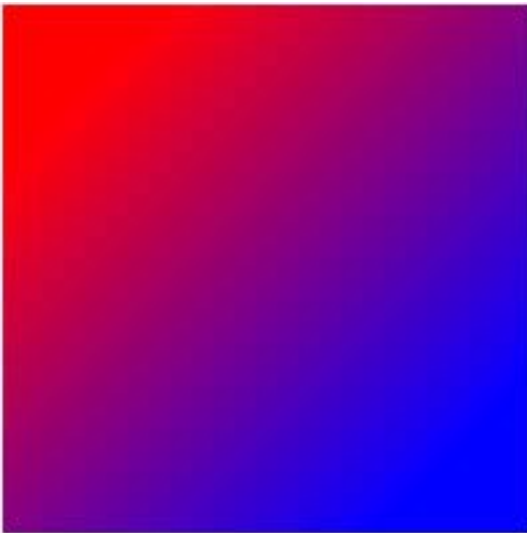
The following example uses a `beginGradientFill` method with a *matrix* parameter that is an object with these properties.

```
_root.createEmptyMovieClip( "grad", 1 );
    with ( _root.grad )

    {

        colors = [ 0xFF0000, 0x0000FF ];
        alphas = [ 100, 100 ];
        ratios = [ 0, 0xFF ];
        matrix = { matrixType:"box", x:100, y:100, w:200, h:200, r:(45/
180)*Math.PI };
        beginGradientFill( "linear", colors, alphas, ratios, matrix );
        moveto(100,100);
        lineto(100,300);
        lineto(300,300);
        lineto(300,100);
        lineto(100,100);
        endFill();
    }
}
```

If a *matrixType* property exists then it must equal "box" and the remaining parameters are all required. The function fails if any of these conditions are not met.



Returns

Nothing.

Description

Method; indicates the beginning of a new drawing path. If the first parameter is `undefined`, or if no parameters are passed, the path has no fill. If an open path exists (that is if the current drawing position does not equal the previous position specified in a `moveTo` method), and it has a fill associated with it, that path is closed with a line and then filled. This is similar to what happens when you call the `endFill` method.

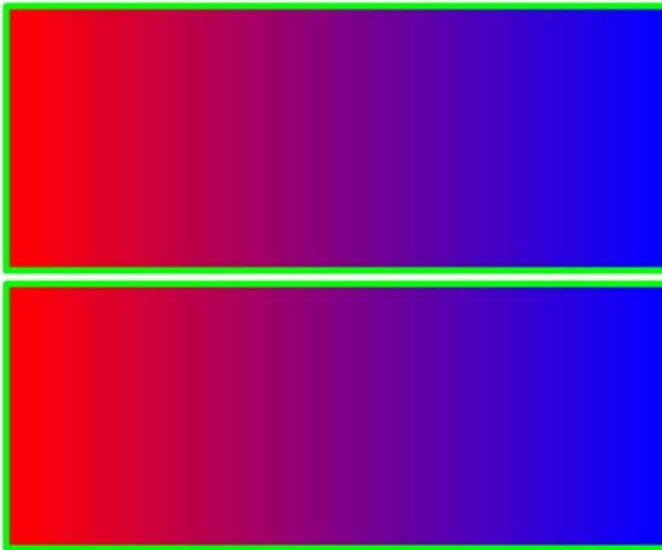
This method fails if any of the following conditions exist:

- The number of items in the *colors*, *alphas*, and *ratios* parameters are not equal.
- The *fillType* parameter is not “linear” or “radial”.
- Any of the fields in the object for the *matrix* parameter are missing or invalid.

Example

The following code uses both methods to draw two stacked rectangles with a red-blue gradient fill and a 5-pt. solid green stroke.

```
_root.createEmptyMovieClip("goober",1);
with ( _root.goober )
{
    colors = [ 0xFF0000, 0x0000FF ];
    alphas = [ 100, 100 ];
    ratios = [ 0, 0xFF ];
    lineStyle( 5, 0x00ff00 );
    matrix = { a:500,b:0,c:0,d:0,e:200,f:0,g:350,h:200,i:1};
    beginGradientFill( "linear", colors, alphas, ratios, matrix );
    moveto(100,100);
    lineto(100,300);
    lineto(600,300);
    lineto(600,100);
    lineto(100,100);
    endFill();
    matrix = { matrixType:"box", x:100, y:310, w:500, h:200, r:(0/180)*Math.PI
    };
    beginGradientFill( "linear", colors, alphas, ratios, matrix );
    moveto(100,310);
    lineto(100,510);
    lineto(600,510);
    lineto(600,310);
    lineto(100,310);
    endFill();
}
```



See also

[MovieClip.beginFill](#), [MovieClip.endFill](#), [MovieClip.lineStyle](#), [MovieClip.lineTo](#), [MovieClip.moveTo](#)

MovieClip.clear

Availability

Flash Player 6.

Usage

```
myMovieClip.clear()
```

Parameters

None.

Returns

Nothing.

Description

Method; removes all the drawing commands associated with a movie clip. Shapes and lines that are drawn with the Flash drawing tools are unaffected. Calling the `clear` method also removes the current line style.

See also

`MovieClip.lineStyle`

MovieClip.createEmptyMovieClip

Availability

Flash Player 6.

Usage

```
myMovieClip.createEmptyMovieClip (instanceName, depth)
```

Parameter

instanceName A string that identifies the instance name of the new movie clip.

depth An integer that specifies the depth of the new movie clip.

Returns

Nothing.

Description

Method; creates an empty movie clip as a child of an existing movie clip. This method behaves similarly to the `attachMovie` method, but you don't need to provide an external linkage name for the new movie clip. The registration point for a newly created empty movie clip is the upper left corner. This method fails if any of the parameters are missing.

See also

`MovieClip.attachMovie`

MovieClip.createTextField

Availability

Flash Player 6.

Usage

```
myMovieClip.createTextField (instanceName, depth, x, y, width, height)
```

Parameters

instanceName A string that identifies the instance name of the new text field.

depth A positive integer that specifies the depth of the new text field.

x An integer that specifies the x coordinate of the new text field.

y An integer that specifies the y coordinate of the new text field.

width A positive integer that specifies the width of the new text field.

height A positive integer that specifies the height of the new text field.

Returns

Nothing.

Description

Method; creates a new, empty text field as a child of the movie clip specified by the *MovieClip* parameter. You can use the `createTextField` method to create text fields while a movie plays. The text field is positioned at (*x*, *y*) with dimensions *width* by *height*. The *x* and *y* parameters are relative to the container movie clip; these parameters correspond to the `_x` and `_y` properties of the text field. The *width* and *height* parameters correspond to the `_width` and `_height` properties of the text field.

The default properties of a text field are as follows:

```
type = "dynamic",  
border = false,  
background = false,  
password = false,  
multiline = false,  
html = false,  
embedFonts = false,  
variable = null,  
maxChars = null
```

A text field created with `createTextField` receives the following default `TextFormat` object:

```
font = "Times New Roman"
size = 12
textColor = 0x000000
bold = false
italic = false,
underline = false
url = ""
target = ""
align = "left"
leftMargin = 0
rightMargin = 0
indent = 0
leading = 0
bullet = false
tabStops = [] (empty array)
```

Example

The following example creates a text field with a width of 300, a height of 100, an *x* coordinate of 100, a *y* coordinate of 100, no border, red, underlined text.

```
_root.createTextField("mytext",1,100,100,300,100);
mytext.multiline = true;
mytext.wordWrap = true;
mytext.border = false;

myformat = new TextFormat();
myformat.color = 0xff0000;
myformat.bullet = false;
myformat.underline = true;

mytext.text = "this is my first test field object text";
mytext.setTextFormat(myformat);
```

See also

`TextFormat` (object)

MovieClip._currentframe

Availability

Flash Player 4.

Usage

myMovieClip.`_currentframe`

Description

Property (read-only); returns the number of the frame in which the playhead is located in the Timeline specified by *MovieClip*.

Example

The following example uses the `_currentframe` property to direct the playhead of the movie clip `actionClip` to advance five frames ahead of its current location.

```
actionClip.gotoAndStop(_currentframe + 5);
```

MovieClip.curveTo

Availability

Flash Player 6.

Usage

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

Parameters

controlX An integer that specifies a horizontal position relative to the registration point of the parent movie clip of the control point.

controlY An integer that specifies a vertical position relative to the registration point of the parent movie clip of the control point.

anchorX An integer that specifies a horizontal position relative to the registration point of the parent movie clip of the next anchor point.

anchorY An integer that specifies a vertical position relative to the registration point of the parent movie clip of the next anchor point.

Returns

Nothing.

Description

Methods; draws a curve using the current line style from the current drawing position to (*anchorX*, *anchorY*) using the control point specified by (*controlX*, *controlY*). The current drawing position is then set to (*anchorX*, *anchorY*). If the movie clip you are drawing in contains content created with the Flash drawing tools, calls to `curveTo` are drawn underneath this content. If you call `curveTo` before any calls to `moveTo`, the current drawing position defaults to (0, 0). If any of the parameters are missing, this method fails and the current drawing position is not changed.

Example

The following example draws a circle with a hairline-point, solid blue line and a solid red fill.

```
_root.createEmptyMovieClip( "circle", 1 );
with ( _root.circle )
{
    lineStyle( 0, 0x0000FF, 100 );
    beginFill( 0xFF0000 );
    moveTo( 500, 500 );
    curveTo( 600, 500, 600, 400 );
    curveTo( 600, 300, 500, 300 );
    curveTo( 400, 300, 400, 400 );
    curveTo( 400, 500, 500, 500 );
    endFill();
}
```

See also

MovieClip.beginFill, MovieClip.createEmptyMovieClip, MovieClip.endFill,
MovieClip.lineStyle, MovieClip.lineTo, MovieClip.moveTo

MovieClip._droptarget

Availability

Flash Player 4.

Usage

myMovieClip._droptarget

Description

Property (read-only); returns the absolute path in slash syntax notation of the movie clip instance on which the *MovieClip* was dropped. The `_droptarget` property always returns a path that starts with a slash (/). To compare the `_droptarget` property of an instance to a reference, use the `eval` function to convert the returned value from slash syntax to a dot syntax reference.

Example

The following example evaluates the `_droptarget` property of the `garbage` movie clip instance and uses `eval` to convert it from slash syntax to a dot syntax reference. The `garbage` reference is then compared to the reference to the `trash` movie clip instance. If the two references are equivalent, the visibility of `garbage` is set to `false`. If they are not equivalent, the `garbage` instance is reset to its original position.

```
if (eval(garbage._droptarget) == _root.trash) {  
    garbage._visible = false;  
} else {  
    garbage._x = x_pos;  
    garbage._y = y_pos;  
}
```

The variables `x_pos` and `y_pos` are set on Frame 1 of the movie with the following script:

```
x_pos = garbage._x;  
y_pos = garbage._y;
```

See also

`startDrag`

MovieClip.cloneMovieClip

Availability

Flash Player 5.

Usage

myMovieClip.cloneMovieClip(newname, depth [,initObject])

Parameters

newname A unique identifier for the duplicate movie clip.

depth A unique number specifying the depth at which the movie specified is to be placed.

initObject An object containing properties with which to populate the duplicated movie clip. This parameter allows dynamically created movie clips to receive clip parameters. If *initObject* is not an object, it is ignored. All properties of *initObject* are copied into the new instance. The properties specified with *initObject* are available to the constructor function. This parameter is optional.

Returns

Nothing.

Description

Method; creates an instance of the specified movie clip while the movie 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 the `duplicateMovieClip` method are not duplicated if you call the `duplicateMovieMethod` on their parent. If the parent movie clip is deleted the duplicate movie clip is also deleted. Movie clips added with `duplicateMovieClip` can be deleted with `removeMovieClip` action or method.

See also

`duplicateMovieClip`, `MovieClip.removeMovieClip`, `removeMovieClip`

MovieClip.enabled

Availability

Flash Player 6.

Usage

myMovieClip.enabled

Description

Property; a Boolean value that indicates whether a button movie clip is enabled. The default value of `enabled` is `true`. If `enabled` is set to `false`, the button movie clip's callback methods and on action events are no longer invoked and the Over, Down and Up frames are disabled. The `enabled` property does not affect the Timeline of the button movie clip; if a movie clip is playing, it continues to play. The movie clip continues to receive movie clip events (for example, `mouseDown`, `mouseUp`, `keyDown` and `keyUp`).

The `enabled` property only governs the button-like properties of a button movie clip. You can change the `enabled` property at any time; the modified button movie clip is immediately enabled or disabled. The `enabled` property can be read out of a prototype object. If `enabled` is set to `false`, the object is not included in automatic tab ordering.

MovieClip.endFill

Availability

Flash Player 6.

Usage

myMovieClip.endFill()

Parameters

None.

Returns

Nothing.

Description

Method; applies a fill to the lines and curves added since the last call to the `beginFill` or `beginGradientFill` method. 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.

MovieClip.focusEnabled

Availability

Flash Player 6.

Usage

myMovieClip.focusEnabled

Description

Property; if value is undefined or false, a movie clip cannot receive input focus unless it is a button movie clip. If the `focusEnabled` property value is true, a movie clip can receive input focus even if it is not a button movie clip.

MovieClip._focusrect

Availability

Flash Player 6.

Usage

myMovieClip._focusrect

Description

Property; a Boolean value that specifies whether a movie clip has a yellow rectangle around it when it has keyboard focus. This property can override the global `_focusrect` property.

MovieClip._framesloaded

Availability

Flash Player 4.

Usage

myMovieClip._framesloaded

Description

Property (read-only); the number of frames that have been loaded from a streaming movie. This property is useful for determining whether the contents of a specific frame, and all the frames before it, have loaded and are available locally in the browser. This property is useful for monitoring the download process of large movies. For example, you might want to display a message to users indicating that the movie is loading until a specified frame in the movie has finished loading.

Example

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

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

MovieClip.getBounds

Availability

Flash Player 5.

Usage

```
myMovieClip.getBounds(targetCoordinateSpace)
```

Parameters

targetCoordinateSpace The target path of the Timeline whose coordinate system you want to use as a reference point.

Returns

An object with the properties `xMin`, `xMax`, `yMin`, and `yMax`.

Description

Method; returns properties that are the minimum and maximum *x* and *y* coordinate values of the instance specified by *MovieClip* for the *targetCoordinateSpace* parameter.

Note: Use the `localToGlobal` and `globalToLocal` methods of the *MovieClip* object to convert the movie clip's local coordinates to Stage coordinates, or Stage coordinates to local coordinates respectively.

Example

In the following example, the object that the `getBounds` method returns is assigned to the identifier `clipBounds`. You can then access the values of each property and use them in a script. In this script, another movie clip instance, `clip2`, is placed alongside `clip`.

```
clipBounds = clip.getBounds(_root);  
clip2._x = clipBounds.xMax;
```

See also

`MovieClip.globalToLocal`, `MovieClip.localToGlobal`

MovieClip.getBytesLoaded

Availability

Flash Player 6.

Usage

```
myMovieClip.getBytesLoaded()
```

Parameters

None.

Returns

An integer indicating the number of bytes loaded.

Description

Method; returns the number of bytes loaded (streamed) for the specified *MovieClip* object. You can compare the value of the `getBytesLoaded` method with the value of the `getBytesTotal` method to determine what percentage of a movie clip has loaded.

See also

`MovieClip.getBytesTotal`

MovieClip.getBytesTotal

Availability

Flash Player 5.

Usage

```
myMovieClip.getBytesTotal()
```

Parameters

None.

Returns

An integer indicating the total size, in bytes, of the specified MovieClip object.

Description

Method; returns the size, in bytes, of the specified MovieClip object. For movie clips that are external (the root movie or a movie clip that is being loaded into a target or a level), the return value is the size of the SWF file.

See also

`MovieClip.getBytesLoaded`

MovieClip.getDepth

Availability

Flash Player 6.

Usage

```
myMovieClip.getDepth
```

Parameters

None.

Returns

An integer.

Description

Method; returns the depth of a movie clip instance.

MovieClip.getURL

Availability

Flash Player 5.

Usage

```
myMovieClip.getURL(URL [,window, variables])
```

Parameters

URL The URL from which to obtain the document.

window An optional parameter specifying the name, frame, or expression specifying the window or HTML frame that the document is loaded into. You can also use one of the following reserved target names: *_self* specifies the current frame in the current window, *_blank* specifies a new window, *_parent* specifies the parent of the current frame, *_top* specifies the top-level frame in the current window.

variables An optional parameter specifying a method for sending variables associated with the movie 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 small numbers of variables. *POST* sends the variables in a separate HTTP header and is used for long strings of variables.

Returns

Nothing.

Description

Method; loads a document from the specified URL into the specified window. The *getURL* method can also be used to pass variables to another application defined at the URL using a *GET* or *POST* method.

See also

getURL

MovieClip.globalToLocal

Availability

Flash Player 5.

Usage

```
myMovieClip.globalToLocal(point)
```

Parameters

point The name or identifier of an object created with the generic *Object* object specifying the *x* and *y* coordinates as properties.

Returns

Nothing.

Description

Method; converts the *point* object from Stage (global) coordinates to the movie clip's (local) coordinates.

Example

The following example converts the global *x* and *y* coordinates of the *point* object to the local coordinates of the movie clip.

```
onClipEvent(mouseMove) {  
    point = new object();  
    point.x = _root._xmouse;  
    point.y = _root._ymouse;  
    globalToLocal(point);  
    trace(_root._xmouse + " " + _root._ymouse);  
    trace(point.x + " " + point.y);  
    updateAfterEvent();  
}
```

See also

`MovieClip.getBounds`, `MovieClip.localToGlobal`

MovieClip.gotoAndPlay

Availability

Flash Player 5.

Usage

```
myMovieClip.gotoAndPlay( frame )
```

Parameters

frame The frame number to which the playhead is sent.

Returns

Nothing.

Description

Method; starts playing the movie at the specified frame.

See also

`gotoAndPlay`

MovieClip.gotoAndStop

Availability

Flash Player 5.

Usage

```
myMovieClip.gotoAndStop( frame )
```

Parameters

frame The frame number to which the playhead is sent.

Returns

Nothing.

Description

Method; brings the playhead to the specified frame of this movie clip and stops it there.

See also

`gotoAndStop`

MovieClip._height

Availability

Flash Player 4.

Usage

```
myMovieClip._height
```

Description

Property; sets and retrieves the height of the movie clip, in pixels.

Example

The following code example sets the height and width of a movie clip when the user clicks the mouse.

```
onClipEvent(mouseDown) {  
    _width=200;  
    _height=200;  
}
```

MovieClip._highquality

Availability

Flash Player 6.

Usage

myMovieClip._highquality

Description

Property (global); specifies the level of anti-aliasing applied to the current movie. Specify 2 (BEST) to apply high quality with bitmap smoothing always on. Specify 1 (high quality) to apply anti-aliasing; this will smooth bitmaps if the movie does not contain animation. Specify 0 (low quality) to prevent anti-aliasing. This property can overwrite the global `_highquality` property.

Example

```
myMovieClip._highquality = 1;
```

See also

`_quality`, `toggleHighQuality`

MovieClip.hitArea

Availability

Flash Player 6.

Usage

myMovieClip.hitArea

Returns

A reference to a movie clip.

Description

Property; designates another movie clip to serve as the hit area for a button movie clip. If the `hitArea` property does not exist or is `null` or `undefined`, the button movie clip itself is used as the hit area. The value of the `hitArea` property may be a reference to a movie clip object.

You can change the `hitArea` property at any time; the modified button movie clip immediately takes on the new hit area behavior. The movie clip designated as the hit area does not need to be visible; its graphical shape, although not visible, is hit-tested. The `hitArea` property can be read out of a prototype object.

MovieClip.hitTest

Availability

Flash Player 5.

Usage

```
myMovieClip.hitTest(x, y, shapeFlag)  
myMovieClip.hitTest(target)
```

Parameters

x The *x* coordinate of the hit area on the Stage.

y The *y* coordinate of the hit area on the Stage.

The *x* and *y* coordinates are defined in the global coordinate space.

target The target path of the hit area that may intersect or overlap with the instance specified by *MovieClip*. The *target* 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 only be specified if the hit area is identified using *x* and *y* coordinate parameters.

Returns

Nothing.

Description

Method; evaluates the instance specified by *MovieClip* to see if it overlaps or intersects with the hit area identified by the *target* or *x* and *y* coordinate parameters.

Usage 1: Compares the *x* and *y* coordinates to the shape or bounding box of the specified instance, according to the *shapeFlag* setting. If *shapeFlag* is set to *true*, only the area actually occupied by the instance on the Stage is evaluated, and if *x* and *y* overlap at any point, a value of *true* is returned. This is useful for determining if the movie clip is within a specified hit or hotspot area.

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

Example

The following example uses `hitTest` with the `x_mouse` and `y_mouse` properties to determine whether the mouse is over the target's bounding box:

```
if (hitTest( _root._xmouse, _root._ymouse, false));
```

The following example uses `hitTest` to determine if the movie clip `ball` overlaps or intersects with the movie clip `square`:

```
if(_root.ball, hittest(_root.square)){  
    trace("ball intersects square");  
}
```

See also

`MovieClip.getBounds`, `MovieClip.globalToLocal`, `MovieClip.localToGlobal`

MovieClip.lineStyle

Availability

Flash Player 6.

Usage

```
myMovieClip.lineStyle ([thickness[, rgb[, alpha]])]
```

Parameters

thickness An integer that indicates the thickness of the line in points; valid values are 0 to 255. If a number is not specified, or if the parameter is `undefined`, a line is not drawn. If a value less than 0 is passed, Flash uses 0. The value 0 indicates hairline thickness; the maximum thickness is 255. If a value greater than 255 is passed, the Flash interpreter uses 255.

rgb A hex color value (for example, red is 0xFF0000, blue is 0x0000FF, and so on) of the line. If a value isn't indicated, Flash uses 0x000000 (black).

alpha An integer that indicates the alpha value of the line's color; valid values are 0–100. If a value isn't indicated, Flash uses 100 (solid). If the value is less than 0, Flash uses 0; if the value is greater than 100, Flash uses 100.

Returns

Nothing.

Description

Method; specifies a line style that Flash uses for subsequent calls to the `lineTo` and `curveTo` methods until you call `lineStyle` with different parameters. You can call the `lineStyle` method in the middle of drawing a path to specify different styles for different line segments within a path.

Note: Calls to `clear` reset the `lineStyle` method back to `undefined`.

Example

The following code draws a triangle with a 5-point, solid magenta line and no fill.

```
_root.createEmptyMovieClip( "triangle", 1 );
with ( _root.triangle )
{
    lineStyle( 5, 0xff00ff, 100 );
    moveTo( 200, 200 );
    lineTo( 300,300 );
    lineTo( 100, 300 );
    lineTo( 200, 200 );
}
```

See also

`MovieClip.beginFill`, `MovieClip.beginGradientFill`, `MovieClip.clear`,
`MovieClip.curveTo`, `MovieClip.lineTo`, `MovieClip.moveTo`,

MovieClip.lineTo

Availability

Flash Player 6.

Usage

myMovieClip.lineTo (x, y)

Parameters

x An integer indicating the horizontal position relative to the registration point of the parent movie clip.

y An integer indicating the vertical position relative to the registration point of the parent movie clip.

Returns

Nothing.

Description

Method; draws a line using the current line style from the current drawing position to (x, y); the current drawing position is then set to (x, y). If the movie clip that you are drawing in contains content that was created with the Flash drawing tools, - calls to `lineTo` are drawn underneath the content. If you call the `lineTo` method 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.

Example

The following example draws a triangle with no lines and a partially transparent blue fill.

```
_root.createEmptyMovieClip ("triangle", 1);
with (_root.triangle){
    beginFill (0x0000FF, 50);
    lineStyle (5, 0xFF00FF, 100);
    moveTo (200, 200);
    lineTo (300, 300);
    lineTo (100, 300);
    lineTo (200, 200);
    endFill();
}
```

See also

`MovieClip.beginFill`, `MovieClip.createEmptyMovieClip`, `MovieClip.endFill`,
`MovieClip.lineStyle`, `MovieClip.moveTo`

MovieClip.loadMovie

Availability

Flash Player 5.

Usage

```
myMovieClip.loadMovie("url" [,variables])
```

Parameters

url An absolute or relative URL of the SWF or JPEG file to be loaded. A relative path must be relative to the SWF file at `_level0`. The URL must be in the same subdomain as the URL where the movie currently resides. For use in the stand-alone Flash Player or for testing in test mode in the Flash authoring application, all SWF files must be stored in the same folder, and the filenames cannot include folder or disk drive specifications.

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

Returns

Nothing.

Description

Method; loads SWF or JPEG files into a movie clip in the Flash Player while the original movie is playing. Without the `loadMovie` method, the Flash Player displays a single movie (SWF file) and then closes. The `loadMovie` method lets you display several movies at once and switch between movies without loading another HTML document.

A movie or image loaded into a movie clip inherits the position, rotation, and scale properties of the movie clip. You can use the target path of the movie clip to target the loaded movie.

Use the `unloadMovie` method to remove movies or images loaded with the `loadMovie` method. Use the `loadVariables` method to keep the active movie, and update the variables with new values.

See also

`loadMovie`, `loadMovieNum`, `MovieClip.loadVariables`, `MovieClip.unloadMovie`, `unloadMovie`, `unloadMovieNum`

MovieClip.loadVariables

Availability

Flash Player 5.

Usage

```
myMovieClip.loadVariables("url", variables)
```

Parameters

url The absolute or relative URL for the external file that contains the variables to be loaded. The host for the URL must be in the same subdomain as the movie clip.

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

Returns

Nothing.

Description

Method; reads data from an external file and sets the values for variables in *MovieClip*. The external file can be a text file generated by 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 at the URL be in the standard MIME format: *application/x-www-form-urlencoded* (CGI script format).

See also

`loadMovie`, `loadVariables`, `loadVariablesNum`, `MovieClip.unloadMovie`

MovieClip.localToGlobal

Availability

Flash Player 5.

Usage

myMovieClip.localToGlobal(point)

Parameters

point The name or identifier of an object created with the Object object, specifying the *x* and *y* coordinates as properties.

Returns

Nothing.

Description

Method; converts the *point* object from the movie clip's (local) coordinates, to the Stage (global) coordinates.

Example

The following example converts *x* and *y* coordinates of the *point* object, from the movie clip's coordinates (local) to the Stage coordinates (global). The local *x* and *y* coordinates are specified using the *_xmouse* and *_ymouse* properties to retrieve the *x* and *y* coordinates of the mouse position.

```
onClipEvent(mouseMove) {  
    point = new object();  
    point.x = _xmouse;  
    point.y = _ymouse;  
    _root.out3 = point.x + " == " + point.y;  
    _root.out = _root._xmouse + " == " + _root._ymouse;  
    localToGlobal(point);  
    _root.out2 = point.x + " == " + point.y;  
    updateAfterEvent();  
}
```

See also

MovieClip.globalToLocal

MovieClip.moveTo

Availability

Flash Player 6.

Usage

myMovieClip.moveTo (x, y)

Parameters

x An integer indicating the horizontal position relative to the registration point of the parent movie clip.

y An integer indicating the vertical position relative to the registration point of the parent movie clip.

Returns

Nothing.

Description

Method; moves the current drawing position to (x, y). If any of the parameters are missing, this method fails and the current drawing position is not changed.

Example

This example draws a triangle with a 5-point, solid magenta lines and no fill. The first line creates an empty movie clip to draw with. Inside the `with` statement, a line type is defined and then the starting drawing position is indicated by the `moveTo` method.

```
_root.createEmptyMovieClip( "triangle", 1 );
with ( _root.triangle )
{
    lineStyle( 5, 0xff00ff, 100 );
    moveTo( 200, 200 );
    lineTo( 300,300 );
    lineTo( 100, 300 );
    lineTo( 200, 200 );
}
```

See also

`MovieClip.createEmptyMovieClip`, `MovieClip.lineStyle`, `MovieClip.lineTo`

MovieClip._name

Availability

Flash Player 4.

Usage

myMovieClip._name

Description

Property; returns the instance name of the movie clip specified by *MovieClip*.

MovieClip.nextFrame

Availability

Flash Player 5.

Usage

myMovieClip.nextFrame()

Parameters

None.

Returns

Nothing.

Description

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

See also

`nextFrame`

MovieClip.onData

Availability

Flash Player 6.

Usage

myMovieClip.onData

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when a movie clip receives data from a `loadVariables` or `loadMovie` call.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onData` method that sends a `trace` action to the Output window.

```
myMovieClip.onData = function () {  
    trace ("onData called");  
};
```

MovieClip.onDragOut

Availability

Flash Player 6.

Usage

myMovieClip.onDragOver

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the pointer is pressed and dragged outside and then over the movie clip.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onDragOut` method that sends a `trace` action to the Output window.

```
myMovieClip.onDragOut = function () {  
    trace ("onDragOut called");  
};
```

See also

`MovieClip.onDragOver`

MovieClip.onDragOver

Availability

Flash Player 6.

Usage

myMovieClip.onDragOver

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the pointer is pressed and dragged outside and then over the movie clip.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onDragOut` method that sends a trace action to the Output window.

```
myMovieClip.onDragOver = function () {  
    trace ("onDragOver called");  
};
```

See also

`MovieClip.onDragOut`

MovieClip.onEnterFrame

Availability

Flash Player 6.

Usage

myMovieClip.onEnterFrame

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked continually at the frame rate of the movie. The actions associated with the `enterFrame` clip event are processed before any frame actions that are attached to the affected frames.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onEnterFrame` method that sends a trace to the Output window.

```
myMovieClip.onEnterFrame = function () {  
    trace ("onEnterFrame called");  
};
```

MovieClip.onKeyDown

Availability

Flash Player 6.

Usage

myMovieClip.onKeyDown

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when a movie clip has input focus and a key is pressed. The `onKeyDown` event is invoked with no parameters. You can use the `Key.getAscii` and `Key.getCode` methods to determine which key was pressed.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onKeyDown` method that sends a trace action to the Output window:

```
myMovieClip.onKeyDown = function () {  
    trace ("onKeyDown called");  
};
```

See also

`MovieClip.onKeyUp`

MovieClip.onKeyUp

Availability

Flash Player 6.

Usage

myMovieClip.onKeyUp

Parameters

None.

Returns

Nothing.

Description

Event; invoked when a key is released. The `onKeyUp` event is invoked with no parameters. You can use the `Key.getAscii` and `Key.getCode` methods to determine which key was pressed.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onKeyPress` method that sends a `trace` action to the Output window.

```
myMovieClip.onKeyUp = function () {  
    trace ("onKeyUp called");  
};
```

MovieClip.onKillFocus

Availability

Flash Player 6.

Usage

```
myMovieClip.onKillFocus = function (newFocus) {  
    statements;  
};
```

Parameters

newFocus The object that is receiving the keyboard focus.

Returns

Nothing.

Description

Event handler; an event that is invoked when a movie clip loses keyboard focus. The `onKillFocus` method receives one parameter, *newFocus*, which is an object representing the new object receiving the focus. If no object receives the focus, *newFocus* contains the value `null`.

MovieClip.onLoad

Availability

Flash Player 6.

Usage

```
myMovieClip.onLoad
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the movie clip is instantiated and appears in the Timeline.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onLoad` method that sends a `trace` action to the Output window:

```
myMovieClip.onLoad = function () {  
    trace ("onLoad called");  
};
```

MovieClip.onMouseDown

Availability

Flash Player 6.

Usage

myMovieClip.onMouseDown

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the mouse button is pressed.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onMouseDown` method that sends a `trace` action to the Output window:

```
myMovieClip.onMouseDown = function () {  
    trace ("onMouseDown called");  
}
```

MovieClip.onMouseMove

Availability

Flash Player 6.

Usage

myMovieClip.onMouseMove

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the mouse moves.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onMouseMove` method that sends a `trace` action to the Output window.

```
myMovieClip.onMouseMove = function () {  
    trace ("onMouseMove called");  
};
```

MovieClip.onMouseUp

Availability

Flash Player 6.

Usage

myMovieClip.onMouseUp

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the mouse is released.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onMouseUp` method that sends a `trace` action to the Output window.

```
myMovieClip.onMouseUp = function () {  
    trace ("onMouseUp called");  
};
```

MovieClip.onPress

Availability

Flash Player 6.

Usage

myMovieClip.onPress

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the mouse pointer is clicked on a movie clip.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onPress` method that sends a trace action to the Output window.

```
myMovieClip.onPress = function () {  
    trace ("onPress called");  
};
```

MovieClip.onRelease

Availability

Flash Player 6.

Usage

myMovieClip.onRelease

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when a button movie clip is released.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onPress` method that sends a trace action to the Output window.

```
myMovieClip.onRelease = function () {  
    trace ("onRelease called");  
};
```

MovieClip.onReleaseOutside

Availability

Flash Player 6.

Usage

myMovieClip.onReleaseOutside

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the mouse is released while the pointer is outside the movie clip after the mouse button is pressed inside the movie clip.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onReleaseOutside` method that sends a trace action to the Output window.

```
myMovieClip.onReleaseOutside = function () {  
    trace ("onReleaseOutside called");  
};
```

MovieClip.onRollOut

Availability

Flash Player 6.

Usage

myMovieClip.onRollOut

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the pointer rolls outside of a movie clip area.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onRollOut` method that sends a trace action to the Output window.

```
myMovieClip.onRollOut = function () {  
    trace ("onRollOut called");  
};
```

MovieClip.onRollOver

Availability

Flash Player 6.

Usage

myMovieClip.onRollOver

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the pointer rolls over a movie clip area.

You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onRollOver` method that sends a `trace` action to the Output window.

```
myMovieClip.onRollOver = function () {  
    trace ("onRollOver called");  
};
```

MovieClip.onSetFocus

Availability

Flash Player 6.

Usage

```
myMovieClip.onSetFocus = function(oldFocus){  
    statements;  
};
```

Parameters

oldFocus The object to lose focus.

Returns

Nothing.

Description

Event handler; invoked when a movie clip receives keyboard focus. The *oldFocus* parameter is the object that loses the focus. For example, if the user presses the Tab key to move the input focus from a movie clip to a text field, *oldFocus* contains the movie clip instance.

If there is no previously focused object, *oldFocus* contains a `null` value.

MovieClip.onUnload

Availability

Flash Player 6.

Usage

```
myMovieClip.onUnload
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked in the first frame after the movie clip is removed from the Timeline. The actions associated with the `onUnload` event are processed before any actions are attached to the affected frame. You must define a function that executes when the event is invoked.

Example

The following example defines a function for the `onUnload` method that sends a trace action to the Output window.

```
myMovieClip.onUnload = function () {  
    trace ("onUnload called");  
};
```

MovieClip._parent

Availability

Flash Player 6.

Usage

```
myMovieClip._parent.property  
_parent._parent.property
```

Description

Property; specifies or returns a reference to the movie clip or object that contains the current movie clip or object. The current object is the object containing the ActionScript code that references `_parent`. Use the `_parent` property to specify a relative path to movie clips or objects that are above the current movie clip or object.

See also

`_root`, `targetPath`

MovieClip.play

Availability

Flash Player 5.

Usage

```
myMovieClip.play()
```

Parameters

None.

Returns

Nothing.

Description

Method; moves the playhead in the Timeline of the movie clip.

See also

`play`

MovieClip.prevFrame

Availability

Flash Player 5.

Usage

```
myMovieClip.prevFrame()
```


Parameters

None.

Returns

Nothing.

Description

Method; sends the playhead to the previous frame and stops it.

See also

`prevFrame`

MovieClip.removeMovieClip

Availability

Flash Player 5.

Usage

myMovieClip.removeMovieClip()

Parameters

None.

Returns

Nothing.

Description

Method; removes a movie clip instance created with the `duplicateMovieclip` action, or the `duplicateMovieClip` or `attachMovie` methods of the `MovieClip` object.

See also

`MovieClip.attachMovie`, `MovieClip.attachMovie`, `removeMovieClip`,
`MovieClip.attachMovie`

MovieClip._rotation

Availability

Flash Player 4.

Usage

myMovieClip._rotation

Description

Property; specifies the rotation of the movie clip in degrees.

MovieClip.setMask

Availability

Flash Player 6.

Usage

myMovieClip.setMask (maskMovieClip)

Parameters

myMovieClip The instance name of a movie clip to be masked.

maskMovieClip The instance name of a movie clip to be a mask.

Returns

Nothing.

Description

Method; makes the movie clip in the parameter *maskMovieClip* into a mask that reveals the movie clip specified by the *myMovieClip* parameter.

The `setMask` method allows multiple-frame movie clips with complex, multilayered content to act as masks. You can shut masks on and off at runtime. However, you can't use the same mask for multiple maskees (which is possible by using mask layers). If you have device fonts in a masked movie clip, they are drawn but not masked. You can't set a movie clip to be its own mask, for example `mc.setMask(mc)`.

If you create a mask layer that contains a movie clip, and then apply the `setMask` method to it, the `setMask` call takes priority and this is not reversible. For example, you could have a movie clip in a mask layer called `UIMask` that masks another layer containing another movie clip called `UIMaskee`. If, as the movie plays, you call `UIMask.setMask(UIMaskee)`, from that point on, `UIMask` is masked by `UIMaskee`.

To cancel a mask created with `ActionScript`, pass the value `null` to the `setMask` method. The following code cancels the mask without affecting the mask layer in the Timeline.

```
UIMask.setMask(null)
```

Example

The following sample code uses the movie clip `circleMask` to mask the movie clip `theMaskee`.

```
theMaskee.setMask(circleMask);
```

MovieClip._soundbuftime

Availability

Flash Player 6.

Usage

```
myMovieClip._soundbuftime
```

Description

Property (global); an integer that specifies the number of seconds a sound prebuffers before it starts to stream.

MovieClip.startDrag

Availability

Flash Player 5.

Usage

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

Parameters

lock A Boolean value specifying whether the draggable movie clip is locked to the center of the mouse position (*true*), or locked to the point where the user first clicked on the movie clip (*false*). This parameter is optional.

left, top, right, bottom Values relative to the coordinates of the movie clip's parent that specify a constraint rectangle for the movie clip. These parameters are optional.

Returns

Nothing.

Description

Method; lets the user drag the specified movie clip. The movie remains draggable until explicitly stopped by calling the `stopDrag` method, or until another movie clip is made draggable. Only one movie clip is draggable at a time.

See also

`MovieClip._droptarget`, `MovieClip.startDrag`, `MovieClip.stopDrag`

MovieClip.stop

Availability

Flash Player 5.

Usage

```
myMovieClip.stop()
```

Parameters

None.

Returns

Nothing.

Description

Method; stops the movie clip currently playing.

See also

`stop`

MovieClip.stopDrag

Availability

Flash Player 5.

Usage

```
myMovieClip.stopDrag()
```

Parameters

None.

Returns

Nothing.

Description

Method; ends a `startDrag` method. A movie that was made draggable with the `startDrag` method remains draggable until a `stopDrag` method is added, or until another movie becomes draggable. Only one movie clip is draggable at a time.

See also

`MovieClip._droptarget`, `MovieClip.startDrag`, `stopDrag`

MovieClip.swapDepths

Availability

Flash Player 5.

Usage

```
myMovieClip.swapDepths(depth)
```

```
myMovieClip.swapDepths(target)
```

Parameters

target The movie clip instance whose depth is swapped by the instance specified in *myMovieClip*. Both instances must have the same parent movie clip.

depth A number specifying the depth level where *MovieClip* is to be placed.

Returns

Nothing.

Description

Method; swaps the stacking, or *z* order (depth level), of the specified instance (*MovieClip*) with the movie specified by the *target* parameter, or with the movie that currently occupies the depth level specified in the *depth* parameter. Both movies must have the same parent movie clip. Swapping the depth level of movie clips has the effect of moving one movie in front of or behind the other. If a movie is tweening when this method is called, the tweening is stopped.

See also

`_level`

MovieClip.tabChildren

Availability

Flash Player 6.

Usage

```
myMovieClip.tabChildren
```

Description

Property; undefined by default. If `tabChildren` is undefined or `true`, then the children of a movie clip are included in automatic tab ordering. If the value of `tabChildren` is `false`, the children of a movie clip are not included in automatic tab ordering.

Example

A list box UI widget built as a movie clip contains several items. Each item may be clicked on to select it, so each item is a button. However, only the list box itself should be a tabstop. The items inside the list box should be excluded from tab ordering. To do this, the `tabChildren` property of the list box should be set to `false`.

The `tabChildren` property has no effect if the `tabIndex` property is used; it only affects automatic tab ordering.

See also

`Button.tabIndex`, `TextField.tabIndex`

MovieClip.tabEnabled

Availability

Flash Player 6.

Usage

MovieClip.tabEnabled

Description

Property; may be set on an instance of the `MovieClip`, `Button`, or `TextField` objects. It is undefined by default.

If the `tabEnabled` property is undefined or `true`, then 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`, then the object is not included in automatic tab ordering. For a movie clip, if `tabEnabled` is `false`, the movie clip's children may still be included in automatic tab ordering, unless the `tabChildren` property is also set to `false`.

See also

`MovieClip.tabChildren`, `MovieClip.tabIndex`

MovieClip.tabIndex

Availability

Flash Player 6.

Usage

myMovieClip.tabIndex

Description

Property; lets you customize the tab ordering of objects in a movie. The `tabIndex` property is undefined by default. You can set `tabIndex` on a button, movie clip, or text field instance.

If an object in a Flash movie contains a `tabIndex` property, automatic tab ordering is disabled, and the tab ordering is calculated from the `tabIndex` properties of objects in the movie. The custom tab ordering only includes objects that have `tabIndex` properties.

The `tabIndex` property must be a positive integer. The objects are ordered according to their `tabIndex` properties, in ascending order. An object with a `tabIndex` of 1 precedes an object with a `tabIndex` of 2. The custom tab ordering disregards the hierarchical relationships of objects in a Flash movie. All objects in the Flash movie with `tabIndex` properties are placed in the tab order. You shouldn't use the same `tabIndex` value for multiple objects.

MovieClip._target

Availability

Flash Player 4.

Usage

myMovieClip._target

Description

Property (read-only); returns the target path of the movie clip instance specified in the *MovieClip* parameter.

MovieClip._totalframes

Availability

Flash Player 4.

Usage

myMovieClip._totalframes

Description

Property (read-only); returns the total number of frames in the movie clip instance specified in the *MovieClip* parameter.

MovieClip.trackAsMenu

Availability

Flash Player 6.

Usage

myMovieClip.trackAsMenu

Description

Property; a Boolean property that indicates whether or not other buttons or movie clips can receive mouse release events. This allows you to create menus. You can set the *trackAsMenu* property on any button or movie clip object. If the *trackAsMenu* property does not exist, the default behavior is *false*.

You can change the *trackAsMenu* property at any time; the modified button movie clip immediately takes on the new behavior.

See also

Button.trackAsMenu

MovieClip.unloadMovie

Availability

Flash Player 5.

Usage

myMovieClip.unloadMovie()

Parameters

None.

Returns

Nothing.

Description

Method; removes a movie clip loaded with the `loadMovie` or `attachMovie` `MovieClip` methods.

See also

`MovieClip.attachMovie`, `MovieClip.loadMovie`, `unloadMovie`, `unloadMovieNum`

MovieClip._url

Availability

Flash Player 4.

Usage

myMovieClip._url

Description

Property (read only); retrieves the URL of the SWF file from which the movie clip was downloaded.

MovieClip.useHandCursor

Availability

Flash Player 6.

Usage

myMovieClip.useHandCursor

Description

Property; a Boolean value that indicates whether the hand cursor displays when a user rolls over a button movie clip. The default value of `useHandCursor` is `true`. If `useHandCursor` is set to `true`, the standard hand cursor used for buttons is displayed when a user rolls over a button movie clip. If `useHandCursor` is `false`, the arrow cursor is used instead.

You can change the `useHandCursor` property at any time; the modified button movie clip immediately takes on the new cursor behavior. The `useHandCursor` property can be read out of a prototype object.

MovieClip._visible

Availability

Flash Player 4.

Usage

myMovieClip._visible

Description

Property; a Boolean value that indicates whether the movie specified by the `MovieClip` parameter is visible. Movie clips that are not visible (`_visible` property set to `false`) are disabled. For example, a button in a movie clip with the `_visible` property set to `false` cannot be clicked.

MovieClip._width

Availability

Flash Player 4 as a read-only property.

Usage

myMovieClip._width

Description

Property; sets and retrieves the width of the movie clip, in pixels.

Example

The following example sets the height and width properties of a movie clip when the user clicks the mouse.

```
onClipEvent(mouseDown) {  
    _width=200;  
    _height=200;  
}
```

See also

MovieClip._height

MovieClip._x

Availability

Flash Player 3.

Usage

myMovieClip._x

Description

Property; an integer that sets the *x* coordinate of movie relative to the local coordinates of the parent movie clip. If a movie clip is in the main Timeline, then its coordinate system refers to the upper left corner of the Stage as (0, 0). If the movie clip is inside another movie clip that has transformations, the movie clip is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90° counterclockwise, the movie clip's children inherit a coordinate system that is rotated 90° counterclockwise. The movie clip's coordinates refer to the registration point position.

See also

MovieClip._xscale, *MovieClip._y*, *MovieClip._yscale*

MovieClip._xmouse

Availability

Flash Player 5.

Usage

myMovieClip._xmouse

Description

Property (read-only); returns the *x* coordinate of the mouse position.

See also

Mouse (object), MovieClip._ymouse

MovieClip._xscale

Availability

Flash Player 4.

Usage

myMovieClip._xscale

Description

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

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

See also

MovieClip._x, MovieClip._y, MovieClip._yscale

MovieClip._y

Availability

Flash Player 3.

Usage

myMovieClip._y

Description

Property; sets the *y* coordinate of movie relative to the local coordinates of the parent movie clip. If a movie clip is in the main Timeline, then its coordinate system refers to the upper left corner of the Stage as (0, 0). If the movie clip is inside another movie clip that has transformations, the movie clip is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90° counterclockwise, the movie clip's children inherit a coordinate system that is rotated 90° counterclockwise. The movie clip's coordinates refer to the registration point position.

See also

MovieClip._x, MovieClip._xscale, MovieClip._yscale

MovieClip._ymouse

Availability

Flash Player 5.

Usage

myMovieClip._ymouse

Description

Property (read-only); indicates the *y* coordinate of the mouse position.

See also

Mouse (object), MovieClip._xmouse

MovieClip._yscale

Availability

Flash Player 4.

Usage

myMovieClip._yscale

Description

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

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

See also

MovieClip._x, MovieClip._xscale, MovieClip._y

NaN

Availability

Flash Player 5.

Usage

NaN

Description

Variable; a predefined variable with the IEEE-754 value for NaN (Not a Number).

ne (not equal – string specific)

Availability

Flash Player 4. This operator has been deprecated in favor of the `!=` (inequality) operator.

Usage

expression1 ne expression2

Parameters

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

Returns

Nothing.

Description

Operator (comparison); compares *expression1* to *expression2* and returns true if *expression1* is not equal to *expression2*; otherwise, returns false.

See also

`!=` (inequality)

new

Availability

Flash Player 5.

Usage

```
new constructor()
```

Parameters

constructor A function followed by any optional parameters in parentheses. The function is usually the name of the object type (for example, Array, Math, Number, or Object) to be constructed.

Returns

Nothing.

Description

Operator; creates a new, initially anonymous, object and calls the function identified by the *constructor* parameter. The *new* operator passes to the function any optional parameters in parentheses, as well as the newly created object, which is referenced using the keyword *this*. The constructor function can then use *this* to set the variables of the object.

The *prototype* property of the constructor function is copied into the *__proto__* property of the new object. As a result, the new object supports all of the methods and properties specified in the Prototype object of the constructor function.

Example

The following example creates the *Book* function and then uses the *new* operator to create the objects *book1* and *book2*.

```
function Book(name, price){
    this.name = name;
    this.price = price;
}

book1 = new Book("Confederacy of Dunces", 19.95);
book2 = new Book("The Floating Opera", 10.95);
```

Example

The following example uses the *new* operator to create an instance of the *Array* object with 18 elements:

```
golfCourse = new Array(18);
```

See also

[] (array access), {} (object initializer)

The constructor method section within each object entry.

newline

Availability

Flash Player 4.

Usage

```
newline
```

Parameters

None.

Returns

Nothing.

Description

Constant; inserts a carriage return character () that inserts a blank line into the ActionScript code. Use `newline` to make space for information that is retrieved by a function or action in your code.

nextFrame

Availability

Flash 2.

Usage

```
nextFrame()
```

Parameters

None.

Returns

Nothing.

Description

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

Example

In this example, when the user clicks the button, the playhead goes to the next frame and stops.

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

nextScene

Availability

Flash 2.

Usage

```
nextScene()
```

Parameters

None.

Returns

Nothing.

Description

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

Example

In this example, when a user releases the button, the playhead is sent to Frame 1 of the next scene.

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

See also

prevScene

not

Availability

Flash Player 4. This operator has been deprecated in favor of the `!` (logical NOT) operator.

Usage

`not expression`

Parameters

expression A variable or other expression that converts to a Boolean value.

Description

Operator; performs a logical NOT operation in Flash Player 4.

See also

`!` (logical NOT)

null

Availability

Flash Player 5.

Usage

`null`

Parameters

None.

Returns

Nothing.

Description

Keyword; a special value that can be assigned to variables, or returned by a function if no data was provided. You can use `null` to represent values that are missing or do not have a defined data type.

Example

In a numeric context, `null` evaluates to 0. Equality tests can be performed with `null`. In this statement, a binary tree node has no left child, so the field for its left child could be set to `null`.

```
if (tree.left == null) {  
    tree.left = new TreeNode();  
}
```

Number (function)

Availability

Flash Player 4.

Usage

`Number(expression)`

Parameters

expression An expression to convert to a number.

Returns

Nothing.

Description

Function; converts the parameter *expression* to a number and returns a value as follows:

If *expression* is a number, the return value is *expression*.

If *expression* is a Boolean value, the return value is 1 if *expression* is true, 0 if *expression* is false.

If *expression* is a string, the function attempts to parse *expression* as a decimal number with an optional trailing exponent, that is, 1.57505e-3.

If *expression* is undefined, the return value is 0.

This function is used to convert Flash 4 files containing deprecated operators that are imported into the Flash 5 authoring environment. See the & operator for more information.

See also

Number (object)

Number (object)

The Number object is a simple wrapper object for the number data type, which means that you can manipulate primitive numeric values using the methods and properties associated with the Number object. This object is identical to the JavaScript Number object. In Flash MX, the Number object has become a native object. As such, you will experience dramatic improvement in performance.

You must use a constructor when calling the methods of the Number object, but you do not need to use the constructor when calling the properties of the Number object. The following examples specify the syntax for calling the methods and properties of the Number object.

The following example calls the `toString` method of the Number object, which returns the string "1234".

```
myNumber = new Number(1234);  
myNumber.toString();
```

This example calls the `MIN_VALUE` property (also called a constant) of the Number object:

```
smallest = Number.MIN_VALUE
```

Method summary for the Number object

Method	Description
<code>Number.toString</code>	Returns the string representation of a Number object.
<code>Number.valueOf</code>	Returns the primitive value of a Number object.

Property summary for the Number object

Property	Description
<code>Number.MAX_VALUE</code>	Constant representing the largest representable number (double-precision IEEE-754). This number is approximately 1.7976931348623158e+308.
<code>Number.MIN_VALUE</code>	Constant representing the smallest representable number (double-precision IEEE-754). This number is approximately 5e-324.
<code>Number.NaN</code>	Constant representing the value for Not a Number (NaN).
<code>Number.NEGATIVE_INFINITY</code>	Constant representing the value for negative infinity.
<code>Number.POSITIVE_INFINITY</code>	Constant representing the value for positive infinity. This value is the same as the global variable <code>Infinity</code> .

Constructor for the Number object

Availability

Flash Player 5.

Usage

```
myNumber = new Number(value)
```

Parameters

value The numeric value of the Number object being created, or a value to be converted to a number.

Returns

Nothing.

Description

Constructor; creates a new Number object. You must use the Number constructor when using the `toString` and `valueOf` methods of the Number object. You do not use a constructor when using the properties of the Number object. The `new Number` constructor is primarily used as a placeholder. An instance of the Number object is not the same as the `Number` function that converts a parameter to a primitive value.

Example

The following code constructs new Number objects.

```
n1 = new Number(3.4);  
n2 = new Number(-10);
```

See also

`Number` (function)

Number.MAX_VALUE

Availability

Flash Player 5.

Usage

`Number.MAX_VALUE`

Description

Property; the largest representable number (double-precision IEEE-754). This number is approximately 1.79E+308.

Number.MIN_VALUE

Availability

Flash Player 5.

Usage

`Number.MIN_VALUE`

Description

Property; the smallest representable number (double-precision IEEE-754). This number is approximately 5e-324.

Number.NaN

Availability

Flash Player 5.

Usage

`Number.NaN`

Description

Property; the IEEE-754 value representing Not A Number (NaN).

Number.NEGATIVE_INFINITY

Availability

Flash Player 5.

Usage

`Number.NEGATIVE_INFINITY`

Description

Property; returns the IEEE-754 value representing negative infinity.

Negative infinity is a special numeric value that is returned when a mathematical operation or function returns a negative value larger than can be represented.

Number.POSITIVE_INFINITY

Availability

Flash Player 5.

Usage

`Number.POSITIVE_INFINITY`

Description

Property; returns the IEEE-754 value representing positive infinity. This value is the same as the global variable `Infinity`.

Positive infinity is a special numeric value that is returned when a mathematical operation or function returns a value larger than can be represented.

Number.toString

Availability

Flash Player 5.

Usage

`myNumber.toString(radix)`

Parameters

radix Specifies the numeric base (from 2 to 36) to use for the number-to-string conversion. If you do not specify the *radix* parameter, the default value is 10.

Returns

Nothing.

Description

Method; returns the string representation of the specified Number object (*myNumber*).

Example

The following example uses the `Number.toString` method, specifying 2 for the *radix* parameter, and returns a string that contains the binary representation of the number 1000.

```
myNumber = new Number(1000);  
myNumber.toString(2);
```

Number.valueOf

Availability

Flash Player 5.

Usage

`myNumber.valueOf()`

Parameters

None.

Returns

Nothing.

Description

Method; returns the primitive value type of the specified Number object.

Object (object)

The generic Object object is at the root of the ActionScript class hierarchy. The generic ActionScript Object object contains a small subset of the features provided by the JavaScript Object object. In Flash MX, the Object object has become a native object. As such, you will experience dramatic improvement in performance.

The generic Object object is supported in Flash Player 5.

Method summary for the Object object

Method	Description
<code>Object.addProperty</code>	Creates a getter/setter property on an object.
<code>Object.registerClass</code>	Assigns an ActionScript class to a movie clip instance.
<code>Object.toString</code>	Converts the specified object to a string, and returns it.
<code>Object.unwatch</code>	Removes the registration that an <code>Object.watch</code> method created.
<code>Object.valueOf</code>	Returns the primitive value of an Object object.
<code>Object.watch</code>	Registers a callback function to be invoked when a specified property of an ActionScript object changes.

Property summary for the Object object

Property	Description
<code>Object.__proto__</code>	A reference to the <code>prototype</code> property of the object's constructor function.

Constructor for the Object object

Availability

Flash Player 5.

Usage

```
new Object([value])
```

Parameters

value A number, Boolean value, or string to be converted to an object. This parameter is optional. If you do not specify *value*, the constructor creates a new object with no defined properties.

Description

Constructor; creates a new Object object.

Object.addProperty

Availability

Flash Player 6.

Usage

```
myObject.addProperty( prop, getFunc, setFunc )
```

Parameters

prop The name of the object property to create.

getFunc The function that is invoked to retrieve the value of the property; this parameter is a function object.

setFunc The function that is invoked to set the value of the property; this parameter is a function object. If you pass the value `null` for this parameter, the property is read-only.

Returns

Returns a value of `true` if the property is successfully created; otherwise, returns `false`.

Description

Method; Creates a getter/setter property. When Flash reads a getter/setter property, it invokes the get function and the function's return value becomes a value of *prop*. When Flash writes a getter/setter property, it invokes the set function and passes it the new value as a parameter. If a property with the given name already exists, the new property overwrites it.

A get function is a function with no parameters. Its return value can be of any type. Its type can change between invocations. The return value is treated as the current value of the property.

A set function is a function that takes one parameter, which is the new value of the property. For instance, if property `x` is assigned by the statement `x = 1`, the set function is passed the parameter 1 of type number. The return value of the setter function is ignored.

You can add getter/setter properties to prototype objects. If you add a getter/setter property to a prototype object, all object instances that inherit the prototype object inherit the getter/setter property. This makes it possible to add a getter/setter property in one location, the prototype object, and have it propagate to all instances of a class (much like adding methods to prototype objects). If a get/set function is invoked for a getter/setter property in an inherited prototype object, the reference passed to the get/set function will be the originally referenced object, not the prototype object.

If invoked incorrectly, `Object.addProperty` may fail with an error. The following table describes errors that may occur:

Error Condition	What Happens
<code>prop</code> is not a valid property name; for instance, an empty string.	Returns <code>false</code> and the property is not added.
<code>getFunc</code> is not a valid function object.	Returns <code>false</code> and the property is not added.
<code>setFunc</code> is not a valid function object.	Returns <code>false</code> and the property is not added.

Example

Usage 1: The built-in properties `TextField.scroll` and `TextField.maxscroll` are getter/setter properties. The `TextField` object has internal methods `getScroll`, `setScroll` and `getMaxScroll`. The `TextField` constructor creates the getter/setter properties and points them to the internal get/set methods, as in the following:

```
this.addProperty("scroll", this.getScroll, this.setScroll);
this.addProperty("maxscroll", this.getMaxScroll, null);
```

When a script retrieves the value of `myTextField.scroll`, the `ActionScript` interpreter automatically invokes `myTextField.getScroll`. When a script modifies the value of `myTextField.scroll`, the interpreter invokes `myTextField.setScroll`. The `maxscroll` property does not specify a set function, so attempts to modify `maxscroll` are ignored.

Usage 2: The above example of `TextField.scroll` and `TextField.maxscroll` work, but the properties `scroll` and `maxscroll` are added to every instance of the `TextField` object. That means the cost of having the properties is two property slots for every instance of the object. If there are many properties like `scroll` and `maxscroll` in a class, they could consume a great deal of memory. Instead, you can add the `scroll` and `maxscroll` properties to `TextField.prototype`:

```
TextField.prototype.addProperty("scroll", this.getScroll, this.setScroll);
TextField.prototype.addProperty("maxscroll", this.getMaxScroll, null);
```

Now, the `scroll` and `maxscroll` properties only exist in one place: the `TextField.prototype` object. The effect, however, is the same as the above code that added `scroll` and `maxscroll` directly to every instance. If `scroll` or `maxscroll` is accessed in a `TextField` instance, the prototype chain is ascended and the getter/setter property in `TextField.prototype` is found.

Object.__proto__

Availability

Flash Player 5.

Usage

```
myObject.__proto__
```

Parameters

None.

Description

Property; refers to the `prototype` property of the constructor function that created *myObject*. The `__proto__` property is automatically assigned to all objects when they are created. The ActionScript interpreter uses the `__proto__` property to access the `prototype` property of the object's constructor function to find out what properties and methods the object inherits from its class.

Object.registerClass

Availability

Flash Player 6

Usage

```
Object.registerClass(symbolID, theClass)
```

Parameters

symbolID The linkage identifier of the movie clip symbol, or the string identifier for the ActionScript class.

theClass A reference to the constructor function of the ActionScript class, or `null` to unregister the symbol.

Returns

If the class registration succeeds, a value of `true` is returned; otherwise, `false` is returned.

Decription

Method; associates a movie clip symbol with an ActionScript object class. If a symbol doesn't exist, Flash creates an association between a string identifier and an object class.

When an instance of the specified movie clip symbol is placed by the Timeline, it is registered to the class specified by the *theClass* parameter rather than to class `MovieClip`.

When an instance of the specified movie clip symbol is created using the `attachMovie` or `duplicateMovieClip` methods, it is registered to the class specified by the *theClass* parameter rather than to class `MovieClip`.

If *theClass* is null, `Object.registerClass` removes any `ActionScript` class definition associated with the specified movie clip symbol or class identifier. For movie clip symbols, any existing instances of the movie clip remain unchanged, but new instances of the symbol are associated with the default class `MovieClip`.

If a symbol is already registered to a class, the `Object.registerClass` method replaces it with the new registration.

When a movie clip instance is placed by the Timeline or created using `attachMovie` or `duplicateMovieClip`, `ActionScript` invokes the constructor for the appropriate class with the keyword `this` pointing to the object. The constructor function is invoked with no parameters.

If you use the `Object.registerClass` method to register a movie clip with an `ActionScript` class other than `MovieClip`, the movie clip symbol doesn't inherit the methods, properties, and events of the built-in `MovieClip` class unless you include the `MovieClip` class in the prototype chain of the new class. The following code creates a new `ActionScript` class called *theClass* that inherits the properties of the `MovieClip` class:

```
theClass.prototype = new MovieClip();
```

Example

This example creates a component for a standard check box UI widget.

First you create a movie clip symbol called Check Box in the library. Then you create a CheckBox class in ActionScript which will represent the check box.

```
// Define constructor for (and thus define)
    CheckBox class

function CheckBox() {
...
}

// Set CheckBox prototype chain
    to inherit from MovieClip

CheckBox.prototype = new MovieClip();

// Define methods for the CheckBox class

CheckBox.prototype.setLabel = function (newLabel) {
    this.label = newLabel;
    ...
};
CheckBox.prototype.setValue = function (newValue) {
    this.value = value;
    ...
};
CheckBox.prototype.getValue = function () {
    return this.value;
};
CheckBox.prototype.getLabel = function () {
    return this.label;
};
```

Now you must associate the CheckBox class with the Check Box movie clip symbol. First, you need the ability to identify the Check Box movie clip symbol with ActionScript. To do this, enter an identifier in the Linkage dialog box in the library and select Export for ActionScript.

Next, write ActionScript to associate the CheckBox class with the Check Box symbol:

```
Object.registerClass("CheckBox" /*symbolID*/, CheckBox /*theClass*/ );
```

Usage 1(Timeline placement):You can now place instances of CheckBox on the stage in the authoring tool, and at runtime the instances will automatically receive the ActionScript class CheckBox. If you place two instances, myCheckBox1 and myCheckBox2, you can control them by invoking methods, as in the following:

```
myCheckBox1.setValue(true);
myCheckBox2.setValue(false);
myCheckBox2.setLabel("new label for #2");
```

Usage 2 (Dynamic instances): You can use the attachMovie method to create a new instance of the check box on the Stage as the movie plays. Because the Check Box symbol is registered to the ActionScript class CheckBox, the new dynamic instance will automatically receive that class.

```
// createCheckBox is a helper function that
// dynamically creates CheckBoxes
function createCheckBox(name, depth) {
    attachMovie("CheckBox", name, depth);
}
createCheckBox("myCheckBox3", 100);
myCheckBox3.setValue(false);
myCheckBox3.setLabel("new label for #3");
```

See also

`MovieClip.attachMovie`, `MovieClip.duplicateMovieClip`

Object.toString

Availability

Flash Player 5.

Usage

```
myObject.toString()
```

Parameters

None.

Returns

Nothing.

Description

Method; converts the specified object to a string and returns it.

Object.unwatch

Availability

Flash Player 6.

Usage

```
myObject.unwatch (prop)
```

Parameters

prop The name of the object property that should no longer be watched, as a string.

Returns

A boolean.

Description

Method; removes a watchpoint that the `Object.watch` method created. This method returns a value of `true` if the watchpoint was successfully removed; otherwise, it returns a `false` value.

Object.valueOf

Availability

Flash Player 5.

Usage

```
myObject.valueOf()
```

Parameters

None.

Returns

Nothing.

Description

Method; returns the primitive value of the specified object. If the object does not have a primitive value, the object itself is returned.

Object.watch

Availability

Flash Player 6.

Usage

```
myObject.watch( prop, callback [, userData] )
```

Parameters

prop A string indicating the name of the object property to watch.

callback The function to invoke when the watched property changes. This parameter is a function object, not a function name as a string. The form of *callback* is *callback(prop, oldval, newval, userData)*.

userData An arbitrary piece of ActionScript data that is passed to the *callback* method. If the *userData* parameter is omitted, *undefined* is passed to the callback method. This parameter is optional.

Returns

A value of *true* if the watchpoint is created successfully, otherwise, returns a *false* value.

Description

Method; registers a callback function to be invoked when a specified property of an ActionScript object changes. When the property changes, the callback function is invoked with *myObject* as the containing object.

A watchpoint can filter (or nullify) the value assignment, by returning a modified *newval* (or *oldval*). If you delete a property for which a watchpoint has been set, that watchpoint does not disappear. If you later recreate the property, the watchpoint is still in effect. To remove a watchpoint, use the *Object.unwatch* method.

Only a single watchpoint may be registered on a property. Subsequent calls to *Object.watch* on the same property replace the original watchpoint.

The *Object.watch* method behaves similarly to the *Object.watch* function in Netscape JavaScript 1.2 and later. The primary difference is the *userData* parameter, which is a Flash addition to *Object.watch* that Netscape Navigator does not support. You can pass the *userData* parameter to the callback function and use it in the callback function.

The *Object.watch* method cannot watch getter/setter properties. Getter/setter properties operate through “lazy evaluation”—the value of the property is not determined until the property is actually queried. “Lazy evaluation” is often efficient because the property is not constantly updated; it is, rather, evaluated when needed. However, *Object.watch* needs to evaluate a property in order to fire watchpoints on it. To work with a getter/setter property, *Object.watch* needs to evaluate the property constantly, which is inefficient.

Generally, ActionScript predefined properties, such as *_x*, *_y*, *_width* and *_height*, are getter/setter properties, and thus cannot be watched with *Object.watch*.

Example

This example shows a `CheckBox` component with methods that set the label or value of each check box instance:

```
myCheckBox1.setValue(true);
myCheckBox1.setLabel("new label");
...
```

It's convenient to think of the value and label of a check box as properties. It's possible to use `Object.watch` to make accessing the value and label look like property access rather than method invocation, as in the following:

```
// Define constructor for (and thus define) CheckBox class
function CheckBox() {
    ...
    this.watch('value', function (id, oldval, newval)) {
        ...
    }
    this.watch('label', function (id, oldval, newval)) {
        ...
    }
}
```

When the value or label property is modified, the function specified by the component is invoked to perform any tasks needed to update the appearance and state of the component to reflect its new parameters. Therefore, the following assignment statement uses an `Object.watch` handler to notify the component that the variable has changed and causes it to update its graphical representation.

```
myCheckBox1.value = false;
```

This syntax is more concise than the former syntax:

```
myCheckBox1.setValue(false);
```

See also

`Object.addProperty`, `Object.unwatch`

onClipEvent

Availability

Flash Player 5.

Usage

```
onClipEvent(movieEvent){
    statement(s);
}
```

Parameters

A *movieEvent* is a trigger called an *event*. When the event takes place, the statements following it within curly brackets are executed. Any of the following values can be specified for the *movieEvent* parameter:

- `load` The action is initiated as soon as the movie clip is instantiated and appears in the Timeline.
- `unload` The action is initiated in the first frame after the movie clip is removed from the Timeline. The actions associated with the `Unload` movie clip event are processed before any actions are attached to the affected frame.

- **enterFrame** The action is triggered continually at the frame rate of the movie. The actions associated with the `enterFrame` clip event are processed before any frame actions that are attached to the affected frames.
- **mouseMove** The action is initiated every time the mouse is moved. Use the `_xmouse` and `_ymouse` properties to determine the current mouse position.
- **mouseDown** The action is initiated when the left mouse button is pressed.
- **mouseUp** The action is initiated when the left mouse button is released.
- **keyDown** The action is initiated when a key is pressed. Use the `Key.getCode` method to retrieve information about the last key pressed.
- **keyUp** The action is initiated when a key is released. Use the `Key.getCode` method to retrieve information about the last key pressed.
- **data** The action is initiated when data is received in a `loadVariables` or `loadMovie` action. When specified with a `loadVariables` action, the `data` event occurs only once, when the last variable is loaded. When specified with a `loadMovie` action, the `data` event occurs repeatedly, as each section of data is retrieved.

statement(s) The instructions to execute when the *mouseEvent* takes place.

Description

Event handler; triggers actions defined for a specific instance of a movie clip.

Example

The following statement includes the script from an external file when the movie is exported; the actions in the included script are run when the movie clip they are attached to loads:

```
onClipEvent(load) {
    #include "myScript.as"
}
```

The following example uses `onClipEvent` with the `keyDown` movie event. The `keyDown` movie event is usually used in conjunction with one or more methods and properties of the `Key` object. The script below uses the `Key.getCode` method to find out which key the user has pressed; if the pressed key matches the `Key.RIGHT` property, the movie is sent to the next frame; if the pressed key matches the `Key.LEFT` property, the movie is sent to the previous frame.

```
onClipEvent(keyDown) {
    if (Key.getCode() == Key.RIGHT) {
        _parent.nextFrame();
    } else if (Key.getCode() == Key.LEFT){
        _parent.prevFrame();
    }
}
```

The following example uses `onClipEvent` with the `mouseMove` movie event. The `_xmouse` and `_ymouse` properties track the position of the mouse each time the mouse moves.

```
onClipEvent(mouseMove) {
    stageX=_root.xmouse;
    stageY=_root.ymouse;
}
```

See also

`Key` (object), `MovieClip._xmouse`, `MovieClip._ymouse`, `on`

on

Availability

Flash Player 2. Not all events are supported in Flash 2.

Usage

```
on(mouseEvent) {  
    statement(s);  
}
```

Parameters

statement(s) The instructions to execute when the *mouseEvent* takes place.

A *mouseEvent* is a trigger called an “event.” When the event takes place, the statements following it within curly brackets execute. Any of the following values can be specified for the *mouseEvent* parameter:

- `press` The mouse button is pressed while the pointer is over the button.
- `release` The mouse button is released while the pointer is over the button.
- `releaseOutside` The mouse button is released while the pointer is outside the button after the button is pressed while the pointer is inside the button.
- `rollOut` The pointer rolls outside of the button area.
- `rollOver` The mouse pointer rolls over the button.
- `dragOut` While the pointer is over the button, the mouse button is pressed and then rolls outside the button area.
- `dragOver` While the pointer is over the button, the mouse button has been pressed then rolled outside the button and then rolled back over the button.
- `keyPress (“key”)` The specified *key* is pressed. The *key* portion of the parameter is specified using any of the key codes listed in in Appendix B, “Keyboard Keys and Key Code Values” of *Using Flash* or any of the key constants listed in the Property summary for the Key object.

Description

Event handler; specifies the mouse event, or keypress that trigger an action.

Example

In the following script, the `startDrag` action executes when the mouse is pressed and the conditional script is executed when the mouse is released and the object is dropped.

```
on(press) {  
    startDrag("rabbit");  
}  
on(release) {  
    trace(_root.rabbit._y);  
    trace(_root.rabbit._x);  
    stopDrag();  
}
```

See also

`onClipEvent`

or

Availability

Flash 4. This operator has been deprecated in favor of the `||` (logical OR) operator.

Usage

condition1 or *condition2*

Parameters

condition1,2 An expression that evaluates to true or false.

Returns

Nothing.

Description

Operator; evaluates *condition1* and *condition2*, and if either expression is true, then the whole expression is true.

See also

`||` (logical OR), `|` (bitwise OR)

ord

Availability

Flash Player 4. This function has been deprecated in favor of the methods and properties of the String (object).

Usage

`ord(character)`

Parameters

character The character to convert to an ASCII code number.

Returns

Nothing.

Description

String function; converts characters to ASCII code numbers.

See also

String (object)

_parent

Availability

Flash Player 4.

Usage

`_parent.property`
`_parent._parent.property`

Description

Property; specifies or returns a reference to the movie clip or object that contains the current movie clip or object. The current object is the object containing the ActionScript code that references `_parent`. Use `_parent` to specify a relative path to movie clips or objects that are above the current movie clip or object.

Example

In the following example, the movie clip `desk` is a child of the movie clip `classroom`. When the script below executes inside the movie clip `desk`, the playhead will jump to Frame 10 in the Timeline of the movie clip `classroom`.

```
_parent.gotoAndStop(10);
```

See also

`_root`, `targetPath`

parseFloat

Availability

Flash Player 5.

Usage

```
parseFloat(string)
```

Parameters

string The string to read and convert to a floating-point number.

Returns

Nothing.

Description

Function; converts a string to a floating-point number. The function reads, or “parses,” and returns the numbers in a string until it reaches a character that is not a part of the initial number. If the string does not begin with a number that can be parsed, `parseFloat` returns NaN. White space preceding valid integers is ignored, as are trailing nonnumeric characters.

Example

The following examples use the `parseFloat` function to evaluate various types of numbers.

```
parseFloat("-2") returns -2
```

```
parseFloat("2.5") returns 2.5
```

```
parseFloat("3.5e6") returns 3.5e6, or 3500000
```

```
parseFloat("foobar") returns NaN
```

```
parseFloat(" 5.1") returns 5.1
```

```
parseFloat("3.75math") returns 3.75
```

```
parseFloat("0garbage") returns 0
```

parseInt

Availability

Flash Player 5.

Usage

```
parseInt(expression, [radix])
```

Parameters

expression A string to convert to a integer.

radix An integer representing the radix (base) of the number to parse. Legal values are from 2–36. This parameter is optional.

Returns

Nothing.

Description

Function; converts a string to an integer. If the specified string in the parameters cannot be converted to a number, the function returns NaN. Integers beginning with 0 or specifying a radix of 8 are interpreted as octal numbers. Strings beginning with 0x are interpreted as hexadecimal numbers. White space preceding valid integers is ignored, as are trailing nonnumeric characters.

Example

The following examples use the parseInt function to evaluate various types of numbers.

```
parseInt("3.5")  
// returns 3.5
```

```
parseInt("bar")  
// returns NaN
```

```
parseInt("4foo")  
// returns 4
```

The following are examples of hexadecimal conversions:

```
parseInt("0x3F8")  
// returns 1016
```

```
parseInt("3E8", 16)  
// returns 1000
```

The following are examples of a binary conversion:

```
parseInt("1010", 2)  
// returns 10 (the decimal representation of the binary 1010)
```

The following is an example of octal number parsing (in this case the octal number is identified by the radix, 8):

```
parseInt("777", 8)  
// returns 511 (the decimal representation of the octal 777)
```

play

Availability

Flash 2.

Usage

`play()`

Parameters

None.

Returns

Nothing.

Description

Action; moves the playhead forward in the Timeline.

Example

The following code uses an `if` statement to check the value of a name the user enters. If the user enters `Steve`, the `play` action is called and the playhead moves forward in the Timeline. If the user enters anything other than `Steve`, the movie does not play and a text field with the variable name `alert` is displayed.

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

prevFrame

Availability

Flash 2.

Usage

`prevFrame()`

Parameters

None.

Returns

Nothing.

Description

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

Example

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

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

See also

`MovieClip.prevFrame`

prevScene

Availability

Flash 2.

Usage

```
prevScene()
```

Parameters

None.

Returns

Nothing.

Description

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

See also

`nextScene`

print

Availability

Flash Player 4.20.

Usage

```
print (level)
print (level, "Bounding box")
print ("target")
print ("target", "Bounding box")
printAsBitmap (level)
printAsBitmap (level, "Bounding box")
printAsBitmap ("target")
printAsBitmap ("target", "Bounding box")
```

Parameters

print In normal mode in the Actions panel, choose `As vectors` to print frames that do not contain bitmap images or use transparency (alpha) or color effects; choose `As bitmap` to print frames that contain bitmap images, transparency or color effects. If you choose the `As bitmap` Print parameter, the `AsBitmap` syntax is appended to the `print` action in the Actions panel.

level The level in the Flash Player to print. In the Actions panel in normal mode, if you choose a level, the `print` action switches to `printNum` or `printAsBitmapNum`; in expert mode, you must specify either `printNum` or `printAsBitmapNum`. By default, all of the frames in the level print. If you want to print specific frames in the level, assign a `#p` frame label to those frames.

target The instance name of a movie clip to print. By default, all of the frames in the target instance print. If you want to print specific frames in the movie clip, assign a `#p` frame label to those frames.

Bounding box A modifier that sets the print area of the movie. This parameter is optional. You can choose one of the following:

- `bmovie` Designates the bounding box of a specific frame in a movie as the print area for all printable frames in the movie. Assign a `#b` frame label to the frame whose bounding box you want to use as the print area.

- **bmax** Designates a composite of all of the bounding boxes of all the printable frames as the print area. Specify the **bmax** parameter when the printable frames in your movie vary in size.
- **bframe** Designates that the bounding box of each printable frame be used as the print area for that frame. This changes the print area for each frame and scales the objects to fit the print area. Use **bframe** if you have objects of different sizes in each frame and want each object to fill the printed page.

Returns

None.

Description

Action; prints the *target* movie clip according to the boundaries specified in the parameter (**bmovie**, **bmax**, or **bframe**). If you want to print specific frames in the target movie, attach a #P frame label to those frames. Although the **print** action results in higher quality prints than the **printAsBitmap** action, it cannot be used to print movies that use alpha transparencies or special color effects.

If you do not specify a print boundary parameter, the print area is determined by the Stage size of the loaded movie by default. The movie does not inherit the main movie's Stage size. You can control the print area by specifying the **bmovie**, **bmax**, or **bframe** parameters.

All of the printable elements in a movie must be fully loaded before printing can begin.

The Flash Player printing feature supports PostScript and non-PostScript printers. Non-PostScript printers convert vectors to bitmaps.

Example

The following example will print all of the printable frames in the movie clip *myMovie* with the print area defined by the bounding box of the frame with the #b frame label attached:

```
print("myMovie","bmovie");
```

The following example will print all of the printable frames in *myMovie* with a print area defined by the bounding box of each frame:

```
print("myMovie","bframe");
```

See also

printNum, **printAsBitmap**, **printAsBitmapNum**

printAsBitmap

Availability

Flash Player 4.20.

Usage

```
printAsBitmap(target, "Bounding box")
```

Parameters

target The instance name of movie clip to print. By default, all of the frames in the movie are printed. If you want to print specific frames in the movie, attach a #P frame label to those frames.

Bounding box A modifier that sets the print area of the movie. You can choose one of the following parameters:

- **bmovie** Designates the bounding box of a specific frame in a movie as the print area for all printable frames in the movie. Assign a *#b* frame label to the frame whose bounding box you want to use as the print area.
- **bmax** Designates a composite of all of the bounding boxes of all the printable frames as the print area. Specify the **bmax** parameter when the printable frames in your movie vary in size.
- **bframe** Designates that the bounding box of each printable frame be used as the print area for that frame. This changes the print area for each frame and scales the objects to fit the print area. Use **bframe** if you have objects of different sizes in each frame and want each object to fill the printed page.

Returns

None.

Description

Action; prints the *target* movie clip as a bitmap. Use the **printAsBitmap** action to print movies that contain frames with objects that use transparency or color effects. The **printAsBitmap** action prints at the highest available resolution of the printer in order to maintain as much definition and quality as possible.

If your movie does not contain alpha transparencies or color effects, it is recommended that you use the **print** action for better quality results.

By default, the print area is determined by the Stage size of the loaded movie. The movie does not inherit the main movie's Stage size. You can control the print area by specifying the **bmovie**, **bmax**, or **bframe** parameters.

All of the printable elements in a movie must be fully loaded before printing can begin.

The Flash Player printing feature supports PostScript and non-PostScript printers. Non-PostScript printers convert vectors to bitmaps.

See also

print, **printAsBitmapNum**, **printNum**

printAsBitmapNum

Availability

Flash Player 5.

Usage

```
printAsBitmapNum(level)
printAsBitmapNum(level, "Bounding box")
```

Parameters

level The level in the Flash Player to print. By default, all of the frames in the level print. If you want to print specific frames in the level, assign a *#p* frame label to those frames.

Bounding box A modifier that sets the print area of the movie. This parameter is optional. You can choose one of the following parameters:

- **bmovie** Designates the bounding box of a specific frame in a movie as the print area for all printable frames in the movie. Assign a *#b* frame label to the frame whose bounding box you want to use as the print area.

- **bmax** Designates a composite of all of the bounding boxes of all the printable frames as the print area. Specify the **bmax** parameter when the printable frames in your movie vary in size.
- **bframe** Designates that the bounding box of each printable frame be used as the print area for that frame. This changes the print area for each frame and scales the objects to fit the print area. Use **bframe** if you have objects of different sizes in each frame and want each object to fill the printed page.

Returns

None.

Description

Action; prints a level in the Flash Player as a bitmap. Use the `printAsBitmapNum` action to print movies that contain frames with objects that use transparency or color effects. The `printAsBitmapNum` action prints at the highest available resolution of the printer in order to maintain the highest possible definition and quality. To calculate the printable file size of a frame designated to print as a bitmap, multiply pixel width by pixel height by printer resolution.

If your movie does not contain alpha transparencies or color effects, it is recommended that you use the `printNum` action for better quality results.

By default, the print area is determined by the Stage size of the loaded movie. The movie does not inherit the main movie's Stage size. You can control the print area by specifying the **bmovie**, **bmax**, or **bframe** parameters.

All of the printable elements in a movie must be fully loaded before printing can begin.

The Flash Player printing feature supports PostScript and non-PostScript printers. Non-PostScript printers convert vectors to bitmaps.

See also

`print`, `printAsBitmap`, `printNum`

printNum

Availability

Flash Player 5.

Usage

```
printNum (level, "Bounding box")
```

Parameters

level The level in the Flash Player to print. By default, all of the frames in the level print. If you want to print specific frames in the level, assign a **#p** frame label to those frames.

Bounding box A modifier that sets the print area of the movie. You can choose one of the following parameters:

- **bmovie** Designates the bounding box of a specific frame in a movie as the print area for all printable frames in the movie. Assign a **#b** frame label to the frame whose bounding box you want to use as the print area.
- **bmax** Designates a composite of all of the bounding boxes of all the printable frames as the print area. Specify the **bmax** parameter when the printable frames in your movie vary in size.

- `bframe` Designates that the bounding box of each printable frame be used as the print area for that frame. This changes the print area for each frame and scales the objects to fit the print area. Use `bframe` if you have objects of different sizes in each frame and want each object to fill the printed page.

Returns

None.

Description

Action; prints the level in the Flash Player according to the boundaries specified in the *Bounding box* parameter ("`bmovie`", "`bmax`", "`bframe`"). If you want to print specific frames in the target movie, attach a *#P* frame label to those frames. Although using the `printNum` action results in higher quality prints than using the `printAsBitmapNum` action, you cannot use `printNum` to print movies with alpha transparencies or special color effects.

If you do not specify a print boundary parameter, the print area is determined by the Stage size of the loaded movie by default. The movie does not inherit the main movie's Stage size. You can control the print area by specifying the `bmovie`, `bmax`, or `bframe` parameters.

All of the printable elements in a movie must be fully loaded before printing can begin.

The Flash Player printing feature supports PostScript and non-PostScript printers. Non-PostScript printers convert vectors to bitmaps.

See also

`print`, `printAsBitmap`, `printAsBitmapNum`

_quality

Availability

Flash Player 5.

Usage

`_quality`

Description

Property (global); sets or retrieves the rendering quality used for a movie. Device fonts are always aliased and therefore are unaffected by the `_quality` property.

The `_quality` property can be set to the following values:

- "`LOW`" Low rendering quality. Graphics are not anti-aliased, bitmaps are not smoothed.
- "`MEDIUM`" Medium rendering quality. Graphics are anti-aliased using a 2 x 2 grid, in pixels, 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 grid, in pixels, 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 grid, in pixels, and bitmaps are always smoothed.

Example

The following example sets the rendering quality to `LOW`:

```
_quality = "LOW";
```

See also

`_highquality`, `toggleHighQuality`

random

Availability

Flash Player 4. This function is deprecated in Flash 5; use of the `Math.random` method is recommended.

Usage

```
random(value)
```

Parameters

value An integer.

Returns

An integer.

Description

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

Example

The following use of `random` returns a value of 0, 1, 2, 3, or 4:

```
random(5);
```

See also

`Math.random`

removeMovieClip

Availability

Flash Player 4.

Usage

```
removeMovieClip(target)
```

Parameters

target The target path of a movie clip instance created with `duplicateMovieClip`, or the instance name of a movie clip created with the `attachMovie` or `duplicateMovieClip` methods of the `MovieClip` object.

Returns

None.

Description

Action; deletes a movie clip instance that was created with the `attachMovie` or `duplicateMovieClip` methods of the `MovieClip` object, or with the `duplicateMovieClip` action.

See also

`duplicateMovieClip`, `MovieClip.duplicateMovieClip`, `MovieClip.attachMovie`, `MovieClip.removeMovieClip`

return

Availability

Flash Player 5.

Usage

```
return[expression]  
return
```

Parameters

expression A string, number, array, or object to evaluate and return as a value of the function. This parameter is optional.

Returns

The evaluated *expression* parameter, if provided.

Description

Action; specifies the value returned by a function. The `return` action evaluates *expression* and returns the result as a value of the function in which it executes. The `return` action causes the function to stop running and replaces the function with the returned value. If the `return` statement is used alone, it returns `null`.

Example

The following example uses the `return` action inside the body of the `sum` function to return the added value of the three parameters. The next line of code calls the `sum` function and assigns the returned value to the variable `newValue`:

```
function sum(a, b, c){  
    return a + b + c;  
}  
  
newValue = sum(4, 32, 78);  
trace(newValue);  
// sends 114 to the Output window
```

See also

[function](#)

_root

Availability

Flash Player 4.

Usage

```
_root.movieClip  
_root.action  
_root.property
```

Parameters

movieClip The instance name of a movie clip.

action An action or method.

property A property of the MovieClip object.

Description

Property; specifies or returns a reference to the root movie Timeline. If a movie has multiple levels, the root movie Timeline is on the level containing the currently executing script. For example, if a script in level 1 evaluates `_root`, `_level1` is returned.

Specifying `_root` is the same as using the slash notation (`/`) to specify an absolute path within the current level.

Example

The following example stops the Timeline of the level containing the currently executing script:

```
_root.stop();
```

The following example sends the Timeline in the current level to frame 3:

```
_root.gotoAndStop(3);
```

See also

`_parent`, `targetPath`

scroll

Availability

Flash Player 4.

Usage

```
textFieldVariableName.scroll = x
```

Description

Property; a deprecated property that controls the display of information in a text field associated with a variable. The `scroll` property defines where the text field begins displaying content; after you set it, the Flash Player updates it as the user scrolls through the text field. The `scroll` property is useful for directing users to a specific paragraph in a long passage, or creating scrolling text fields. This property can be retrieved and modified.

Example

The following code is attached to an Up button that scrolls the text field `myText`:

```
on (release) {  
    myText.scroll = myText.scroll + 1;  
}
```

See also

`TextField.maxscroll`, `TextField.scroll`

Selection (object)

The Selection object lets you set and control in which text field the cursor is located in a Flash movie. The text field that is said to have “focus” is the field in which the cursor is currently located. Selection-span indexes are zero-based (for example, the first position is 0, the second position is 1, and so on).

There is no constructor method for the Selection object, as there can only be one currently focused field at a time.

Method summary for the Selection object

Method	Description
<code>Selection.addListener</code>	Registers an object to receive notification when the <code>onSetFocus</code> method is invoked.
<code>Selection.getBeginIndex</code>	Returns the index at the beginning of selection span. Returns -1 if there is no index or currently selected field.
<code>Selection.getCaretIndex</code>	Returns the current caret position in the currently focused selection span. Returns -1 if there is no caret position or currently focused selection span.
<code>Selection.getEndIndex</code>	Returns the index at the end of the selection span. Returns -1 if there is no index or currently selected field.
<code>Selection.getFocus</code>	Returns the name of the variable for the currently focused text field. Returns <code>null</code> if there is no currently focused text field.
<code>Selection.removeListener</code>	Removes an object that was registered with <code>addListener</code> .
<code>Selection.setFocus</code>	Focuses the text field associated with the variable specified in the parameter.
<code>Selection.setSelection</code>	Sets the beginning and ending indexes of the selection span.

Listener summary for the Selection object

Method	Description
<code>Selection.onSetFocus</code>	Notified when the input focus changes.

Selection.addListener

Availability

Flash Player 6.

Usage

```
Selection.addListener(newListener)
```

Parameters

newListener An object with an `onSetFocus` method.

Returns

None.

Description

Method; registers an object to receive keyboard focus change notifications. When the focus changes (for example, whenever the `Selection.SetFocus` method is invoked), all listening objects registered with `addListener` have their `onSetFocus` method invoked. Multiple objects may listen for focus change notifications. If the listener *newListener* is already registered, no change occurs.

Selection.getBeginIndex

Availability

Flash Player 5.

Usage

`Selection.getBeginIndex()`

Parameters

None.

Returns

An integer.

Description

Method; returns the index at the beginning of the selection span. If no index exists or no text field currently has focus, the method returns -1. Selection span indexes are zero-based (for example, the first position is 0, the second position is 1, and so on).

Selection.getCaretIndex

Availability

Flash Player 5.

Usage

`Selection.getCaretIndex()`

Parameters

None.

Returns

An integer.

Description

Method; returns the index of the blinking cursor position. If there is no blinking cursor displayed, the method returns -1. Selection span indexes are zero-based (for example, the first position is 0, the second position is 1, and so on).

Selection.getEndIndex

Availability

Flash Player 5.

Usage

`Selection.getEndIndex()`

Parameters

None.

Returns

An integer.

Description

Method; returns the ending index of the currently focused selection span. If no index exists, or if there is no currently focused selection span, the method returns -1. Selection span indexes are zero-based (for example, the first position is 0, the second position is 1, and so on).

Selection.setFocus

Availability

Flash Player 5. Instance names for buttons and text fields work in Flash Player 6.

Usage

```
Selection.setFocus()
```

Parameters

None.

Returns

Nothing.

Description

Method; returns the variable name of the text field that has focus. If no text field has focus, the method returns `null`. If the current focus is a button, and the button is an instance of the `Button` object, `getFocus` returns the target path as a string. If the current focus is a text field, and the text field is an instance of the `TextField` object, `getFocus` returns the target path as a string.

If a button movie clip is the currently focused button, `Selection.getFocus` returns the target path of the button movie clip. If a Text Field with an instance name is currently focused, `Selection.getFocus` returns the target path of the `TextField` object. Otherwise, it returns the Text Field's variable name.

Selection.onSetFocus

Availability

Flash Player 6.

Usage

```
someListener.onSetFocus = function(oldFocus, newFocus){  
  statements;  
}
```

Description

Listener; notified when the input focus changes. To use `onSetFocus`, you must create a listener object. You can then define a function for `onSetFocus` and use the `addListener` method to register the listener with the `Selection` object, as in the following:

```
someListener = new Object();  
someListener.onSetFocus = function () { ... };  
Selection.addListener(someListener);
```

Listeners enable different pieces of code to cooperate because multiple listeners can receive notification about a single event.

See also

`Selection.addListener`

Selection.removeListener

Availability

Flash Player 6.

Usage

`Selection.removeListener(listener)`

Parameters

listener The object that will no longer receive focus notifications.

Returns

If the *listener* was successfully removed, the method returns a `true` value. If the *listener* was not successfully removed, for example if the *listener* was not on the Selection object's listener list, the method returns a value of `false`.

Description

Method; removes an object previously registered with `addListener`.

Selection.setFocus

Availability

Flash Player 5. Instance names for button and text fields only work in Flash Player 6.

Usage

`Selection.setFocus("variablePath")`

Parameters

variablePath A string specifying the path to the name of a variable associated with a text field.

Returns

An event.

Description

Method; focuses the editable text field associated with the variable specified by the *variablePath*. The *variablePath* parameter must be a string literal of the path to that variable. You can use dot or slash notation to specify the path. You can also use a relative or absolute path.

If a target path of a button instance is passed as the *variablePath* parameter, that button becomes the new focus. If a target path of a text field instance is passed as the *variablePath* parameter, that text field becomes the new focus. If `null` is passed, the current focus is removed.

If a button movie clip is passed to `Selection.setFocus`, it becomes the currently focused button. If a `TextField` object is specified, it becomes the current focus. If a `Button` object is specified, it becomes the currently focused button.

Example

The following example sets focus on a text field on the main Timeline that is associated with the *myVar* variable. The *variablePath* parameter is an absolute path, so you can call the action from any Timeline.

```
Selection.setFocus("_root.myVar");
```

In the following example, the text field associated with *myVar* is in a movie clip called *myClip* on the main Timeline. You can use either of the following two paths to set focus; the first is relative and the second is absolute.

```
Selection.setFocus("myClip.myVar");  
Selection.setFocus("_root.myClip.myVar");
```

Selection.setSelection

Availability

Flash Player 5.

Usage

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

Parameters

start The beginning index of the selection span.

end The ending index of the selection span.

Returns

Nothing.

Description

Method; sets the selection span of the currently focused text field. The new selection span will begin at the index specified in the *start* parameter, and end at the index specified in the *end* parameter. Selection span indexes are zero-based (for example, the first position is 0, the second position is 1, and so on). This method has no effect if there is no currently focused text field.

set variable

Availability

Flash Player 4.

Usage

```
set(variable, expression)
```

Parameters

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

expression A value assigned to the variable.

Returns

Nothing.

Description

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

Variables can hold any data type (for example, string, number, Boolean, object, or movie clip). The Timeline of each movie and movie clip has its own set of variables, and each variable has its own value independent of variables on other timelines.

ActionScript is a dynamically typed language. Every variable has a type. The type is assigned at runtime and can change during execution. This is unlike a statically typed language like Java or C++ where the type is assigned at compile time and cannot change at runtime.

Example

This example sets a variable called `orig_x_pos`, which stores the original *x* axis position of the ship movie clip in order to reset the ship to its starting location later in the movie.

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

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

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

See also

`var`, `call`

setInterval

Availability

Flash Player 6.

Usage

```
setInterval( function, interval[, arg1, arg2, ..., argn] )
```

```
setInterval( object, methodName, interval[, arg1, arg2, ..., argn] )
```

Parameters

function A function name or a reference to an anonymous function.

object An object derived from the Object object.

methodName The name of the method to call on the *object* parameter.

interval The time in milliseconds between calls to the *function* or *methodName* parameter.

arg1, arg2, ..., argn Optional parameters passed to the *function* or *methodName* parameter.

Returns

An interval identifier that you can pass to `clearInterval` to cancel the interval.

Description

Action; calls a function or a method or an object at periodic intervals while a movie plays. You can use an interval function to update variables from a database or update a time display.

If *interval* is less than the movie frame rate (for example, 10 frames per second (fps) is equal to 100 milliseconds), the interval function is called as close to *interval* as possible. You must use the `updateAfterEvent` function to make sure that the screen refreshes often enough. If *interval* is greater than the movie frame rate, the interval function is only called each time the playhead enters a frame in order to minimize the impact each time the screen is refreshed.

The first syntax example above is the default syntax for the `setInterval` function in the Actions panel in normal mode. To use the second syntax example, you must use the Actions panel in expert mode.

Example

Usage 1: The following example calls an anonymous function every 1000 milliseconds (every 1 second).

```
setInterval( function(){ trace("interval called"); }, 1000 );
```

Usage 2: The following example defines two callback functions and calls each of them. Both calls to the `setInterval` function send the string "interval called" to the Output window every 1000 milliseconds. The first call to `setInterval` calls the `callback1` function, which contains a trace action. The second call to `setInterval` passes the "interval called" string to the function `callback2` as a parameter.

```
function callback1() {  
    trace("interval called");  
}  
  
function callback2(arg) {  
    trace(arg);  
}  
  
setInterval( callback1, 1000 );  
setInterval( callback2, 1000, "interval called" );
```

Usage 3: This example uses a method of an object. You must use this syntax when you want to call a method that is defined for an object. You can only use this syntax in expert mode.

```
obj = new Object();  
obj.interval = function() {  
    trace("interval function called");  
}  
  
setInterval( obj, "interval", 1000 );  
  
obj2 = new Object();  
obj2.interval = function(s) {  
    trace(s);  
}  
setInterval( obj2, "interval", 1000, "interval function called" );
```

You must use the second form of the `setInterval` syntax to call a method of an object, as follows:

```
setInterval( obj2, "interval", 1000, "interval function called" );
```

See also

`clearInterval`, `updateAfterEvent`

setProperty

Availability

Flash Player 4.

Usage

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

Parameters

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

property The property to be set.

value The new literal value of the property.

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

Returns

Nothing.

Description

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

Example

This statement sets the `_alpha` property of a movie clip named `star` to 30% when the button is clicked:

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

See also

`getProperty`

Sound (object)

The Sound object lets you control sound in a movie. You can add sounds to a movie clip from the Library while the movie is playing and control those sounds. If you do not specify a *target* when you create a new Sound object, you can use the methods to control sound for the whole movie. You must use the constructor `new Sound` to create an instance of the Sound object before calling the methods of the Sound object.

The Sound object is supported in Flash Player 5 and Flash Player 6.

Method summary for the Sound object

Method	Description
<code>Sound.attachSound</code>	Attaches the sound specified in the parameter.
<code>Sound.getBytesLoaded</code>	Returns the number of bytes loaded for the specified sound.
<code>Sound.getBytesTotal</code>	Returns the size of the sound in bytes.
<code>Sound.getPan</code>	Returns the value of the previous <code>setPan</code> call.
<code>Sound.getTransform</code>	Returns the value of the previous <code>setTransform</code> call.
<code>Sound.getVolume</code>	Returns the value of the previous <code>setVolume</code> call.
<code>Sound.loadSound</code>	Loads an MP3 file into the Flash Player.
<code>Sound.setPan</code>	Sets the left/right balance of the sound.
<code>Sound.setTransform</code>	Sets the amount of each channel, left and right, to be played in each speaker.
<code>Sound.setVolume</code>	Sets the volume level for a sound.
<code>Sound.start</code>	Starts playing a sound from the beginning or, optionally, from an offset point set in the parameter.
<code>Sound.stop</code>	Stops the specified sound or all sounds currently playing.

Property summary for the Sound object

Method	Description
<code>Sound.duration</code>	Length of a sound in milliseconds.
<code>Sound.position</code>	Number of milliseconds the sound has been playing.

Event handler summary for the Sound object

Method	Description
<code>Sound.onLoad</code>	Invoked when a sound loads.
<code>Sound.onSoundComplete</code>	Invoked when a sound stops playing.

Constructor for the Sound object

Availability

Flash Player 5.

Usage

```
new Sound([target])
```

Parameters

target The movie clip instance on which the Sound object operates. This parameter is optional.

Returns

Nothing.

Description

Constructor; creates a new Sound object for a specified movie clip. If you do not specify a target instance, the Sound object controls all of the sounds in the movie.

Example

The following example creates a new instance of the Sound object called `GlobalSound`. The second line calls the `setVolume` method and adjusts the volume on all sounds in the movie to 50%.

```
globalsound = new Sound();
globalsound.setVolume(50);
```

The following example creates a new instance of the Sound object, passes it the target movie clip *myMovie*, and calls the `start` method, which starts any sound in *myMovie*.

```
moviesound = new Sound(myMovie);
moviesound.start();
```

Sound.attachSound

Availability

Flash Player 5.

Usage

```
mySound.attachSound(" idName ")
```

Parameters

idName The identifier of an exported sound in the Library. The identifier is located in the Symbol Linkage Properties dialog box.

Returns

Nothing.

Description

Method; attaches the sound specified in the *idName* parameter to the specified Sound object. The sound must be in the library of the current movie and specified for export in the Symbol Linkage Properties dialog box. You must call `Sound.start` to start playing the sound.

See also

`Sound.start`

Sound.duration

Availability

Flash Player 6.

Usage

```
mySound.duration
```

Description

Property (read-only); the duration of a sound in milliseconds.

Sound.getBytesLoaded

Availability

Flash Player 6.

Usage

Sound.getBytesLoaded()

Parameters

None.

Returns

An integer indicating the number of bytes loaded.

Description

Method; returns the number of bytes loaded (streamed) for the specified Sound object. You can compare the value of `getBytesLoaded` with the value of `getBytesTotal` to determine what percentage of a sound has loaded.

See also

`Sound.getBytesTotal`

Sound.getBytesTotal

Availability

Flash Player 6.

Usage

Sound.getBytesTotal()

Parameters

None.

Returns

An integer indicating the total size, in bytes, of the specified Sound object.

Description

Method; returns the size, in bytes, of the specified Sound object.

See also

`Sound.getBytesLoaded`

Sound.getPan

Availability

Flash Player 5.

Usage

mySound.getPan();

Parameters

None.

Returns

Nothing.

Description

Method; returns the pan level set in the last `setPan` call as an integer from -100 (left) to 100 (right). (0 sets the left and right channels equally.) The pan setting controls the left-right balance of the current and future sounds in a movie.

This method is cumulative with the `setVolume` or `setTransform` methods.

See also

`Sound.setPan`

Sound.getTransform

Availability

Flash Player 5.

Usage

```
mySound.getTransform();
```

Parameters

None.

Returns

Nothing.

Description

Method; returns the sound transform information for the specified Sound object set with the last `setTransform` call.

See also

`Sound.setTransform`

Sound.getVolume

Availability

Flash Player 5.

Usage

```
mySound.getVolume()
```

Parameters

None.

Returns

Nothing.

Description

Method; returns the sound volume level as an integer from 0 to 100, where 0 is off and 100 is full volume. The default setting is 100.

See also

`Sound.setVolume`

Sound.loadSound

Availability

Flash Player 6.

Usage

```
mySound.loadSound("url", isStreaming)
```

Parameters

url The location on a server of an MP3 sound file.

isStreaming A Boolean value that indicates whether the sound is a streaming or event sound.

Returns

Nothing.

Description

Method; loads an MP3 file into an instance of the 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 object and respond to all methods and properties of this object.

Streaming sounds play while they are downloading. Playback begins when sufficient data has been received to start the decompressor. As with event sounds, streaming sounds exist only in virtual memory; they are not downloaded to the hard drive.

Example

The following example loads an event sound:

```
s.loadSound( "http://serverpath:port/mp3filename", false);
```

The following example loads a streaming sound:

```
loadSound( "http://serverpath:port/mp3filename", true);
```

Sound.onLoad

Availability

Flash Player 6.

Usage

```
mySoundObject.onLoad = callbackFunction
```

Parameters

mySoundObject A Sound object.

callbackFunction A function.

Returns

Nothing.

Description

Event handler; invoked automatically when a sound loads. You must create a function that executes when the `onLoad` event is invoked. You can use either an anonymous function or a named function.

See also

`Sound.onSoundComplete`

Sound.onSoundComplete

Availability

Flash Player 6.

Usage

mySoundObject.onSoundComplete = callbackFunction

Parameters

mySoundObject A Sound object.

callbackFunction A function.

Returns

Nothing.

Description

Event; invoked automatically when a sound finishes playing. You can use the `onSoundComplete` event to trigger events in a movie based on the completion of a sound.

You must create a function that executes when the `onSoundComplete` event is invoked. You can use either an anonymous function or a named function.

Example

Usage 1: The following example uses an anonymous function:

```
s = new Sound();
s.attachSound("mySound");
s.onSoundComplete = function() { trace("mySound completed"); };
s.start();
```

Usage 2: The following example uses a named function:

```
function callback1() {
    trace("mySound completed");
}

s = new Sound();
s.attachSound("mySound");

s.onSoundComplete = callback1;

s.start();
```

Sound.position

Availability

Flash Player 6.

Usage

mySound.position

Parameters

None.

Returns

Number of milliseconds the sound has been playing.

Description

Property (read-only); returns the number of milliseconds a sound has been playing. If the sound is looped, the position will be reset to 0 at the beginning of each loop.

Sound.setPan

Availability

Flash Player 5.

Usage

```
mySound.setPan(pan);
```

Parameters

pan An integer specifying the left-right balance for a sound. The range of valid values is -100 to 100, where -100 uses only the left channel, 100 uses only the right channel, and 0 balances the sound evenly between the two channels.

Returns

Nothing.

Description

Method; determines how the sound is played in the left and right channels (speakers). For mono sounds, *pan* determines which speaker (left or right) the sound plays through.

Example

The following example creates an instance of the Sound object *s* and attaches a sound with the Identifier *L7* from the Library. It also calls the `setVolume` and `setPan` methods to control the *L7* sound.

```
onClipEvent(mouseDown) {  
    // create a sound object  
    s = new Sound(this);  
    // attach a sound from the library  
    s.attachSound("L7");  
    //set volume to 50%  
    s.setVolume(50);  
    //turn off the sound in the right channel  
    s.setPan(-100);  
    //start 30 seconds into the sound and play it 5 times  
    s.start(30, 5);  
}
```

See also

`Sound.attachSound`, `Sound.setPan`, `Sound.setTransform`, `Sound.setVolume`, `Sound.start`

Sound.setTransform

Availability

Flash Player 5.

Usage

```
mySound.setTransform(soundTransformObject)
```

Parameters

soundTransformObject An object created with the constructor for the generic Object object.

Returns

Nothing.

Description

Method; sets the sound transform, or “balance” information, for a Sound object.

The *soundTransformObject* parameter is an object that you create using the constructor method of the generic Object object 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 the *setTransform* method to play mono sounds as stereo, play stereo sounds as mono, and to add interesting effects to sounds.

The parameters for the *soundTransformObject* are as follows:

- ll A percentage value specifying how much of the left input to play in the left speaker (0–100).
- lr A percentage value specifying how much of the right input to play in the left speaker (0–100).
- rr A percentage value specifying how much of the right input to play in the right speaker (0–100).
- rl A percentage value specifying how much of the left input to play in the right speaker (0–100).

The net result of the parameters is represented by the following formula:

```
leftOutput = left_input * ll + right_input * lr  
rightOutput = right_input * rr + left_input * rl
```

The values for *left_input* or *right_input* are determined by the type (stereo or mono) of sound in your movie.

Stereo sounds divide the sound input evenly between the left and right speakers and have the following transform settings by default:

```
ll = 100  
lr = 0  
rr = 100  
rl = 0
```

Mono sounds play all sound input in the left speaker and have the following transform settings by default:

```
ll = 100  
lr = 100  
rr = 0  
rl = 0
```

Example

The following example illustrates a setting that can be achieved by using the *setTransform* method, but cannot be achieved by using the *setVolume* or *setPan* methods, even if they are combined.

The following code creates a new `soundTransformObject` object and sets its properties so that sound from both channels will play only in the left channel.

```
mySoundTransformObject = new Object;  
mySoundTransformObject.ll = 100;  
mySoundTransformObject.lr = 100;  
mySoundTransformObject.rr = 0;  
mySoundTransformObject.rl = 0;
```

To apply the `soundTransformObject` object to a `Sound` object, you then need to pass the object to the `Sound` object using the `setTransform` method as follows:

```
mySound.setTransform(mySoundTransformObject);
```

The following example plays a stereo sound as mono; the `soundTransformObjectMono` has the following parameters.

```
mySoundTransformObjectMono = new Object;  
mySoundTransformObjectMono.ll = 50;  
mySoundTransformObjectMono.lr = 50;  
mySoundTransformObjectMono.rr = 50;  
mySoundTransformObjectMono.rl = 50;
```

```
mySound.setTransform(soundTransformObjectMono);
```

This example plays the left channel at half capacity and adds the rest of the left channel to the right channel; the `soundTransformObjectHalf` has the following parameters.

```
mySoundTransformObjectHalf = new Object;  
mySoundTransformObjectHalf.ll = 50;  
mySoundTransformObjectHalf.lr = 0;  
mySoundTransformObjectHalf.rr = 100;  
mySoundTransformObjectHalf.rl = 50;
```

```
setTransform(soundTransformObjectHalf);
```

See also

Constructor for the `Object` object

Sound.setVolume

Availability

Flash Player 5.

Usage

```
mySound.setVolume(volume)
```

Parameters

volume A number from 0 to 100 representing a volume level. 100 is full volume and 0 is no volume. The default setting is 100.

Returns

Nothing.

Description

Method; sets the volume for the `Sound` object.

Example

The following example sets volume to 50% and transfers the sound over time from the left speaker to the right speaker:

```
onClipEvent (load) {  
    i = -100;  
    s = new Sound();  
    s.setVolume(50);  
}  
onClipEvent (enterFrame) {  
    if (i <= 100) {  
        S.setPan(i++);  
    }  
}
```

See also

Sound.setPan, Sound.setTransform

Sound.start

Availability

Flash Player 5.

Usage

```
mySound.start([secondOffset, loop])
```

Parameters

secondOffset An optional parameter that lets you start playing the sound at a specific point. For example, if you have a 30-second sound and want the sound to start playing in the middle, specify 15 for the *secondOffset* parameter. The sound is not delayed 15 seconds, but rather starts playing at the 15-second mark.

loop An optional parameter allowing you to specify the number of times the sound should play consecutively.

Returns

Nothing.

Description

Method; starts playing the last attached sound from the beginning if no parameter is specified, or starting at the point in the sound specified by the *secondOffset* parameter.

See also

Sound.stop

Sound.stop

Availability

Flash Player 5.

Usage

```
mySound.stop(["idName"])
```

Parameters

idName An optional parameter specifying a specific sound to stop playing. The *idName* parameter must be enclosed in quotation marks (" ").

Returns

Nothing.

Description

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

See also

`Sound.start`

`_soundbuftime`

Availability

Flash Player 4.

Usage

```
_soundbuftime = integer
```

Parameters

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

Description

Property (global); establishes the number of seconds of streaming sound to prebuffer. The default value is 5 seconds.

Stage (object)

The Stage object is a top-level object that you can access without using a constructor.

Use the methods and properties of this object to access and manipulate information about the boundaries of a Flash movie.

The Stage object is available in the Flash Player 6 and later.

Method summary for the Stage object

Method	Description
<code>Stage.addListener</code>	Adds a listener object to the Stage object.
<code>Stage.removeListener</code>	Removes a listener object from the Stage object.

Property summary for the Stage object

Method	Description
<code>Stage.align</code>	Alignment of the Flash movie in the browser.
<code>Stage.height</code>	Height of the Stage, in pixels.
<code>Stage.width</code>	Width of the Stage, in pixels.
<code>Stage.scaleMode</code>	The current scaling of the Flash movie.

Event handler summary for the Stage object

Method	Description
<code>Stage.onResize</code>	Indicates that the movie was resized.

Stage.addListener

Availability

Flash Player 6.

Usage

```
Stage.addListener(myListener)
```

Parameters

myListener An object that listens for a callback notification from the `onResize` event.

Returns

Nothing.

Description

Method; detects when a Flash movie is resized if `Stage.scaleMode = "noScale"`. The `addListener` method doesn't work with the default movie scaling setting ("`showAll`") or other scaling settings ("`exactFit`", and "`noBorder`").

To use `addListener`, you must first create a *listener object*. A listener object is an object that receives notification from an event when that event is triggered in a movie. Listener objects of the Stage object receive notification from `Stage.onResize`.

Example

This example creates a new listener object called `myListener`. It then uses `myListener` to call `onResize` and define a function that will be called when `onResize` is triggered. Finally, the code adds the `myListener` object to the callback list of the Stage object. Listener objects allow multiple objects to listen for resize notifications.

```
myListener = new Object();  
myListener.onResize = function () { ... }  
Stage.addListener(myListener);
```

Stage.align

Availability

Flash Player 6.

Usage

`Stage.align`

Description

Property; indicates the current alignment of the Flash movie within the Stage.

The following table lists the values for the `align` property. Any value not listed here centers the movie in the Stage area.

Value	Vertical	Horizontal
"T"	top	center
"B"	bottom	center
"L"	center	left
"R"	center	right
"TL"	top	left
"TR"	top	right
"BL"	bottom	left
"BR"	bottom	right

Stage.height

Availability

Flash Player 6.

Usage

`Stage.height`

Description

Property (read-only); indicates the current height, in pixels, of the Flash movie Stage. When the `Stage.noScale` property has a value of `true`, *height* represents the height of the Flash Player. When the `Stage.noScale` value is `false` (movie scales when player window resized), *height* represents the height of the Flash movie.

Stage.onResize

Availability

Flash Player 6.

Usage

```
Stage.onResize() = function() {...}
```

Parameters

None.

Returns

Nothing.

Description

Callback method; indicates that the Flash movie was resized. You can use this event to write a function that lays out the objects on the Stage when a movie is resized.

Stage.removeListener

Availability

Flash Player 6.

Usage

```
Stage.removeListener(myListener)
```

Parameters

myListener An object added to an object's callback list with the `addListener` method.

Returns

Nothing.

Description

Method; removes a listener object created with `addListener`.

See also

`Stage.addListener`

Stage.scaleMode

Availability

Flash Player 6.

Usage

```
Stage.scaleMode = "value"
```

Description

Property; indicates the current scaling of the Flash movie within the Stage. The `scaleMode` property forces the movie into a specific scaling mode. By default, the movie uses the HTML parameters set in the Publish Settings dialog box.

The `scaleMode` property can use the values "exactFit", "showAll", "noBorder", and "noScale". Any other value sets the `scaleMode` property to the default "showAll".

Stage.width

Availability

Flash Player 6.

Usage

Stage.width

Description

Property (read-only); indicates the current width, in pixels, of the Flash movie Stage. When the value of `Stage.noScale` is `true`, the `width` property represents the width of the Player. When the value of `Stage.noScale` is `false` (movie scales when the Player window is resized), `width` represents the width of the Flash movie.

startDrag

Availability

Flash Player 4.

Usage

```
startDrag(target,[lock ,left , top , right , bottom])
```

Parameters

target The target path of the movie clip to drag.

lock A Boolean value specifying whether the draggable movie clip is locked to the center of the mouse position (*true*), or locked to the point where the user first clicked on the movie clip (*false*). This parameter is optional.

left, top, right, bottom Values relative to the coordinates of the movie clip's parent that specify a constraint rectangle for the movie clip. These parameters are optional.

Returns

Nothing.

Description

Action; makes the *target* movie clip draggable while the movie is playing. Only one movie clip can be dragged at a time. Once a `startDrag` operation is executed, the movie clip remains draggable until explicitly stopped by a `stopDrag` action, or until a `startDrag` action for another movie clip is called.

Example

To create a movie clip that users can position in any location, attach the `startDrag` and `stopDrag` actions to a button inside the movie clip.

```
on(press) {  
    startDrag(this,true);  
}  
on(release) {  
    stopDrag();  
}
```

See also

`MovieClip._droptarget`, `stopDrag`

stop

Availability

Flash 2.

Usage

stop

Parameters

None.

Returns

Nothing.

Description

Action; stops the movie that is currently playing. The most common use of this action is to control movie clips with buttons.

stopAllSounds

Availability

Flash Player 3.

Usage

stopAllSounds()

Parameters

None.

Returns

Nothing.

Description

Action; stops all sounds currently playing in a movie without stopping the playhead. Sounds set to stream will resume playing as the playhead moves over the frames they are in.

Example

The following code could be applied to a button that, when clicked, stops all sounds in the movie.

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

See also

Sound (object)

stopDrag

Availability

Flash Player 4.

Usage

stopDrag()

Parameters

None.

Returns

Nothing.

Description

Action; stops the current drag operation.

Example

This statement stops the drag action on the instance `mc` when the user releases the mouse button:

```
on(press) {  
    startDrag("mc");  
}  
on(release) {  
    stopdrag();  
}
```

See also

`MovieClip.stopDrag`, `MovieClip._droptarget`, `startDrag`,

String (function)

Availability

Flash Player 4.

Usage

`String(expression)`

Parameters

expression An expression to convert to a string.

Returns

Nothing.

Description

Function; returns a string representation of the specified parameter as follows:

If *expression* is a Boolean value, the return string is `true` or `false`.

If *expression* is a number, the return string is a text representation of the number.

If *expression* is a string, the return string is *expression*.

If *expression* is an object, the return value is a string representation of the object generated by calling the `string` property for the object, or by calling `Object.toString` if no such property exists.

If *expression* is a movie clip, the return value is the target path of the movie clip in slash (/) notation.

If *expression* is undefined, the return value is an empty string(`"`).

See also

`Number.toString`, `Object.toString`, `String(object)`, `" "` (string delimiter)

" " (string delimiter)

Availability

Flash Player 4.

Usage

`"text"`

Parameters

text A character.

Returns

Nothing.

Description

String delimiter; when used before and after characters, quotes indicate that the characters have a literal value and are considered a *string*—not a variable, numerical value, or other ActionScript element.

Example

This example uses quotes to indicate that the value of the variable *yourGuess* is the literal string “Prince Edward Island” and not the name of a variable. The value of *province* is a variable, not a literal; to determine the value of *province*, the value of *yourGuess* must be located.

```
yourGuess = "Prince Edward Island";

on(release){
    province = yourGuess
    trace(province);
}

// displays Prince Edward Island in the Output window
```

See also

`String (object)`, `String (function)`

String (object)

The String object is a wrapper for the string primitive data type, which allows you to use the methods and properties of the String object to manipulate primitive string value types. You can convert the value of any object into a string using the `String()` function. In Flash MX, the String object has become a native object. As such, you will experience dramatic improvement in performance.

All of the methods of the String object, except for `concat`, `fromCharCode`, `slice`, and `substr`, are generic. This means the methods themselves call `this.toString` before performing their operations, and you can use these methods with other non-String objects.

Since all string indexes are zero-based, the index of the last character for any string *x* is as follows:

```
x.length - 1
```

You can call any of the methods of the String object using the constructor method `new String` or using a string literal value. If you specify a string literal, the ActionScript interpreter automatically converts it to a temporary String object, calls the method, and then discards the temporary String object. You can also use the `String.length` property with a string literal.

It is important that you do not confuse a string literal with an instance of the `String` object. In the following example, the first line of code creates the string literal `s1`, and the second line of code creates an instance of the `String` object `s2`.

```
s1 = "foo"
s2 = new String("foo")
```

Use string literals unless you specifically need to use a `String` object.

Method summary for `String` object

Method	Description
<code>String.charAt</code>	Returns the character at a specific location in a string.
<code>String.charCodeAt</code>	Returns the value of the character at the given index as a 16-bit integer between 0 and 65535.
<code>String.concat</code>	Combines the text of two strings and returns a new string.
<code>String.fromCharCode</code>	Returns a string made up of the characters specified in the parameters.
<code>String.indexOf</code>	Searches the string and returns the index of the substring specified in the parameters. If value occurs more than once, the index of the first occurrence is returned. If value is not found, -1 is returned.
<code>String.lastIndexOf</code>	Returns the index of the last substring within the string that appears before the start position specified in the parameter, or -1 if not found.
<code>String.slice</code>	Extracts a section of a string and returns a new string.
<code>String.split</code>	Splits a <code>String</code> object into an array of strings by separating the string into substrings.
<code>String.substr</code>	Returns a specified number of characters in a string beginning at the location specified in the parameter.
<code>String.substring</code>	Returns the characters between two indexes, specified in the parameters as a string.
<code>String.toLowerCase</code>	Converts the string to lowercase and returns the result; does not change the contents of the original object.
<code>String.toUpperCase</code>	Converts the string to uppercase and returns the result; does not change the contents of the original object.

Property summary for the `String` object

Property	Description
<code>String.length</code>	Returns the length of the string.

Constructor for the `String` object

Availability

Flash Player 5.

Usage

```
new String(value)
```

Parameters

value The initial value of the new `String` object.

Returns

Nothing.

Description

Constructor; creates a new `String` object.

See also

`String (function)`, `" "` (string delimiter)

String.charAt

Availability

Flash Player 5.

Usage

```
myString.charAt(index)
```

Parameters

index The number of the character in the string to be returned.

Returns

Nothing.

Description

Method; returns the character in the position specified by the parameter *index*. The index of the first character in a string is 0. If *index* is not a number from 0 to `string.length - 1`, an empty string is returned.

String.charCodeAtAt

Availability

Flash Player 5.

Usage

```
myString.charCodeAt(index)
```

Parameters

index An integer that specifies the position of a character in the string. The first character is indicated by 0, and the last character is indicated by `myString.length-1`.

Returns

Nothing.

Description

Method; returns a 16-bit integer from 0 to 65535 that represents the character specified by *index*.

This method is similar to `string.charAt` except that the returned value is a 16-bit integer character code, not a character.

Example

In the following example, the `charCodeAt` method is called on the first letter of the string "Chris".

```
s = new String("Chris");  
i = s.charCodeAt(0);  
// i = 67
```

String.concat

Availability

Flash Player 5.

Usage

```
myString.concat(value1,...valueN)
```

Parameters

value1,...*valueN* Zero or more values to be concatenated.

Returns

Nothing.

Description

Method; combines the value of the String object with the parameters and returns the newly formed string; the original value, *myString*, is unchanged.

String.fromCharCode

Availability

Flash Player 5.

Usage

```
String.fromCharCode(c1,c2,...cN)
```

Parameters

c1,*c2*,...*cN* Decimal integers that represent ASCII values.

Returns

Nothing.

Description

Method; returns a string made up of the characters represented by the ASCII values in the parameters.

Example

This example uses the `fromCharCode` method to insert an “@” character in the e-mail address.

```
address = "dog" + String.fromCharCode(64) + "house.net";  
trace(address);  
// output: dog@house.net
```

String.indexOf

Availability

Flash Player 5.

Usage

```
myString.indexOf(substring, [startIndex])
```

Parameters

substring An integer or string specifying the substring to be searched for within *myString*.

startIndex An integer specifying the starting point in *myString* to search for substring. This parameter is optional.

Returns

Nothing.

Description

Method; searches the string and returns the position of the first occurrence of the specified *substring*. If the value is not found, the method returns -1.

String.lastIndexOf

Availability

Flash Player 5.

Usage

```
myString.lastIndexOf(substring, [startIndex])
```

Parameters

substring An integer or string specifying the string to be searched for.

startIndex An integer specifying the starting point to search for *substring*. This parameter is optional.

Returns

Nothing.

Description

Method; searches the string from right to left and returns the index of the last occurrence of *substring* found before *startIndex* within the calling string. If *substring* is not found, the method returns -1.

String.length

Availability

Flash Player 5.

Usage

```
string.length
```

Parameters

None.

Description

Property; returns the number of characters in the specified String object.

String.slice

Availability

Flash Player 5.

Usage

```
myString.slice(start, [end])
```

Parameters

start A number specifying the index of the starting point for the slice. If *start* is a negative number, the starting point is determined from the end of the string, where -1 is the last character.

end A number specifying the index of the ending point for the slice. If *end* is not specified, the slice includes all characters from *start* to the end of the string. If *end* is a negative number, the ending point is determined from the end of the string, where -1 is the last character.

Returns

Nothing.

Description

Method; extracts a slice, or substring, of the specified String object; then returns it as a new string without modifying the original String object. The returned string includes the *start* character and all characters up to (but not including) the *end* character.

Example

The following example sets a variable, *text*, creates an instance of the String object, *s*, and passes it the *text* variable. The *slice* method extracts a section of the string contained in the variable and the *trace* action sends it to the Output window.

```
text = "lexington";  
s = new String( text );  
trace(s.slice( 1, 3 ));  
trace(s);
```

The Output window displays *ex*.

The following code produces the same result, but the parameter passed to the String function is a string instead of a variable.

```
s = new String( "lexington" );  
trace(s.slice( 1, 3 ));  
trace(s);
```

The Output window displays *ex*.

String.split

Availability

Flash Player 5.

Usage

```
myString.split(delimiter, [limit])
```

Parameters

delimiter The character or string at which *myString* splits. If the *delimiter* parameter is undefined, the entire string is placed in the first element of the array.

limit The number of items to place into the array. This parameter is optional.

Returns

An array containing the substrings of *myString*.

Description

Method; splits a String object into substrings by breaking it wherever the specified *delimiter* parameter occurs, and returns the substrings in an array. If you use an empty string ("") as a delimiter, each character in the string is placed as an element in the array, as in the following code.

```
myString = "Joe";  
i = myString.split("");  
trace (i);
```

The Output window displays the following:

J, O, E

If the *delimiter* parameter is undefined, the entire string is placed into the first element of the returned array.

Example

The following example returns an array with five elements.

```
myString = "P, A, T, S, Y";  
myString.split(",");
```

This example returns an array with two elements.

```
myString.split(", ", 2);
```

String.substr

Availability

Flash Player 5.

Usage

```
myString.substr(start, [length])
```

Parameters

start An integer that indicates the position of the first character in *myString* to be used to create the substring. If *start* is a negative number, the starting position is determined from the end of the string, where the -1 is the last character.

length The number of characters in the substring being created. If *length* is not specified, the substring includes all of the characters from the start to the end of the string.

Returns

Nothing.

Description

Method; returns the characters in a string from the index specified in the *start* parameter through the number of characters specified in the *length* parameter. The *substr* method does not change the string specified by *myString*, it returns a new string.

String.substring

Availability

Flash Player 5.

Usage

```
myString.substring(from, to)
```

Parameters

from An integer that indicates the position of the first character of *myString* used to create the substring. Valid values for *from* are 0 through `string.length - 1`. If *from* is a negative value, 0 is used.

to An integer that is 1+ the index of the last character in *myString* to be extracted. Valid values for *to* are 1 through `string.length`. The character indexed by the *to* parameter is not included in the extracted string. If this parameter is omitted, `string.length` is used. If this parameter is a negative value, 0 is used.

Returns

Nothing.

Description

Method; returns a string consisting of the characters between the points specified by the *from* and *to* parameters. If the *to* parameter is not specified, the end of the substring is the end of the string. If the value of *from* equals the value of *to*, the method returns an empty string. If the value of *from* is greater than the value of *to*, the parameters are automatically swapped before the function executes and the original value is unchanged.

String.toLowerCase

Availability

Flash Player 5.

Usage

```
myString.toLowerCase()
```

Parameters

None.

Returns

Nothing.

Description

Method; returns a copy of the String object, with all of the uppercase characters converted to lowercase. The original value is unchanged.

String.toUpperCase

Availability

Flash Player 5.

Usage

```
myString.toUpperCase()
```


Parameters

None.

Returns

Nothing.

Description

Method; returns a copy of the String object, with all of the lowercase characters converted to uppercase. The original value is unchanged.

substring

Availability

Flash Player 4. This function has been deprecated in favor of `String.substr`.

Usage

```
substring("string", index, count)
```

Parameters

string The string from which to extract the new string.

index The number of the first character to extract.

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

Returns

Nothing.

Description

String function; extracts part of a string. This function is 1-based, whereas the String object methods are 0-based.

See also

`String.substr`

super

Availability

Flash Player 6.

Usage

```
super.method([arg1, ..., argN])
```

```
super([arg1, ..., argN])
```

Parameters

method The method to invoke in the superclass.

arg1 Optional parameters that are passed to the superclass version of the method (syntax 1) or to the constructor function of the superclass (syntax 2).

Returns

Both forms invoke a function. The function may return any value.

Description

Operator: the first syntax style may be used within the body of an object method to invoke the superclass version of a method, and can optionally pass parameters (*arg1 ... argN*) to the superclass method. This is useful for creating subclass methods that add additional behavior to superclass methods, but also invoke the superclass methods to perform their original behavior.

The second syntax style may be used within the body of a constructor function to invoke the superclass version of the constructor function and may optionally pass it parameters. This is useful for creating a subclass that performs additional initialization, but also invokes the superclass constructor to perform superclass initialization.

Example

The following example creates two classes, `ParentClass` and `ChildClass`, and defines a method called `method` for each class. Each definition of method has a `trace` action that sends a message to the Output window. The second-to-last line of code creates an instance of the `ChildClass` and calls its method `method`:

```
function ParentClass() {  
}  
ParentClass.prototype.method = function () {  
    trace("ParentClass implementation of method");  
};  
function ChildClass() {  
}  
ChildClass.prototype = new ParentClass();  
ChildClass.prototype.method = function () {  
    trace("ChildClass implementation of method");  
    super.method();  
};  
x = new ChildClass();  
x.method();
```

The following is displayed in the Output window:

```
ChildClass implementation of method  
ParentClass implementation of method
```

The following example passes parameters to the super constructor:

```
function SuperClass(arg){  
    trace("SuperClass constructor was passed " + arg);  
}  
function SubClass(arg){  
    super(arg);  
    trace("SubClass constructor");  
}
```

switch

Availability

Flash Player 4.

Usage

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

Parameters

expression Any expression.

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

defaultClause A `default` keyword followed by statements to execute if none of the case expressions match the switch *expression* parameter strict equality (`===`).

Returns

Nothing.

Description

Action; creates a branching structure for ActionScript statements. Like the `if` action, the `switch` action tests a condition and executes statements if the condition returns a value of `true`.

Example

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

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

In the following example, there isn't a `break` in the first case group so if the number is 1, both A and B are sent to the Output window:

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

See also

`===` (strict equality), `break`, `case`, `default`, `if`

System (object)

This is a top-level object that contains the Capabilities object. You must use the System object to use the Capabilities object and its properties. For example, the following code checks to see if a system has audio capabilities.

```
System.capabilites.hasAudio
```

System.capabilities (object)

You can use the `System.capabilities` object to determine the abilities of the system and player hosting a Flash movie. This allows you to tailor content for different formats. For example, the screen of a cell phone (black and white, 100 square pixels) is different than the 1000-square-pixel color PC screen. To provide appropriate content to as many users as possible, you can use the Capabilities object to determine the type of device a user has. You can then either specify to the server to send different SWFs based on the device capabilities, or tell the Flash movie to alter its presentation based on the capabilities of the device.

You can send capabilities information using a GET or POST HTTP method. The following is an example of a server string for a device that does not have MP3 support and has a 400 x 200 pixel, 8 x 4 centimeter screen:

```
"A=t&MP3=f&AE=gsm&VE=h11&ACC=f&V=WIN%206%2C0%2C0%2C129&M=Macromedia%WINDOWS&R=400x200&DP=72&COL=color&AR=1.0&OS=WINDOWS%2000&L=en-US"
```

The Capabilities object is available in the Flash Player 6.

You must access all properties of the Capabilities object through the `System.capabilities` object.

Property summary for the Capabilities object

Property	Description
<code>System.capabilities.hasAudioEncoder</code>	Indicates the supported audio encoders.
<code>System.capabilities.hasAccessibility</code>	Indicates whether the device meets accessibility standards.
<code>System.capabilities.hasAudio</code>	Indicates whether the device has audio capabilities.
<code>System.capabilities.hasMP3</code>	Indicates whether the device has an MP3 decoder.
<code>System.capabilities.language</code>	Indicates the language the Flash Player supports.
<code>System.capabilities.manufacturer</code>	Indicates the manufacturer of the Flash Player.
<code>System.capabilities.os</code>	Indicates the operating system hosting the Flash Player.
<code>System.capabilities.pixelAspectRatio</code>	Indicates the pixel aspect ration of the screen.
<code>System.capabilities.screenColor</code>	Indicates whether the screen is color, black and white, or grayscale.
<code>System.capabilities.screenDPI</code>	Indicates the screen dots per inch.
<code>System.capabilities.screenResolution.x</code>	Indicates the horizontal size of the screen.
<code>System.capabilities.screenResolution.y</code>	Indicates the vertical size of the screen.
<code>System.capabilities.version</code>	Indicates the lowest supported Flash Player version.
<code>System.capabilities.hasVideoEncoder</code>	Indicates the supported video encoders.

System.capabilities.hasAudioEncoder

Availability

Flash Player 6.

Usage

`System.capabilities.hasAudioEncoder`

Description

Property; an array of audio decoders. The server string is `AE`.

System.capabilities.hasAccessibility

Availability

Flash Player 6.

Usage

`System.capabilities.hasAccessibility`

Description

Property; a Boolean value that indicates whether or not the device supports communication between the Flash Player and accessibility aids. The default value is `false`. The server string is `ACC`.

System.capabilities.hasAudio

Availability

Flash Player 6.

Usage

`System.capabilities.hasAudio`

Description

Property; a Boolean value that indicates whether or not the player has audio capabilities. The default value is `true`. The server string is `A`.

System.capabilities.hasMP3

Availability

Flash Player 6.

Usage

`System.capabilities.hasMP3`

Description

Property; a Boolean value that indicates whether or not the player has an MP3 decoder. The default value is `true`. The server string is `MP3`.

System.capabilities.language

Availability

Flash Player 6.

Usage

System.capabilities.language

Description

Property; a lowercase two-letter language code from ISO 639-1, and an optional uppercase two-letter country code subtag from ISO 3166. The languages themselves are named with the English tags. For example, “en-US” is the language of the document you are currently reading. The server string is LAN. Flash supports the following subset of the language tags:

Language	Tag	Supported Countries and Tags
English	en	United States = US, United Kingdom = UK
French	fr	
Korean	ko	
Japanese	ja	
Swedish	sv	
German	de	
Spanish	es	
Italian	it	
Simplified Chinese	zh	PRC (Simplified Chinese) = CN
Traditional Chinese	zh	Taiwan (Traditional Chinese) = TW
Portuguese	pt	
Polish	pl	
Hungarian	hu	
Czech	cs	
Turkish	tr	
Finnish	fi	
Danish	da	
Norwegian	no	
Dutch	nl	
Russian	ru	
Other/Unknown	xu	

System.capabilities.manufacturer

Availability

Flash Player 6.

Usage

`System.capabilities.manufacturer`

Description

Property; a string that indicates the manufacturer of the Flash Player. The default is "Macromedia OSName" (*OSName* could be "Windows", "Macintosh", or "Other OS Name"). The server string is M.

System.capabilities.os

Availability

Flash Player 6.

Usage

`System.capabilities.os`

Description

Property; a string that indicates the manufacturer of the Flash Player. The default is an empty string (""). The `os` property can return the following strings: "Windows XP", "Windows 2000", "Windows NT", "Windows 98/ME", "Windows 95", "Windows CE" (Only available only in SDK, not in the desktop version), and "MacOS". The server string is OS.

System.capabilities.pixelAspectRatio

Availability

Flash Player 6.

Usage

`System.capabilities.hasVideoEncoder`

Description

Property; an integer that indicates the pixel aspect ratio of the screen. The default value is 1.0. The server string is PAR.

System.capabilities.screenColor

Availability

Flash Player 6.

Usage

`System.capabilities.screenColor`

Description

Property; indicates the color of the screen, color (color), gray (gray) or black and white (bw). The default value is color. The server string is SC.

System.capabilities.screenDPI

Availability

Flash Player 6.

Usage

`System.capabilities.screenDPI`

Description

Property; indicates the dots per inch (dpi) of the screen, in pixels. The default value is 72. The server string is DPI.

System.capabilities.screenResolution.x

Availability

Flash Player 6.

Usage

`System.capabilities.screenResolution.x`

Description

Property; an integer that indicates the maximum horizontal resolution of the screen. The default value is 800 (pixels). The server string is SRX.

System.capabilities.screenResolution.y

Availability

Flash Player 6.

Usage

`System.capabilities.screenResolution.y`

Description

Property; an integer that indicates the maximum vertical resolution of the screen. The default value is 600 (pixels). The server string is SRY.

System.capabilities.version

Availability

Flash Player 6.

Usage

`System.capabilities.version`

Description

Property; an integer that specifies the supported Flash Player version. The default is 6.0. The server string is VER.

System.capabilities.hasVideoEncoder

Availability

Flash Player 6.

Usage

`System.capabilities.hasVideoEncoder`

Description

Property; an array of video encoders. The server string is VE.

targetPath

Availability

Flash Player 5.

Usage

`targetPath(movieClipObject)`

Parameters

movieClipObject Reference (for example, `_root` or `_parent`) to the movie clip for which the target path is being retrieved.

Returns

Nothing.

Description

Function; returns a string containing the target path of *movieClipObject*. The target path is returned in dot notation. To retrieve the target path in slash notation, use the `_target` property.

Example

This example displays the target path of a movie clip as soon as it loads.

```
onClipEvent(load){
    trace(targetPath(this));
}
```

See also

`eval`

tellTarget

Availability

Flash Player 3. (Deprecated in Flash 5; use of dot notation and the `with` action is recommended.)

Usage

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

Parameters

target A string that specifies the target path of the Timeline to be controlled.

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

Returns

Nothing.

Description

Action; applies the instructions specified in the *statements* parameter to the Timeline specified in the *target* parameter. The `tellTarget` action is useful for navigation controls. Assign `tellTarget` to buttons that stop or start movie clips elsewhere on the Stage. You can also make movie clips go to a particular frame in that clip. For example, you might assign `tellTarget` to buttons that stop or start movie clips on the Stage or prompt movie clips to jump to a particular frame.

In Flash 5, you can use dot notation instead of the `tellTarget` action. You can use the `with` action to issue multiple actions to the same Timeline. You can use the `with` action to target any object, whereas the `tellTarget` action can only target movie clips.

Example

This `tellTarget` statement controls the movie clip instance `ball` on the main Timeline. Frame 1 of the `ball` instance is blank and has a `stop` action so that it isn't visible on the Stage. When the button with the following action is clicked, `tellTarget` tells the playhead in `ball` to go to frame 2 where the animation starts.

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

The following example uses dot notation to achieve the same results.

```
on(release) {  
    ball.gotoAndPlay(2);  
}
```

If you need to issue multiple commands to the `ball` instance, you can use the `with` action, as in the following statement.

```
on(release) {  
    with(ball) {  
        gotoAndPlay(2);  
        _alpha = 15;  
        _xscale = 50;  
        _yscale = 50;  
    }  
}
```

See also

`with`

TextField (object)

All dynamic and input text fields in a Flash movie are instances of the `TextField` object. You can give a text field an instance name in the Property inspector and use the methods and properties of the `TextField` object to manipulate it with ActionScript. `TextField` instance names are displayed in the Movie Explorer and in the Insert Target Path dialog box in the Actions panel.

The `TextField` object inherits from the `Object` object.

To create a text field dynamically, you can use the `MovieClip.createTextField` method.

The `TextField` object is supported by Flash Player 6 and later versions of the Flash Player.

Method summary for the TextField object

Method	Description
<code>TextField.addListener</code>	Registers an object to receive notification when the <code>onChanged</code> and <code>onScroller</code> events are invoked.
<code>TextField.getDepth</code>	Returns the depth of a text field.
<code>TextField.getNewTextFormat</code>	Gets the default text format assigned to newly inserted text.
<code>TextField.removeListener</code>	Removes a listener object.
<code>TextField.removeTextField</code>	Removes a text field that was created with <code>MovieClip.createTextField</code> .
<code>TextField.setNewTextFormat</code>	Sets a text format object for text that is inserted by a user or by a method.
<code>TextField.replaceSel</code>	Replaces the current selection.
<code>TextField.setTextFormat</code>	Sets the default text format assigned to newly inserted text.

Property summary for the TextField object

Property	Description
<code>TextField._alpha</code>	The transparency value of a text field instance.
<code>TextField.autoSize</code>	Controls automatic alignment and sizing of a text field.
<code>TextField.background</code>	Indicates if the text field has a background fill.
<code>TextField.backgroundColor</code>	Indicates the color of the background fill.
<code>TextField.border</code>	Indicates if the text field has a border.
<code>TextField.borderColor</code>	Indicates the color of the border.
<code>TextField.bottomScroll</code>	The bottommost visible line in a text field.
<code>TextField.embedFonts</code>	Indicates whether the text field uses embedded font outlines or device fonts.
<code>TextField._highQuality</code>	Indicates the rendering quality of the movie.
<code>TextField._height</code>	The height of a text field instance in pixels. This only affects the bounding box of the text field, it does not affect the border thickness or text font size.
<code>TextField.hscroll</code>	Indicates the horizontal scroll value of a text field.
<code>TextField.html</code>	Indicates the current maximum scrolling position of a text field.
<code>TextField.htmlText</code>	Contains the HTML representation of a text field's contents.
<code>TextField.length</code>	The number of characters in a text field.
<code>TextField.maxChars</code>	The maximum number of characters that a text field can contain.
<code>TextField.maxhscroll</code>	The maximum value of <code>TextField.hscroll</code> .
<code>TextField.maxscroll</code>	The maximum value of <code>TextField.scroll</code> .
<code>TextField.multiline</code>	Indicates if the text field contains multiple lines.
<code>TextField._name</code>	The instance name of a text field instance.
<code>TextField._parent</code>	A reference to the instance that is the parent of this instance; either of type <code>Button</code> or <code>MovieClip</code> .
<code>TextField.password</code>	Indicates if a text field hides the input characters.
<code>TextField._quality</code>	Indicates the rendering quality of a movie.

Property	Description
<code>TextField.restrict</code>	The set of characters that a user can enter into a text field.
<code>TextField._rotation</code>	The degree of rotation of a text field instance.
<code>TextField.scroll</code>	Indicates the current scrolling position of a text field.
<code>TextField.selectable</code>	Indicates whether a text field is selectable.
<code>TextField._soundbuftime</code>	The amount of time a sound must prebuffer before it streams.
<code>TextField.tabEnabled</code>	Indicates whether a movie clip is included in automatic tab ordering.
<code>TextField.tabIndex</code>	Indicates the tab order of an object.
<code>TextField.text</code>	The current text in the text field.
<code>TextField.textColor</code>	The color of the current text in the text field.
<code>TextField.textHeight</code>	The height of the text field's bounding box.
<code>TextField.textWidth</code>	The width of the text field's bounding box.
<code>TextField.type</code>	Indicates whether a text field is an input text field or dynamic text field.
<code>TextField._url</code>	The URL of the SWF file that created the text field instance.
<code>TextField.variable</code>	The variable name associated with the text field.
<code>TextField._visible</code>	A Boolean value that determines whether a text field instance is hidden or visible.
<code>TextField._width</code>	The width of a text field instance in pixels. This only affects the bounding box of the text field, it does not affect the border thickness or text font size.
<code>TextField.wordWrap</code>	Indicates whether the text field word-wraps.
<code>TextField._x</code>	The <i>x</i> coordinate of a text field instance
<code>TextField._xmouse</code>	The <i>x</i> coordinate of the cursor relative to a text field instance.
<code>TextField._xscale</code>	The value specifying the percentage for horizontally scaling a text field instance.
<code>TextField._y</code>	The <i>y</i> coordinate of a text field instance.
<code>TextField._ymouse</code>	The <i>y</i> coordinate of the cursor relative to a text field instance.
<code>TextField._yscale</code>	The value specifying the percentage for vertically scaling a text field instance.

Event handler summary for the TextField object

Method	Description
<code>TextField.onChangeed</code>	Invoked when the text field is changed.
<code>TextField.onKillFocus</code>	Invoked when the text field loses focus.
<code>TextField.onScroller</code>	Invoked when the <code>scroll</code> , <code>maxscroll</code> , <code>hscroll</code> , <code>maxhscroll</code> , or <code>bottomscroll</code> property of a text field changes.
<code>TextField.onSetFocus</code>	Invoked when the text field receives focus.

Listener summary for the TextField object

Method	Description
<code>TextField.onChangeed</code>	Notified when the text field is changed.
<code>TextField.onScroller</code>	Notified when the <code>scroll</code> or <code>maxscroll</code> property of a text field changes.

TextField._alpha

Availability

Flash Player 6.

Usage

TextField._alpha

Description

Property; sets or retrieves the alpha transparency (*value*) of the text field specified by *TextField*. Valid values are 0 (fully transparent) to 100 (fully opaque).

Example

The following statements set the `_alpha` property of a text field named `text1` to 30%.

```
on(release) {  
    text1._alpha = 30;  
}
```

TextField.addListener

Availability

Flash Player 6.

Usage

TextField.addListener(newListener)

Parameters

newListener An object with the events `onChangeed` and `onScroller` notifications.

Returns

Nothing.

Description

Method; registers an object to receive event notifications. When the `onChanged` or `onScroller` event occurs, the `TextField.onChange` and `TextField.onScroller` events are invoked, followed by the `onChange` and `onScroller` methods of listening objects registered with `addListener`. Multiple objects may listen for change notifications. If the listener *newListener* is already registered, no change occurs.

TextField.autoSize

Availability

Flash Player 6.

Usage

TextField.autoSize

Description

Property; controls automatic sizing and alignment of text fields. If the value of `autosize` is "none", the text field behaves normally and does not automatically resize or align to match the text. If the value is "left", the text field expands or contracts its right and bottom sides to fit all contained text. The left and top sides remain at the same positions. If the value of `autosize` is "center", the text field auto-sizes, but the horizontal center of the text field stays anchored at the text field's original horizontal center position. The bottom side still expands to fit all contained text. If the value of `autosize` is "right", the text field auto-sizes; but the left and bottom sides expand or contract. The top and right side remain in the same positions. When setting the `autoSize` property, `true` is a synonym for "left" and `false` is a synonym for "none".

Example

The following sets the `autosize` property of the text field `textField2` to "center".

```
textField2.autosize = "center";
```

TextField.background

Availability

Flash Player 6.

Usage

TextField.background

Description

Property; if `true`, the text field has a background fill. If `false`, the text field has no background fill.

TextField.backgroundColor

Availability

Flash Player 6.

Usage

TextField.backgroundColor

Description

Property; the color of the text field background. Default is `0xFFFFFFFF` (white). This property may be retrieved or set, even if there currently is no background but the color is only visible if the text field has a border.

See also

`TextField.background`

TextField.border

Availability

Flash Player 6.

Usage

`TextField.border`

Description

Property; if `true`, the text field has a border. If `false`, the text field has no border.

TextField.borderColor

Availability

Flash Player 6.

Usage

`TextField.borderColor`

Description

Property; the color of the text field border, the Default is `0x000000` (black). This property may be retrieved or set, even if there is currently no border.

See also

`TextField.border`

TextField.bottomScroll

Availability

Flash Player 6.

Usage

`TextField.bottomScroll`

Description

Property (read-only); an integer (1-based index) that indicates the bottommost line that is currently visible in *TextField*. Think of the text field as “a window” onto a block of text. The property `TextField.scroll` is the 1-based index of the topmost visible line in the window.

All the text between lines `TextField.scroll` and `TextField.bottomScroll` is currently visible in the text field.

TextField.embedFonts

Availability

Flash Player 6.

Usage

TextField.embedFonts

Description

Property; a Boolean value that, when `true`, renders the text field using embedded font outlines. If `false`, it renders text field using device fonts.

TextField._focusrect

Availability

Flash Player 6.

Usage

TextField._focusrect

Description

Property; a Boolean value that specifies whether a text field has a yellow rectangle around it when it has focus.

TextField.getDepth

Availability

Flash Player 6.

Usage

TextField.getDepth

Parameters

None.

Returns

An integer.

Description

Method; returns the depth of a text field.

TextField.getFontList

Availability

Flash Player 6.

Usage

`TextField.getFontList`

Parameters

None.

Returns

An array.

Description

Method; Returns an Array object whose elements are the names of all fonts on the Flash Player host system, including fonts in the SWF file and any loaded asset SWF files. The names are of type string.

TextField.getNewTextFormat

Availability

Flash Player 6.

Usage

`TextField.getNewTextFormat()`

Parameters

None.

Returns

A TextFormat object.

Description

Method; returns a TextFormat object containing a copy of the text field's text format object. The text format object is the format that newly inserted text, such as text inserted with the `replaceSel` method or text entered by a user, receives. When `getNewTextFormat` is invoked, the TextFormat object returned has all of its properties defined. No property is null.

TextField.getTextFormat

Availability

Flash Player 6.

Usage

`TextField.getTextFormat()`
`TextField.getTextFormat (index)`
`TextField.getTextFormat (beginIndex, endIndex)`

Parameters

index An integer that specifies a character in a string.

Returns

An object.

Description

Method; (Usage 1) returns a `TextFormat` object containing formatting information for all text in a text field. Only properties that are common to all text in the text field are set in the resulting `TextFormat` object. Any property which is *mixed*, meaning that it has different values at different points in the text, has its value set to `null`.

Usage 2: Returns a `TextFormat` object containing a copy of the text field's text format at *index*.

Usage 3: Returns a `TextFormat` object containing formatting information for the span of text from *beginIndex* to *endIndex*.

See also

`TextField.getNewTextFormat`, `TextField.setNewTextFormat`, `TextField.setTextFormat`

TextField._height

Availability

Flash Player 6.

Usage

`TextField._height`

Description

Property; sets and retrieves the height of the text field, in pixels.

Example

The following code example sets the height and width of a text field.

```
myTextField._width = 200;  
myTextField._height = 200;
```

TextField._highquality

Availability

Flash Player 6.

Usage

`TextField._highquality`

Description

Property (global); specifies the level of anti-aliasing applied to the current movie. Specify 2 (BEST) to apply high quality with bitmap smoothing always on. Specify 1 (high quality) to apply anti-aliasing; this will smooth bitmaps if the movie does not contain animation. Specify 0 (low quality) to prevent anti-aliasing.

Example

```
_highquality = 1;
```

See also

`_quality`, `toggleHighQuality`

TextField.hscroll

Availability

Flash Player 6.

Usage

`TextField.hscroll`

Returns

An integer.

Description

Property; indicates the current horizontal scrolling position. If the `hscroll` property is 0, the text is not horizontally scrolled.

Example

The following example scrolls the text horizontally.

```
on (release) {  
    myTextField.hscroll += 1;  
}
```

See also

`TextField.maxhscroll`, `TextField.scroll`

TextField.html

Availability

Flash Player 6.

Usage

`TextField.html`

Description

Property; a flag that indicates whether the text field contains an HTML representation. If the `html` property is `true`, the text field is an HTML text field. If `html` is `false`, the text field is a non-HTML text field.

See also

`TextField.htmlText`

TextField.htmlText

Availability

Flash Player 6.

Usage

TextField.htmlText

Description

Property; if the text field is an HTML text field, this property contains the HTML representation of the text field's contents. If the text field is not an HTML text field, it behaves identically to the *text* property. You can indicate that a text field is an HTML text field in the Property inspector, or by setting the text field's *html* property to *true*.

Example

In the following example, the text in the text field *text2* is rendered bold.

```
text2.html = true;  
text2.htmlText = "<b> this is bold text </b>";
```

TextField.length

Availability

Flash Player 6.

Usage

TextField.length

Description

Property (read-only); indicates the number of characters in a text field. This property returns the same value as *text.length*, but is faster. A character such as tab ("t") counts as one character.

TextField.maxChars

Availability

Flash Player 6.

Usage

TextField.maxChars

Description

Property; indicates the maximum number of characters that the text field can contain. A script may insert more text than *maxChars* allows; the *maxChars* property only indicates 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.

TextField.maxhscroll

Availability

Flash Player 6.

Usage

TextField.maxhscroll

Description

Property (read-only); indicates the maximum value of *TextField.hscroll*.

See also

TextField.hscroll

TextField.maxscroll

Availability

Flash Player 6.

Usage

TextField.maxscroll

Description

Property (read-only); indicates the maximum value of *TextField.scroll*.

See also

TextField.scroll

TextField.multiline

Availability

Flash Player 6.

Usage

TextField.multiline

Description

Property; indicates whether the text field is a multiline text field. If the value is *true*, the text field is multiline; if the value is *false*, the text field is a single-line text field.

TextField._name

Availability

Flash Player 6.

Usage

TextField._name

Description

Property; returns the instance name of the text field specified by *TextField*.

TextField.onChangeed

Availability

Flash Player 6.

Usage

TextField.onChangeed

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when the content of a text field changes. By default, it is undefined; you can define it in a script.

TextField.onKillFocus

Availability

Flash Player 6.

Usage

```
TextField.onKillFocus = function (newFocus) {  
    statements;  
};
```

Parameters

newFocus The object that is receiving the focus.

Returns

Nothing.

Description

Event handler; an event that is invoked when a text field loses keyboard focus. The `onKillFocus` method receives one parameter, *newFocus*, which is an object representing the new object receiving the focus. If no object receives the focus, *newFocus* contains the value `null`.

TextField.onScroller

Availability

Flash Player 6.

Usage

TextField.onScroller

Description

Event handler; an event that is invoked when one of the text field scroll properties changes.

See also

`TextField.hscroll`, `TextField.maxhscroll`, `TextField.maxscroll`, `TextField.scroll`

TextField.onSetFocus

Availability

Flash Player 6.

Usage

```
TextField.onSetFocus = function(oldFocus){  
statements;  
};
```

Parameters

oldFocus The object to lose focus.

Returns

Nothing.

Description

Event handler; invoked when a text field receives keyboard focus. The *oldFocus* parameter is the object that loses the focus. For example, if the user presses the Tab key to move the input focus from a button to a text field, *oldFocus* contains the text field instance.

If there is no previously focused object, *oldFocus* contains a `null` value.

TextField._parent

Availability

Flash Player 6.

Usage

```
_parent.property  
_parent._parent.property
```

Description

Property; specifies or returns a reference to the movie clip or object that contains the current movie clip or object. The current object is the object containing the ActionScript code that references `_parent`. Use `_parent` to specify a relative path to movie clips or objects that are above the current movie clip or object.

See also

`_root`, `targetPath`

TextField.password

Availability

Flash Player 6.

Usage

```
TextField.password
```

Description

Property; If the value of `password` is `true`, the text field is a password text field and hides the input characters. If `false`, the text field is not a password text field.

TextField._quality

Availability

Flash Player 6.

Usage

TextField._quality

Description

Property (global); sets or retrieves the rendering quality used for a movie. Device fonts are always aliased and therefore are unaffected by the `_quality` property.

The `_quality` property can be set to the following values:

- "LOW" Low rendering quality. Graphics are not anti-aliased; bitmaps are not smoothed.
- "MEDIUM" Medium rendering quality. Graphics are anti-aliased using a 2 x 2 grid, in pixels, 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 grid, in pixels, 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 grid, in pixels, and bitmaps are always smoothed.

Example

The following example sets the rendering quality to LOW:

```
textfield._quality = "LOW";
```

See also

`_highquality`, `toggleHighQuality`

TextField.removeListener

Availability

Flash Player 6.

Usage

`Selection.removeListener(listener)`

Parameters

listener The object that will no longer receive focus notifications.

Returns

If the *listener* was successfully removed, the method returns a `true` value. If the *listener* was not successfully removed (for example if the *listener* was not on the TextField object's listener list), the method returns a value of `false`.

Description

Method; removes a listener object previously registered to a text field instance with `addListener`.

TextField.removeTextField

Availability

Flash Player 6.

Usage

```
TextField.removeTextField()
```

Description

Method; removes the text field specified by *TextField*. This operation can only be performed on a text field that was created with the `createTextField` method of the `MovieClip` object. It will not work on text fields placed by the Timeline. When you call this method, the text field is commanded to remove itself. This is similar to the `MovieClip.removeMovieClip` method.

See also

`MovieClip.createTextField`

TextField.replaceSel

Availability

Flash Player 6.

Usage

```
TextField.replaceSel(text)
```

Parameters

text A string.

Returns

Nothing.

Description

Method; replaces the current selection with the contents of the *text* parameter. The text is inserted at the position of the current selection, using the current default character format and default paragraph format. The text is not treated as HTML, even if the text field is an HTML text field.

You can use the `replaceSel` method to insert and delete text without disrupting the character and paragraph formatting of the rest of the text.

TextField.restrict

Availability

Flash Player 6.

Usage

TextField.restrict

Description

Property; indicates the set of characters that a user may enter into the text field. If the value of the `restrict` property is `null`, you can enter any character. If the value of the `restrict` property is an empty string, you can't enter any character. If the value of the `restrict` property is a string of characters, you can enter only characters in the string into the text field. The string is scanned from left to right. A range may be specified using the dash (-). This only restricts user interaction; a script may put any text into the text field. This property does not synchronize with the Embed Font Outlines check boxes in the Property inspector.

If the string begins with ^, all characters are initially accepted and succeeding characters in the string are excluded from the set of accepted characters. If the string does not begin with ^, no characters are initially accepted and succeeding characters in the string are included in the set of accepted characters.

Example

The following example allows only uppercase characters, spaces, and numbers to be entered into a text field:

```
my_txt.restrict = "A-Z 0-9";
```

The following example includes all characters, but excludes lowercase letters:

```
my_txt.restrict = "^a-z";
```

You can use a backslash to enter a ^ or - verbatim. The accepted backslash sequences are \-, \^ or \\. The backslash must be an actual character in the string, so when specified in ActionScript, a double backslash must be used. For example, the following code includes only the dash (-) and caret (^):

```
my_txt.restrict = "\\-\\^";
```

The ^ may be used anywhere in the string to toggle between including characters and excluding characters. The following code includes only uppercase letters, but excludes the uppercase letter Q:

```
my_txt.restrict = "A-Z^Q";
```

You can use the \u escape sequence to construct `restrict` strings. The following code includes only the characters from ASCII 32 (space) to ASCII 126 (tilde).

```
my_txt.restrict = "\u0020-\u007E";
```

TextField._rotation

Availability

Flash Player 6.

Usage

TextField._rotation

Description

Property; specifies the rotation of the text field in degrees.

TextField.scroll

Availability

Flash Player 6.

Usage

TextField.scroll

Description

Property; defines the vertical position of text in a text field. The `scroll` property is useful for directing users to a specific paragraph in a long passage, or creating scrolling text fields. This property can be retrieved and modified.

Example

The following code is attached to an Up button that scrolls the `myText` text field.

```
on (release) {  
    myText.scroll = myText.scroll + 1;  
}
```

See also

`TextField.maxscroll`, `TextField.scroll`

TextField.selectable

Availability

Flash Player 6.

Usage

TextField.selectable

Description

Property; a Boolean value that indicates whether the text field is selectable. The value `true` indicates that the text is selectable.

TextField.setNewTextFormat

Availability

Flash Player 6.

Usage

```
TextField.setNewTextFormat(textFormat)
```

Parameters

textFormat An instance of the TextFormat object.

Returns

Nothing.

Description

Method; sets a TextFormat object for newly inserted text, such as text inserted with the `replaceSel` method or text entered by a user in a text field. Each text field has a new text format. When text is inserted, the new text is assigned the new text format.

The text format is set in a new instance of the TextFormat object. It contains both character and paragraph formatting information. Character formatting information describes the appearance of individual characters; for example, font name, point size, color, and associated URL. Paragraph formatting information describes the appearance of a paragraph; for example, left margin, right margin, indentation of the first line, and left, right, and center alignment.

See also

`TextField.getNewTextFormat`, `TextField.getTextFormat`, `TextField.setTextFormat`

TextField.setTextFormat

Availability

Flash Player 6.

Usage

```
TextField.setTextFormat (textFormat)
TextField.setTextFormat (index, textFormat)
TextField.setTextFormat (beginIndex, endIndex, textFormat)
```

Parameters

beginIndex An integer.

endIndex An integer that specifies the first character after the desired text span.

textFormat An instance of the TextFormat object. A TextFormat object contains character and paragraph formatting information.

Returns

Nothing.

Description

Method; sets a text format object for a specified range of text in a text field. You can assign each character in a text field a text format. The text format of the first character of a paragraph is examined to perform paragraph formatting for the entire paragraph. The `setTextFormat` method changes the text format applied to individual characters, to groups of characters, or to the entire body of text in a text field.

The text format is set in a new instance of the `TextFormat` object. It contains both character and paragraph formatting information. Character formatting information describes the appearance of individual characters, for example, font name, point size, color, and associated URL. Paragraph formatting information describes the appearance of a paragraph, for example, left margin, right margin, indentation of the first line, and left, right, and center alignment.

Usage 1: Applies the properties of *textFormat* to all text in the text field.

Usage 2: Applies the properties of *textFormat* to the character at position *index*.

Usage 3: Applies the properties of the *textFormat* parameter to the span of text from the *beginIndex* parameter to the *endIndex* parameter.

Example

This example creates a new `TextFormat` object called `myTextFormat` and sets its `bold` property to `true`. It then calls the `setTextFormat` method and applies the new text format to the `myTextField` text field.

```
myTextFormat = new TextFormat();
myTextFormat.bold = true;
myTextField.setTextFormat(myTextFormat);
```

See also

`TextFormat` (object)

TextField._soundbuftime

Availability

Flash Player 6.

Usage

`TextField._soundbuftime`

Description

Property (global); an integer that specifies the number of seconds a sound prebuffers before it starts to stream.

TextField.tabEnabled

Availability

Flash Player 6.

Usage

`TextField.tabEnabled`

Description

Property; may be set on an instance of the `MovieClip`, `Button`, or `TextField` objects. It is undefined by default.

If the `tabEnabled` property is undefined or has a value of `true`, then the object is included in automatic tab ordering, and the object is included in custom tab ordering if the `tabIndex` property is also set to a value. If `tabEnabled` is `false`, then the object is not included in automatic tab ordering. For a movie clip, if `tabEnabled` is `false`, the movie clip's children may still be included in automatic tab ordering, unless the `tabChildren` property is also set to `false`.

If `tabEnabled` is undefined or `true`, then the object is included in custom tab ordering if the `tabIndex` property is set. If `tabEnabled` is `false`, then the object is not included in custom tab ordering, even if the `tabIndex` property is set. If `tabEnabled` is set to `false` in a movie clip, the movie clip's children can still be included in custom tab ordering.

TextField.tabIndex

Availability

Flash Player 6.

Usage

TextField.tabIndex

Parameters

None.

Returns

Nothing.

Description

Property; lets you customize the tab ordering of objects in a movie. You can set the `tabIndex` property on a button, movie clip, or text field instance; it is undefined by default.

If any currently displayed object in the Flash movie contains a `tabIndex` property, automatic tab ordering is disabled, and the tab ordering is calculated from the `tabIndex` properties of objects in the movie. The custom tab ordering only includes objects that have `tabIndex` properties.

The `tabIndex` property must be a positive integer. The objects are ordered according to their `tabIndex` properties, in ascending order. An object with a `tabIndex` of 1 precedes an object with `tabIndex` 2. If two objects have the same `tabIndex` value, the one that precedes the other in the tab ordering is undefined.

The custom tab ordering defined by the `tabIndex` property is flat. This means that no attention is paid to the hierarchical relationships of objects in the Flash movie. All objects in the Flash movie with `tabIndex` properties are placed in the tab order, and the tab order is determined by the order of the `tabIndex` values. If two objects have the same `tabIndex` value, the one that goes first is undefined. You shouldn't use the same `tabIndex` value for multiple objects.

TextField._target

Availability

Flash Player 6.

Usage

TextField._target

Description

Property (read-only); returns the target path of the text field instance specified in the *TextField* parameter.

TextField.text

Availability

Flash Player 6.

Usage

`TextField.text`

Description

Property; indicates the current text in the text field. Lines are separated by the carriage return character ('\r', ASCII 13). This property contains the normal, unformatted text in the text field, without HTML tags, even if the text field is HTML.

See also

`TextField.htmlText`

TextField.textColor

Availability

Flash Player 6.

Usage

`TextField.textColor`

Description

Property; indicates the color of the text in a text field.

TextField.textHeight

Availability

Flash Player 6.

Usage

`TextField.textHeight`

Description

Property; indicates the height of the text.

TextField.textWidth

Availability

Flash Player 6.

Usage

`TextField.textWidth`

Description

Property; indicates the width of the text.

TextField.type

Availability

Flash Player 6.

Usage

TextField.type

Description

Property; Specifies the type of text field. There are two values: "dynamic", which specifies a dynamic text field (cannot be edited by the user) and "input", which specifies an input text field.

Example

```
TextField.type = "dynamic";
```

TextField._url

Availability

Flash Player 6.

Usage

TextField._url

Description

Property (read only); retrieves the URL of the SWF file that created the text field.

TextField.variable

Availability

Flash Player 6.

Usage

TextField._variable

Description

Property; The name of the variable that the text field is associated with. The type of this property is String.

TextField._visible

Availability

Flash Player 6.

Usage

TextField._visible

Description

Property; a Boolean value that indicates whether the text field specified by the *TextField* parameter is visible. Text fields that are not visible (*_visible* property set to *false*) are disabled.

TextField._width

Availability

Flash Player 6.

Usage

TextField._width

Description

Property; sets and retrieves the width of the text field, in pixels.

Example

The following example sets the height and width properties of a text field:

```
myTextField._width=200;  
myTextField._height=200;
```

See also

MovieClip._height

TextField.wordWrap

Availability

Flash Player 6.

Usage

TextField.wordWrap

Description

Property; a Boolean value that indicates if the text field has word wrap. If the value of `wordWrap` is `true`, the text field has word wrap; if the value is `false`, the text field does not have word wrap.

TextField._x

Availability

Flash Player 6.

Usage

TextField._x

Description

Property; an integer that sets the *x* coordinate of a text field relative to the local coordinates of the parent movie clip. If a text field is on the main Timeline, then its coordinate system refers to the upper left corner of the Stage as (0, 0). If the text field is inside a movie clip that has transformations, the text field is in the local coordinate system of the enclosing movie clip. Thus, for a movie clip rotated 90° counterclockwise, the enclosed text field inherits a coordinate system that is rotated 90° counterclockwise. The text field's coordinates refer to the registration point position.

See also

TextField._xscale, *TextField._y*, *TextField._yscale*

TextField._xmouse

Availability

Flash Player 6.

Usage

TextField._xmouse

Description

Property (read-only); returns the *x* coordinate of the mouse position relative to the text field.

See also

TextField._ymouse

TextField._xscale

Availability

Flash Player 6.

Usage

TextField._xscale

Description

Property; determines the horizontal scale (*percentage*) of the text field as applied from the registration point of the text field. The default registration point is (0,0).

See also

TextField._x, TextField._y, TextField._yscale

TextField._y

Availability

Flash Player 6.

Usage

TextField._y

Description

Property; 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 90° counterclockwise, the enclosed text field inherits a coordinate system that is rotated 90° counterclockwise. The text field's coordinates refer to the registration point position.

See also

TextField._x, TextField._xscale, TextField._yscale

TextField._ymouse

Availability

Flash Player 6.

Usage

TextField._ymouse

Description

Property (read-only); indicates the *y* coordinate of the mouse position relative to the text field.

See also

TextField._xmouse

TextField._yscale

Availability

Flash Player 6.

Usage

TextField._yscale

Description

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

See also

TextField._x, TextField._xscale, TextField._y

TextFormat (object)

The TextFormat object represents character formatting information.

You must use the constructor `new TextFormat` to create an instance of the TextFormat object before calling its methods.

You can set TextFormat parameters to `null` to indicate that they are undefined. When you apply a TextFormat object to a text field using the `setTextFormat` method, only its defined properties are applied, as in the following example:

```
myTextFormat = new TextFormat();
myTextFormat.bold = true;
myTextField.setTextFormat(myTextFormat);
```

This code first creates an empty TextFormat object with all of its properties undefined, then sets the `bold` property to a defined value.

The code `myTextField.setTextFormat(myTextFormat)` only changes the `bold` property of the text field's default text format, because the `bold` property is the only one defined in `myTextFormat`. All other aspects of the text field's default text format remain unchanged.

When `getTextFormat` is invoked, a TextFormat object is returned with all of its properties defined; no property is `null`.

Method summary for the `TextFormat` object

Method	Description
<code>TextFormat.getTextExtent</code>	Returns an object with two properties, <code>width</code> and <code>height</code> , that indicate the size of text in a text field.

Property summary for the `TextFormat` object

Property	Description
<code>TextFormat.align</code>	Indicates the alignment of a paragraph.
<code>TextFormat.blockIndent</code>	Indicates the block indentation in points.
<code>TextFormat.bold</code>	Indicates whether text is boldface.
<code>TextFormat.bullet</code>	Indicates whether or not text is in a bulleted list.
<code>TextFormat.color</code>	Indicates the color of text.
<code>TextFormat.font</code>	Indicates the font name of the text with a text format.
<code>TextFormat.indent</code>	Indicates the indentation from the left margin to the first character in the paragraph.
<code>TextFormat.italic</code>	Indicates whether text is italicized.
<code>TextFormat.leading</code>	Indicates the amount of leading vertical space between lines.
<code>TextFormat.leftMargin</code>	Indicates the left margin of the paragraph, in points.
<code>TextFormat.rightMargin</code>	Indicates the right margin of the paragraph, in points.
<code>TextFormat.tabStops</code>	Specifies custom tab stops.
<code>TextFormat.target</code>	Indicates the window in a browser where a hyperlink is displayed.
<code>TextFormat.size</code>	Indicates the point size of text.
<code>TextFormat.underline</code>	Indicates whether text is underlined.
<code>TextFormat.url</code>	Indicates the URL to which the text links.

Constructor for the `TextFormat` object

Usage

```
new TextFormat([font, [size, [color, [bold, [italic, [underline, [url,  
    [target, [align, [leftMargin, [rightMargin, [indent, [leading]]]]]]]]]]])
```

Parameters

font The name of a font for text as a string.

size An integer that indicates the point size.

color The color of text using this text format. A number containing three 8-bit RGB components; for example, 0xFF0000 is red, 0x00FF00 is green.

bold A Boolean value that indicates whether the text is boldface.

italic A Boolean value that indicates whether the text is italicized.

underline A Boolean value that indicates whether the text is underlined.

url The URL to which the text in this text format hyperlinks. If *url* is an empty string, the text does not have a hyperlink.

target The target window where the hyperlink is displayed. If the target window is an empty string, the text is displayed in the default target window `_self`. If the `TextFormat.url` property is set to empty string or to the value `null`, this property may be get or set but has no effect.

align The alignment of the paragraph, represented as a string. If `"left"`, the paragraph is left-aligned. If `"center"`, the paragraph is centered. If `"right"`, the paragraph is right-aligned.

leftMargin Indicates the left margin of the paragraph, in points.

rightMargin Indicates the right margin of the paragraph, in points.

indent An integer that indicates the indentation from the left margin to the first character in the paragraph.

leading A number that indicates the amount of leading vertical space between lines.

Description

Constructor; creates an instance of the `TextFormat` object with the specified properties. You can then change the properties of the `TextFormat` object to change the formatting of text fields.

Any parameter may be set to the value `null` to indicate that it is not defined. All of the parameters are optional; any omitted parameters are treated as `null`.

Availability

Flash Player 6.

TextFormat.align

Availability

Flash Player 6.

Usage

`TextFormat.align`

Description

Property; indicates the alignment of the paragraph, represented as a string. The alignment of the paragraph, represented as a string. If `"left"`, the paragraph is left-aligned. If `"center"`, the paragraph is centered. If `"right"`, the paragraph is right-aligned. The default value is `null` which indicates that the property is undefined.

TextFormat.blockIndent

Availability

Flash Player 6.

Usage

`TextFormat.blockIndent`

Description

Property; a number that indicates the block indentation in points. Block indentation is applied to an entire block of text; that is, to all lines of the text. In contrast, normal indentation (`TextFormat.indent`) only affects the first line of each paragraph. If this property is `null`, the `TextFormat` object does not specify block indentation.

TextFormat.bold

Availability

Flash Player 6.

Usage

TextFormat.bold

Description

Property; a Boolean value that indicates if the text is boldface. The default value is `null`, which indicates that the property is undefined.

TextFormat.bullet

Availability

Flash Player 6.

Usage

TextFormat.bullet

Description

Property; a Boolean value that indicates that the text is part of a bulleted list. In a bulleted list, each paragraph of text is indented. To the left of the first line of each paragraph, a bullet symbol is displayed. If this property is `null`, the `TextFormat` object does not specify that text be bulleted or not be bulleted.

TextFormat.color

Availability

Flash Player 6.

Usage

TextFormat.color

Description

Property; indicates the color of text. A number containing three 8-bit RGB components; for example, `0xFF0000` is red, `0x00FF00` is green.

TextFormat.font

Availability

Flash Player 6.

Usage

TextFormat.font

Description

Property; The name of the font for text in this text format, as a string. The default value is `null` which indicates that the property is undefined.

TextFormat.getTextExtent

Availability

Flash Player 6.

Usage

```
TextFormat.getTextExtent (text)
```

Parameters

text A string.

Returns

An object with the properties `width` and `height`.

Description

Method; returns the size of the text string specified in the *text* parameter in this character format. The return value is an object of class `Object` with two properties, `width`, and `height`. The *text* is treated as plain text (not HTML). The *text* is a single line of text; carriage returns and line feeds are ignored, and no word wrap is applied.

TextFormat.indent

Availability

Flash Player 6.

Usage

```
TextFormat.indent
```

Description

Property; an integer that indicates the indentation from the left margin to the first character in the paragraph. The default value is `null`, which indicates that the property is undefined.

TextFormat.italic

Availability

Flash Player 6.

Usage

```
TextFormat.italic
```

Description

Property; a Boolean value that indicates whether text in this text format is italicized. The default value is `null`, which indicates that the property is undefined.

TextFormat.leading

Availability

Flash Player 6.

Usage

TextFormat.leading

Description

Property; the amount of leading vertical space between lines. The default value is `null`, which indicates that the property is undefined.

TextFormat.leftMargin

Availability

Flash Player 6.

Usage

TextFormat.leftMargin

Description

Property; the left margin of the paragraph, in points. The default value is `null`, which indicates that the property is undefined.

TextFormat.rightMargin

Availability

Flash Player 6.

Usage

TextFormat.rightMargin

Description

Property; the right margin of the paragraph, in points. The default value is `null`, which indicates that the property is undefined.

TextFormat.size

Availability

Flash Player 6.

Usage

TextFormat.size

Description

Property; the point size of text in this text format. The default value is `null`, which indicates that the property is undefined.

TextFormat.tabStops

Availability

Flash Player 6.

Usage

TextFormat.tabStops

Description

Property; specifies custom tab stops as an Array of non-negative integers. Each tab stop is specified in points. If custom tab stops are not specified (*null*), the default tab stop is 4 (average character width).

TextFormat.target

Availability

Flash Player 6.

Usage

TextFormat.target

Description

Property; indicates the target window where the hyperlink is displayed. If the target window is an empty string, the text is displayed in the default target window *_self*. If the *TextFormat.url* property is set to empty string or to the value *null*, this property may be get or set but has no effect.

TextFormat.underline

Availability

Flash Player 6.

Usage

TextFormat.underline

Description

Property; a Boolean value that indicates if the text that uses this *TextFormat* is underlined. If *underline* is set to *true*, text in this text format is underlined. If *false*, text in this text format is not underlined. Note that this is the same underline accomplished by the *<U>* tag, which is not "true" underlining because it does not skip descenders correctly. The default value is *null*, which indicates that the property is undefined.

TextFormat.url

Availability

Flash Player 6.

Usage

TextFormat.url

Description

Property; indicates the URL that text in this text format hyperlinks to. If the `url` property is an empty string, the text does not have a hyperlink. The default value is `null`, which indicates that the property is undefined.

this

Availability

Flash Player 5.

Usage

this

Description

Keyword; references an object or movie clip instance. When a script executes, `this` references the movie clip instance that contains the script. When a method is called, `this` contains a reference to the object that contains the called method.

Inside an `onEvent` handler action attached to a button, `this` refers to the Timeline that contains the button. Inside an `onClipEvent` event handler action attached to a movie clip, `this` refers to the Timeline of the movie clip itself.

Example

In the following example, the keyword `this` references the `Circle` object.

```
function Circle(radius) {  
    this.radius = radius;  
    this.area = Math.PI * radius * radius;  
}
```

In the following statement assigned to a frame, the keyword `this` references the current movie clip.

```
// sets the alpha property of the current movie clip to 20  
this._alpha = 20;
```

In the following statement inside an `onClipEvent` handler, the keyword `this` references the current movie clip.

```
// when the movie clip loads, a startDrag operation  
// is initiated for the current movie clip.
```

```
onClipEvent (load) {  
    startDrag (this, true);  
}
```

See also

`new`

toggleHighQuality

Availability

Flash 2.

Usage

`toggleHighQuality()`

Parameters

None.

Returns

Nothing.

Description

Action; turns anti-aliasing on and off in Flash Player. Anti-aliasing smooths the edges of objects and slows down the movie playback. The `toggleHighQuality` action affects all movies in the Flash Player.

Example

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

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

See also

`_quality`, `_highquality`

trace

Availability

Flash Player 4.

Usage

`trace(expression)`

Parameters

expression An expression to evaluate. When a SWF file is opened in the Flash authoring tool (via the Test Movie command), the value of the *expression* parameter is displayed in the Output window.

Returns

Nothing.

Description

Action; evaluates the *expression* and displays the result in the Output window in test mode.

Use `trace` to record programming notes or to display messages in the Output window while testing a movie. Use the *expression* parameter to check if a condition exists, or to display values in the Output window. The `trace` action is similar to the `alert` function in JavaScript.

You can use the Omit Trace actions command in Publish Settings to remove `trace` actions from the exported SWF file.

Example

This example is from a game in which a draggable movie clip instance named `rabbi` must be released on a specific target. A conditional statement evaluates the `_droptarget` property and executes different actions depending on where `rabbi` is released. The `trace` action is used at the end of the script to evaluate the location of the `rabbi` movie clip, and display the result in the Output window. If `rabbi` doesn't behave as expected (for example, if it snaps to the wrong target), the values sent to the Output window by the `trace` action will help you determine the problem in the script.

```
on(press) {
    rabbi.startDrag();
}

on(release) {
    if(eval(_droptarget) != target) {
        rabbi._x = rabbi._x;
        rabbi._y = rabbi._y;
    } else {
        rabbi._x = rabbi._x;
        rabbi._y = rabbi._y;
        target = "_root.pasture";
    }
    trace("rabbi_y = " + rabbi._y);
    trace("rabbi_x = " + rabbi._x);
    stopDrag();
}
```

true

Availability

Flash Player 5.

Usage

true

Description

A unique Boolean value that represents the opposite of `false`.

See also

false

typeof

Availability

Flash Player 5.

Usage

`typeof expression`

Parameters

expression A string, movie clip, button, object, or function.

Description

Operator; a unary operator placed before a single parameter. The `typeof` operator causes the Flash interpreter to evaluate *expression*; the result is a string specifying whether the expression is a string, movie clip, object, or function. The following table shows the results of the `typeof` operator on each type of expression:

Parameter	Output
String	string
MovieClip	movieclip
Button	object
Text field	object
Number	number
Boolean	boolean
Object	object
Function	function

undefined

Availability

Flash Player 5.

Usage

`undefined`

Parameters

None.

Returns

Nothing.

Description

A special value, usually used to indicate that a variable has not yet been assigned a value. A reference to an undefined value returns the special value `undefined`. The ActionScript code `typeof(undefined)` returns the string `"undefined"`. The only value of type `undefined` is `undefined`.

When `undefined` is converted to a string, it converts to the empty string.

The value `undefined` is similar to the special value `null`. In fact, when `null` and `undefined` are compared with the equality operator, they compare as equal.

Example

In this example, the variable `x` has not been declared and therefore has the value `undefined`. In the first section of code, the equality operator (`==`) compares the value of `x` to the value `undefined` and the appropriate result is sent to the Output window. In the second section of code, the equality operator compares the values `null` and `undefined`.

```
// x has not been declared
trace ("The value of x is " + x);
if (x == undefined) {
    trace ("x is undefined");
} else {
    trace ("x is not undefined");
}

trace ("typeof (x) is " + typeof (x));
if (null == undefined) {
    trace ("null and undefined are equal");
} else {
    trace ("null and undefined are not equal");
}
```

The following result is displayed in the Output window:

```
The value of x is x is undefined
typeof (x) is undefined
null and undefined are equal
```

Note: In the ECMA-262 specification, `undefined` converts to the string `"undefined"`, not the empty string. This is a difference between ActionScript and the ECMA-262 specification.

unescape

Availability

Flash Player 5.

Usage

`unescape(x)`

Parameters

`x` A string with hexadecimal sequences to escape.

Returns

Nothing.

Description

Top-level function; evaluates the parameter `x` as a string, decodes the string from URL-encoded format (converting all hexadecimal sequences to ASCII characters), and returns the string.

Example

The following example illustrates the escape-to-unescape conversion process.

```
escape("Hello{[World]}");
```

The escaped result is as follows:

```
("Hello%7B%5BWorld%5D%7D");
```

Use unescape to return to the original format:

```
unescape("Hello%7B%5BWorld%5D%7D")
```

The result is as follows:

```
Hello{[World]}
```

unloadMovie

Availability

Flash Player 3.

Usage

```
unloadMovie[Num](level/target)
```

Parameters

level The level (`_levelN`) of a loaded movie. When you unload a movie from a level, the `unloadMovie` action in the Actions panel in normal mode switches to `unloadMovieNum`; in expert mode, you must specify `unloadMovieNum` or choose it from the Actions toolbox.

target The target path of a movie clip.

Returns

None.

Description

Action; removes a loaded movie or a movie clip from the Flash Player. To unload a movie that was loaded into a level in the Flash Player, use the *level* parameter. To unload a loaded movie clip, use the *target* parameter.

Example

The following example unloads the movie clip `draggable` on the main Timeline, and loads the movie `movie.swf` into level 4.

```
on (press) {  
    unloadMovie ("_root.draggable");  
    loadMovieNum ("movie.swf", 4);  
}
```

The following example unloads the movie loaded into level 4:

```
on (press) {  
    unloadMovieNum (4);  
}
```

See also

`loadMovie`, `loadMovieNum`, `unloadMovieNum`

unloadMovieNum

Availability

Flash Player 3.

Usage

```
unloadMovieNum(level)
```

Parameters

level The level (`_levelN`) of a loaded movie.

Returns

Nothing.

Description

Action; removes a loaded movie from the Flash Player.

See also

`loadMovie`, `loadMovieNum`

updateAfterEvent

Availability

Flash Player 5.

Usage

```
updateAfterEvent()
```

Parameters

None.

Returns

Nothing.

Description

Action; updates the display (independent of the frames per second set for the movie) when you call it within an `onClipEvent` handler or as part of a function or method that you pass to `setInterval`. Flash ignores calls to `updateAfterEvent` that are not within an `onClipEvent` handler or part of a function or method passed to `setInterval`.

See also

`onClipEvent`, `setInterval`

var

Availability

Flash Player 5.

Usage

```
var variableName1 [= value1] [...,variableNameN [=valueN]]
```

Parameters

variableName An identifier.

value The value assigned to the variable.

Returns

Nothing.

Description

Action; used to declare local variables. If you declare local variables inside a function, the variables are defined for the function and expire at the end of the function call. If variables are not declared inside a block (`{ }`), but the action list was executed with a `call` action, the variables are local and expire at the end of the current list. If variables are not declared inside a block and the current action list was not executed with the `call` action, the variables are not local.

Example

The following examples use the `var` action to declare and assign variables:

```
var x;  
var y = 1;  
var z = 3, w = 4;  
var s, t, u = z;
```

void

Availability

Flash Player 5.

Usage

```
void (expression)
```

Description

Operator; a unary operator that discards the *expression* value and returns an undefined value.

The `void` operator is often used in comparisons using the `==` operator to test for undefined values.

while

Availability

Flash Player 4.

Usage

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

Parameters

condition The expression that is reevaluated each time the `while` action is executed. If the statement evaluates to `true`, the *statement(s)* is run.

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

Returns

Nothing.

Description

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

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

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

The `while` statement performs the following series of steps. Each repetition of steps 1–4 is called an *iteration* of the loop. The *condition* is retested at the beginning of each iteration, as in the following steps:

- 1 The expression *condition* is evaluated.
- 2 If *condition* evaluates to `true` or a value that converts to the Boolean value `true`, such as a nonzero number, go to step 3.
Otherwise, the `while` statement is completed and execution resumes at the next statement after the `while` loop.
- 3 Run the statement block *statement(s)*.
- 4 Go to step 1.

Example

This example duplicates five movie clips on the Stage, each with a randomly generated *x* and *y* position, *xscale* and *yscale*, and *_alpha* property to achieve a scattered effect. The variable *foo* is initialized with the value 0. The *condition* parameter is set so that the *while* loop will run five times, or as long as the value of the variable *foo* is less than 5. Inside the *while* loop, a movie clip is duplicated and *setProperty* is used to adjust the various properties of the duplicated movie clip. The last statement of the loop increments *foo* so that when the value reaches 5, the *condition* parameter evaluates to *false*, and the loop will not be executed.

```
on(release) {  
    foo = 0;  
    while(foo < 5) {  
        duplicateMovieClip("_root.flower", "mc" + foo, foo);  
        setProperty("mc" + foo, _x, random(275));  
        setProperty("mc" + foo, _y, random(275));  
        setProperty("mc" + foo, _alpha, random(275));  
        setProperty("mc" + foo, _xscale, random(200));  
        setProperty("mc" + foo, _yscale, random(200));  
        foo++;  
    }  
}
```

See also

do while, continue, for, for..in

with

Availability

Flash Player 5.

Usage

```
with (object) {  
    statement(s);  
}
```

Parameters

object An instance of an ActionScript object or movie clip.

statement(s) An action or group of actions enclosed in curly brackets.

Returns

Nothing.

Description

Action; allows you to specify an object (such as a movie clip) with the *object* parameter and evaluate expressions and actions inside that object with the *statement(s)* parameter. This prevents you from having to repeatedly write the object's name or the path to the object.

The *object* parameter becomes the context in which the properties, variables, and functions in the *statement(s)* parameter are read. For example, if *object* is *myArray*, and two of the properties specified are *length* and *concat*, those properties are automatically read as *myArray.length* and *myArray.concat*. In another example, if *object* is *state.california*, any actions or statements inside the *with* action are called from inside the *california* instance.

To find the value of an identifier in the `statement(s)` parameter, ActionScript starts at the beginning of the scope chain specified by the `object` and searches for the identifier at each level of the scope chain, in a specific order.

The scope chain used by the `with` action to resolve identifiers starts with the first item in the following list and continues to the last item:

- The object specified in the `object` parameter in the innermost `with` action.
- The object specified in the `object` parameter in the outermost `with` action.
- The Activation object. (A temporary object that is automatically created when a function is called that holds the local variables called in the function.)
- The movie clip containing the currently executing script.
- The Global object (built-in objects such as `Math` and `String`).

To set a variable inside a `with` action, the variable must have been declared outside the `with` action or you must enter the full path to the Timeline on which you want the variable to live. If you set a variable in a `with` action without declaring it, the `with` action will look for the value according to the scope chain. If the variable doesn't already exist, the new value will be set on the Timeline from which the `with` action was called.

In Flash 5, the `with` action replaces the deprecated `tellTarget` action. You are encouraged to use `with` instead of `tellTarget` because it is a standard ActionScript extension to the ECMA-262 standard. The principal difference between the `with` and `tellTarget` actions is that `with` takes a reference to a movie clip or other object as its parameter, while `tellTarget` takes a target path string that identifies a movie clip as its parameter, and cannot be used to target objects.

Example

The following example sets the `x` and `y` properties of the `someOtherMovieClip` instance, and then instructs `someOtherMovieClip` to go to frame 3 and stop.

```
with (someOtherMovieClip) {  
    _x = 50;  
    _y = 100;  
    gotoAndStop(3);  
}
```

The following code snippet shows how to write the preceding code without using a `with` action.

```
someOtherMovieClip._x = 50;  
someOtherMovieClip._y = 100;  
someOtherMovieClip.gotoAndStop(3);
```

You could also write this code using the `tellTarget` action. However, if `someOtherMovieClip` were not a movie clip, but an object, you could not use the `with` action.

```
tellTarget ("someOtherMovieClip") {  
    _x = 50;  
    _y = 100;  
    gotoAndStop(3);  
}
```

The `with` action is useful for accessing multiple items in a scope chain list simultaneously. In the following example, the built-in `Math` object is placed at the front of the scope chain. Setting `Math` as a default object resolves the identifiers `cos`, `sin`, and `PI` to `Math.cos`, `Math.sin`, and `Math.PI`, respectively. The identifiers `a`, `x`, `y`, and `r` are not methods or properties of the `Math` object, but since they exist in the object activation scope of the function `polar`, they resolve to the corresponding local variables.

```
function polar(r) {  
    var a, x, y;  
    with (Math) {  
        a = PI * r * r;  
        x = r * cos(PI);  
        y = r * sin(PI/2);  
    }  
    trace("area = " + a);  
    trace("x = " + x);  
    trace("y = " + y);  
}
```

You can use nested `with` actions to access information in multiple scopes. In the following example, the instance `fresno` and the instance `salinas` are children of the instance `california`. The statement sets the `_alpha` values of `fresno` and `salinas` without changing the `_alpha` value of `california`.

```
with (california){  
    with (fresno){  
        _alpha = 20;  
    }  
    with (salinas){  
        _alpha = 40;  
    }  
}
```

See also

`tellTarget`

XML (object)

Use the methods and properties of the XML object to load, parse, send, build, and manipulate XML document trees. In Flash MX, the XML object has become a native object. As such, you will experience dramatic improvement in performance.

You must use the constructor `new XML()` to create an instance of the XML object before calling any of the methods of the XML object.

XML is supported by Flash Player 5 and Flash Player 6.

Method summary for the XML object

Method	Description
<code>XML.appendChild</code>	Appends a node to the end of the specified object's child list.
<code>XML.cloneNode</code>	Clones the specified node and, optionally, recursively clones all children.
<code>XML.createElement</code>	Creates a new XML element.
<code>XML.createTextNode</code>	Creates a new XML text node.
<code>XML.getBytesLoaded</code>	Returns the number of bytes loaded for the specified XML document.
<code>XML.getBytesTotal</code>	Returns the size of the XML document in bytes.
<code>XML.hasChildNodes</code>	Returns <code>true</code> if the specified node has child nodes; otherwise, returns <code>false</code> .
<code>XML.insertBefore</code>	Inserts a node in front of an existing node in the specified node's child list.
<code>XML.load</code>	Loads a document (specified by the XML object) from a URL.
<code>XML.parseXML</code>	Parses an XML document into the specified XML object tree.
<code>XML.removeNode</code>	Removes the specified node from its parent.
<code>XML.send</code>	Sends the specified XML object to a URL.
<code>XML.sendAndLoad</code>	Sends the specified XML object to a URL and loads the server response into another XML object.
<code>XML.toString</code>	Converts the specified node and any children to XML text.

Property summary for the XML object

Property	Description
<code>XML.contentType</code>	Indicates the MIME type transmitted to the server.
<code>XML.docTypeDecl</code>	Sets and returns information about an XML document's <code>DOCTYPE</code> declaration.
<code>XML.firstChild</code>	References the first child in the list for the specified node.
<code>XML.ignoreWhite</code>	When set to <code>true</code> , text nodes that only contain white space are discarded during the parsing process.
<code>XML.lastChild</code>	References the last child in the list for the specified node.
<code>XML.load</code>	Checks if the specified XML object has loaded.
<code>XML.nextSibling</code>	References the next sibling in the parent node's child list.
<code>XML.nodeName</code>	Returns the tag name of an XML element.
<code>XML.nodeType</code>	Returns the type of the specified node (XML element or text node).
<code>XML.nodeValue</code>	Returns the text of the specified node if the node is a text node.
<code>XML.parentNode</code>	References the parent node of the specified node.
<code>XML.previousSibling</code>	References the previous sibling in the parent node's child list.
<code>XML.status</code>	Returns a numeric status code indicating the success or failure of an XML document parsing operation.
<code>XML.xmlDecl</code>	Sets and returns information about an XML document's document declaration.

Collections summary for the XML object

Method	Description
<code>XML.attributes</code>	Returns an associative array containing all of the attributes of the specified node.
<code>XML.childNodes</code>	Returns an array containing references to the child nodes of the specified node.

Event handler summary for the XML object

Method	Description
<code>XML.onData</code>	A callback function that is invoked when XML text has been completely downloaded from the server, or when an error occurs downloading XML text from a server.
<code>XML.onLoad</code>	A callback function for <code>load</code> and <code>sendAndLoad</code> .

Constructor for the XML object

Availability

Flash Player 5.

Usage

```
new XML([source])
```

Parameters

source The XML text parsed to create the new XML object.

Returns

Nothing.

Description

Constructor; creates a new XML object. You must use the constructor method to create an instance of the XML object before calling any of the XML object methods.

Note: The `createElement` and `createTextNode` methods are the 'constructor' methods for creating the elements and text nodes in an XML document tree.

Example

Usage 1: The following example creates a new, empty XML object.

```
myXML = new XML();
```

Usage 2: The following example creates an XML object by parsing the XML text specified in the *source* parameter, and populates the newly created XML object with the resulting XML document tree.

```
anotherXML = new XML("<state>California<city>san francisco</city></state>");
```

See also

`XML.createElement`, `XML.createTextNode`

XML.appendChild

Availability

Flash Player 5.

Usage

myXML.appendChild(childNode)

Parameters

childNode The child node to be added to the specified XML object's child list.

Returns

Nothing.

Description

Method; appends the specified child node to the XML object's child list. The appended child node is placed in the tree structure once removed from its existing parent node, if any.

Example

The following example clones the last node from `doc1` and appends it to `doc2`.

```
doc1 = new XML(src1);  
doc2 = new XML();  
node = doc1.lastChild.cloneNode(true);  
doc2.appendChild(node);
```

XML.attributes

Availability

Flash Player 5.

Usage

myXML.attributes

Parameters

None.

Returns

Nothing.

Description

Collection (read-write); returns an associative array containing all attributes of the specified XML object.

Example

The following example writes the names of the XML attributes to the Output window.

```
str = "<mytag name=\"Val\"> item </mytag>";
doc = new XML(str);
y = doc.firstChild.attributes.name;
    trace (y);
doc.firstChild.attributes.order = "first";
z = doc.firstChild.attributes.order
    trace(z);
```

The following is written to the Output window:

```
Val
first
```

XML.childNodes

Availability

Flash Player 5.

Usage

myXML.childNodes

Parameters

None.

Returns

Nothing.

Description

Collection (read-only); returns an array of the specified XML object's children. Each element in the array is a reference to an XML object that represents a child node. This is a read-only property and cannot be used to manipulate child nodes. Use the methods `appendChild`, `insertBefore`, and `removeNode` to manipulate child nodes.

This collection is undefined for text nodes (`nodeType == 3`).

See also

`XML.nodeType`

XML.cloneNode

Availability

Flash Player 5.

Usage

myXML.cloneNode(*deep*)

Parameters

deep Boolean value specifying whether the children of the specified XML object are recursively cloned.

Returns

Nothing.

Description

Method; constructs and returns a new XML node of the same type, name, value, and attributes as the specified XML object. If *deep* is set to `true`, all child nodes are recursively cloned, resulting in an exact copy of the original object's document tree.

The clone of the node that is returned is no longer associated with the tree of the cloned item. Consequently, `nextSibling`, `parentNode`, and `previousSibling` all have a value of `null`. If a clip copy is not performed, `firstChild` and `lastChild` are also `null`.

XML.contentType

Availability

Flash Player 6.

Usage

```
myXML.contentType
```

Description

Property; the MIME type that is sent to the server when you call the `XML.send` or `XML.sendAndLoad` method. The default is *application/x-www-form-urlencoded*.

See also

`XML.send`, `XML.sendAndLoad`

XML.createElement

Availability

Flash Player 5.

Usage

```
myXML.createElement(name)
```

Parameters

name The tag name of the XML element being created.

Returns

Nothing.

Description

Method; creates a new XML element with the name specified in the parameter. The new element initially has no parent, no children, and no siblings. The method returns a reference to the newly created XML object representing the element. This method and `createTextNode` are the constructor methods for creating nodes for an XML object.

XML.createTextNode

Availability

Flash Player 5.

Usage

```
myXML.createTextNode(text)
```

Parameters

text The text used to create the new text node.

Returns

Nothing.

Description

Method; creates a new XML text node with the specified text. The new node initially has no parent, and text nodes cannot have children or siblings. This method returns a reference to the XML object representing the new text node. This method and `createElement` are the constructor methods for creating nodes for an XML object.

XML.docTypeDecl

Availability

Flash Player 5.

Usage

```
myXML.XMLdocTypeDecl
```

Description

Property; sets and returns information about the XML document DOCTYPE declaration. After the XML text has been parsed into an XML object, the `XML.docTypeDecl` property of the XML object is set to the text of the XML document's DOCTYPE declaration. For example, `<!DOCTYPE greeting SYSTEM "hello.dtd">`. This property is set using a string representation of the DOCTYPE declaration, not an XML node object.

ActionScript's XML parser is not a validating parser. The DOCTYPE declaration is read by the parser and stored in the `docTypeDecl` property, but no DTD validation is performed.

If no DOCTYPE declaration was encountered during a parse operation, `XML.docTypeDecl` is set to undefined. `XML.toString` outputs the contents of `XML.docTypeDecl` immediately after the XML declaration stored in `XML.xmlDecl`, and before any other text in the XML object. If `XML.docTypeDecl` is undefined, no DOCTYPE declaration is output.

Example

The following example uses `XML.docTypeDecl` to set the DOCTYPE declaration for an XML object:

```
myXML.docTypeDecl = "<!DOCTYPE greeting SYSTEM \"hello.dtd\">";
```

See also

`XML.toString`, `XML.xmlDecl`

XML.firstChild

Availability

Flash Player 5.

Usage

myXML.firstChild

Description

Property (read-only); evaluates the specified XML object and references the first child in the parent node's children list. This property is `null` if the node does not have children. This property is undefined if the node is a text node. This is a read-only property and cannot be used to manipulate child nodes; use the methods `appendChild`, `insertBefore`, and `removeNode` to manipulate child nodes.

See also

`XML.appendChild`, `XML.insertBefore`, `XML.removeNode`

XML.getBytesLoaded

Availability

Flash Player 6.

Usage

XML.getBytesLoaded()

Parameters

None.

Returns

An integer indicating the number of bytes loaded.

Description

Method; returns the number of bytes loaded (streamed) for the XML document. You can compare the value of `getBytesLoaded` with the value of `getBytesTotal` to determine what percentage of an XML document has loaded.

See also

`XML.getBytesTotal`

XML.getBytesTotal

Availability

Flash Player 6.

Usage

XML.getBytesTotal()

Parameters

None.

Returns

An integer.

Description

Method; returns the size, in bytes, of the XML document.

See also

`XML.getBytesLoaded`

XML.hasChildNodes

Availability

Flash Player 5.

Usage

```
myXML.hasChildNodes()
```

Parameters

None.

Returns

Nothing.

Description

Method; returns `true` if the specified XML object has child nodes; otherwise, returns `false`.

Example

The following example uses the information from the XML object in a user-defined function.

```
if (rootNode.hasChildNodes()) {  
    myfunc (rootNode.firstChild);  
}
```

XML.ignoreWhite

Availability

Flash Player 5.

Usage

```
myXML.ignoreWhite = boolean  
XML.prototype.ignoreWhite = boolean
```

Parameters

boolean A Boolean (true or false) value.

Description

Property; Default setting is `false`. When set to `true`, text nodes that only contain white space are discarded during the parsing process. Text nodes with leading or trailing white space are unaffected.

Usage 1: You can set the `ignoreWhite` property for individual XML objects, as in the following code:

```
myXML.ignoreWhite = true
```

XML.insertBefore

Availability

Flash Player 5.

Usage

```
myXML.insertBefore(childNode, beforeNode)
```

Parameters

childNode The node to be inserted.

beforeNode The node before the insertion point for the *childNode*.

Returns

Nothing.

Description

Method; inserts a new child node into the XML object's child list, before the *beforeNode* node. If the *beforeNode* parameter is undefined or null, the node is added using `appendChild`. If *beforeNode* is not a child of *myXML*, the insert fails.

XML.lastChild

Availability

Flash Player 5.

Usage

```
myXML.lastChild
```

Description

Property (read-only); evaluates the XML object and references the last child in the parent node's child list. This method returns null if the node does not have children. This is a read-only property and cannot be used to manipulate child nodes; use the methods `appendChild`, `insertBefore`, and `removeNode` to manipulate child nodes.

See also

`XML.appendChild`, `XML.insertBefore`, `XML.removeNode`

XML.load

Availability

Flash Player 5.

Usage

```
myXML.load(url)
```

Parameters

url The URL where the XML document to be loaded is located. The URL must be in the same subdomain as the URL where the movie currently resides.

Returns

Nothing.

Description

Method; loads an XML document from the specified URL, and replaces the contents of the specified XML object with the downloaded XML data. The load process is asynchronous; it does not finish immediately after the `load` method is executed. When `load` is executed, the XML object property `loaded` is set to `false`. When the XML data finishes downloading, the `loaded` property is set to `true`, and the `onLoad` method is invoked. The XML data is not parsed until it is completely downloaded. If the XML object previously contained any XML trees, they are discarded.

You can specify your own callback function in place of the `onLoad` method.

Example

The following is a simple example using `XML.load`:

```
doc = new XML();
doc.load ("theFile.xml");
```

See also

`XML.loaded`, `XML.onLoad`

XML.loaded

Availability

Flash Player 5.

Usage

myXML.loaded

Description

Property (read-only); determines whether the document-loading process initiated by the `XML.load` call has completed. If the process completes successfully, the method returns `true`; otherwise, it returns `false`.

Example

The following example uses `XML.loaded` in a simple script.

```
if (doc.loaded) {
    gotoAndPlay(4);
}
```

XML.nextSibling

Availability

Flash Player 5.

Usage

myXML.nextSibling

Description

Property (read-only); evaluates the XML object and references the next sibling in the parent node's child list. This method returns `null` if the node does not have a next sibling node. This is a read-only property and cannot be used to manipulate child nodes. Use the methods `appendChild`, `insertBefore`, and `removeNode` to manipulate child nodes.

See also

`XML.appendChild`, `XML.insertBefore`, `XML.removeNode`

XML.nodeName

Availability

Flash Player 5.

Usage

myXML.nodeName

Description

Property; takes or returns the node name of the XML object. If the XML object is an XML element (`nodeType == 1`), `nodeName` is the name of the tag representing the node in the XML file. For example, `TITLE` is the `nodeName` of an HTML `TITLE` tag. If the XML object is a text node (`nodeType == 3`), the `nodeName` is `null`.

See also

`XML.nodeType`

XML.nodeType

Availability

Flash Player 5.

Usage

myXML.nodeType

Description

Property (read-only); takes or returns a `nodeType` value, where 1 is an XML element and 3 is a text node.

See also

`XML.nodeValue`

XML.nodeValue

Availability

Flash Player 5.

Usage

myXML.nodeValue

Description

Property; returns the node value of the XML object. If the XML object is a text node, the `nodeType` is 3, and the `nodeValue` is the text of the node. If the XML object is an XML element (`nodeType` is 1), it has a `null` `nodeValue` and is read-only.

See also

`XML.nodeType`

XML.onData

Availability

Flash Player 5

Usage

myXML.onData()

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when XML text has been completely downloaded from the server, or when an error occurs downloading XML text from a server. This handler is invoked before the XML is parsed and therefore can be used to call a custom parsing routine instead of using the Flash XML parser. The `XML.onData` method returns either the value `undefined`, or a string that contains XML text downloaded from the server. If the returned value is `undefined`, an error occurred while downloading the XML from the server.

By default, the `XML.onData` method invokes the `XML.onLoad` method. You can override the `XML.onData` method with your own behavior, but `XML.onLoad` will no longer be called unless you call it in your implementation of `XML.onData`.

Example

The following example shows what the `onData` method looks like by default:

```
XML.prototype.onData = function (src) {  
    if (src == undefined) {  
        this.onLoad(false);  
    } else {  
        this.parseXML(src);  
        this.loaded = true;  
        this.onLoad(true);  
    }  
}
```

The `XML.onData` method can be overridden to intercept the XML text without parsing it.

XML.onLoad

Availability

Flash Player 5.

Usage

myXML.onLoad(*success*)

Parameters

success A Boolean value indicating whether the XML object was successfully loaded with a `XML.load` or `XML.sendAndLoad` operation.

Returns

Nothing.

Description

Method; invoked by the Flash Player when an XML document is received from the server. If the XML document is received successfully, the *success* parameter is *true*. If the document was not received, or if an error occurred in receiving the response from the server, the *success* parameter is *false*. The default implementation of this method is not active. To override the default implementation, you must assign a function containing your own actions.

Example

The following example creates a simple Flash movie for a simple e-commerce storefront application. The `sendAndLoad` method transmits an XML element containing the user's name and password, and installs an `onLoad` handler to handle the reply from the server.

```
function myOnLoad(success) {
    if (success) {
        if (e.firstChild.nodeName == "LOGINREPLY" &&
            e.firstChild.attributes.status == "OK") {
            gotoAndPlay("loggedIn")
        } else {
            gotoAndStop("loginFailed")
        }
    } else {
        gotoAndStop("connectionFailed")
    }
}
var myLoginReply = new XML();
myLoginReply.onLoad = myOnLoad;
myXML.sendAndLoad("http://www.samplestore.com/login.cgi",
    myLoginReply);
```

See also

`function`, `XML.load`, `XML.sendAndLoad`

XML.parentNode

Availability

Flash Player 5.

Usage

myXML.parentNode

Description

Property (read-only); references the parent node of the specified XML object, or returns *null* if the node has no parent. This is a read-only property and cannot be used to manipulate child nodes; use the methods `appendChild`, `insertBefore`, and `removeNode` to manipulate children.

XML.parseXML

Availability

Flash Player 5.

Usage

```
myXML.parseXML(source)
```

Parameters

source The XML text to be parsed and passed to the specified XML object.

Returns

Nothing.

Description

Method; parses the XML text specified in the *source* parameter, and populates the specified XML object with the resulting XML tree. Any existing trees in the XML object are discarded.

XML.previousSibling

Availability

Flash Player 5.

Usage

```
myXML.previousSibling
```

Description

Property (read-only); returns a reference to the previous sibling in the parent node's child list. Returns `null` if the node does not have a previous sibling node. This is a read-only property and cannot be used to manipulate child nodes; use the methods `appendChild`, `insertBefore`, and `removeNode` to manipulate child nodes.

XML.removeNode

Availability

Flash Player 5.

Usage

```
myXML.childNodes[1].removeNode()
```

Parameters

None.

Returns

Nothing.

Description

Method; removes the specified XML object from its parent. All descendents of the node are also deleted.

XML.send

Availability

Flash Player 5.

Usage

```
myXML.send(url, [window])
```

Parameters

url The destination URL for the specified XML object.

window The browser window to display data returned by the server: `_self` specifies the current frame in the current window, `_blank` specifies a new window, `_parent` specifies the parent of the current frame, and `_top` specifies the top-level frame in the current window. This parameter is optional; if no *window* parameter is specified, it is the same as specifying `_self`.

Returns

Nothing.

Description

Method; encodes the specified XML object into an XML document and sends it to the specified URL using the POST method.

XML.sendAndLoad

Availability

Flash Player 5.

Usage

```
myXML.sendAndLoad(url, targetXMLObject)
```

Parameters

url The destination URL for the specified XML object. The URL must be in the same subdomain as the URL where the movie was downloaded from.

targetXMLObject An XML object created with the XML constructor method that will receive the return information from the server.

Returns

Nothing.

Description

Method; encodes the specified XML object into an XML document, sends it to the specified URL using the POST method, downloads the server's response and then loads it into the *targetXMLObject* specified in the parameters. The server response is loaded in the same manner used by the `load` method.

See also

`XML.load`

XML.status

Availability

Flash Player 5.

Usage

myXML.status

Description

Property; automatically sets and returns a numeric value indicating whether an XML document was successfully parsed into an XML object. The numeric status codes and a description of each are listed as follows:

- 0 No error; parse was completed successfully.
- -2 A CDATA section was not properly terminated.
- -3 The XML declaration was not properly terminated.
- -4 The DOCTYPE declaration was not properly terminated.
- -5 A comment was not properly terminated.
- -6 An XML element was malformed.
- -7 Out of memory.
- -8 An attribute value was not properly terminated.
- -9 A start-tag was not matched with an end-tag.
- -10 An end-tag was encountered without a matching start-tag.

XML.toString

Availability

Flash Player 5.

Usage

myXML.toString()

Parameters

None.

Returns

Nothing.

Description

Method; evaluates the specified XML object, constructs a textual representation of the XML structure including the node, children, and attributes, and returns the result as a string.

For top-level XML objects (those created with the constructor), `XML.toString` outputs the document's XML declaration (stored in `XML.xmlDecl`), followed by the document's DOCTYPE declaration (stored in `XML.docTypeDecl`), followed by the text representation of all XML nodes in the object. The XML declaration is not output if `XML.xmlDecl` is undefined. The DOCTYPE declaration is not output if `XML.docTypeDecl` is undefined.

Example

The following code is an example of the `XML.toString` method that sends `<h1>test</h1>` to the output window.

```
node = new XML("<h1>test</h1>");
trace(node.toString());
```

See also

`XML.docTypeDecl`, `XML.xmlDecl`

XML.xmlDecl

Availability

Flash Player 5.

Usage

myXML.xmlDecl

Description

Property; sets and returns information about a document's XML declaration. After the XML document is parsed into an XML object, this property is set to the text of the document's XML declaration. This property is set using a string representation of the XML declaration, not an XML node object. If no XML declaration was encountered during a parse operation, the property is set to `undefined`. XML. The `toString` method outputs the contents of `XML.xmlDecl` before any other text in the XML object. If `XML.xmlDecl` contains the `undefined` type, no XML declaration is output.

Example

The following example uses `XML.xmlDecl` to set the XML document declaration for an XML object.

```
myXML.xmlDecl = "<?xml version=\"1.0\" ?>";
```

The following is an example of XML Declaration:

```
<?xml version="1.0" ?>
```

See also

`XML.docTypeDecl`, `XML.toString`

XMLSocket (object)

The XMLSocket object implements client sockets that allow the computer running the Flash Player to communicate with a server computer identified by an IP address or domain name.

Using the XMLSocket object

To use the XMLSocket object, the server computer must run a daemon that understands the protocol used by the XMLSocket object. The protocol is as follows:

- XML messages are sent over a full-duplex TCP/IP stream socket connection.
- Each XML message is a complete XML document, terminated by a zero byte.
- An unlimited number of XML messages can be sent and received over a single XMLSocket connection.

The XMLSocket object is useful for client-server applications that require low latency, such as real-time chat systems. A traditional HTTP-based chat solution frequently polls the server and downloads new messages using an HTTP request. In contrast, an XMLSocket chat solution maintains an open connection to the server, which allows the server to immediately send incoming messages without a request from the client.

Setting up a server to communicate with the XMLSocket object can be challenging. If your application does not require real-time interactivity, use the `loadVariables` action, or Flash HTTP-based XML server connectivity (`XML.load`, `XML.sendAndLoad`, `XML.send`), instead of the XMLSocket object.

To use the methods of the XMLSocket object, you must first use the constructor, `new XMLSocket`, to create a new XMLSocket object.

XMLSocket and security

Because the XMLSocket object establishes and maintains an open connection to the server, the following restrictions have been placed on the XMLSocket object for security reasons:

- The `XMLSocket.connect` method can connect only to TCP port numbers greater than or equal to 1024. One consequence of this restriction is that the server daemons that communicate with the XMLSocket object must also be assigned to port numbers greater than or equal to 1024. Port numbers below 1024 are often used by system services such as FTP, Telnet, and HTTP, thus the XMLSocket object is barred from these ports for security reasons. The port number restriction limits the possibility that these resources will be inappropriately accessed and abused.
- The `XMLSocket.connect` method can connect only to computers in the same subdomain where the SWF file (movie) resides. This restriction does not apply to movies running off a local disk. (This restriction is identical to the security rules for `loadVariables`, `XML.sendAndLoad`, and `XML.load`.)

Method summary for the XMLSocket object

Method	Description
<code>XMLSocket.close</code>	Closes an open socket connection.
<code>XMLSocket.connect</code>	Establishes a connection to the specified server.
<code>XMLSocket.send</code>	Sends an XML object to the server.

Event handler summary for the XMLSocket object

Method	Description
<code>XMLSocket.onClose</code>	A callback function that is invoked when an XMLSocket connection is closed.
<code>XMLSocket.onConnect</code>	A callback function that is invoked when an XMLSocket connection is established.
<code>XMLSocket.onData</code>	A callback function that is invoked when an XML message has been downloaded from the server.
<code>XMLSocket.onXML</code>	A callback function that is invoked when an XML object arrives from the server.

Constructor for the XMLSocket object

Availability

Flash Player 5.

Usage

```
new XMLSocket()
```

Parameters

None.

Returns

Nothing.

Description

Constructor; creates a new XMLSocket object. The XMLSocket object is not initially connected to any server. You must call the `XMLSocket.connect` method to connect the object to a server.

Example

```
myXMLSocket = new XMLSocket();
```

See also

`XMLSocket.connect`

XMLSocket.close

Availability

Flash Player 5.

Usage

```
myXMLSocket.close()
```

Parameters

None.

Returns

Nothing.

Description

Method; closes the connection specified by XMLSocket object.

See also

`XMLSocket.connect`

XMLSocket.connect

Availability

Flash Player 5.

Usage

```
myXMLSocket.connect(host, port)
```

Parameters

host A fully qualified DNS domain name, or an IP address in the form *aaa.bbb.ccc.ddd*. You can also specify `null` to connect to the host server on which the movie resides.

port The TCP port number on the host used to establish a connection. The port number must be 1024 or higher.

Returns

Nothing.

Description

Method; establishes a connection to the specified Internet host using the specified TCP port (must be 1024 or higher), and returns `true` or `false` depending on whether a connection is successfully established. If you don't know the port number of your Internet host machine, contact your network administrator. If the Flash Netscape plug-in or ActiveX control is being used, the host specified in the parameter must have the same subdomain as the host from where the movie was downloaded.

If you specify `null` for the *host* parameter, the host contacted will be the host where the movie calling `XMLSocket.connect` resides. For example, if the movie was downloaded from `http://www.yoursite.com`, specifying `null` for the host parameter is the same as entering the IP address for `www.yoursite.com`.

If `XMLSocket.connect` returns a value of `true`, the initial stage of the connection process is successful; later, the `XMLSocket.onConnect` method is invoked to determine whether the final connection succeeded or failed. If `XMLSocket.connect` returns `false`, a connection could not be established.

Example

The following example uses `XMLSocket.connect` to connect to the host where the movie resides, and uses `trace` to display the return value indicating the success or failure of the connection.

```
function myOnConnect(success) {
    if (success) {
        trace ("Connection succeeded!")
    } else {
        trace ("Connection failed!")
    }
}
socket = new XMLSocket()
socket.onConnect = myOnConnect
if (!socket.connect(null, 2000)) {
    trace ("Connection failed!")
}
```

See also

function, `XMLSocket.onConnect`

XMLSocket.onClose

Availability

Flash Player 5.

Usage

myXMLSocket.onClose()

Parameters

None.

Returns

Nothing.

Description

Method; a callback function that is invoked only when an open connection is closed by the server. The default implementation of this method performs no actions. To override the default implementation, you must assign a function containing your own actions.

See also

function, `XMLSocket.onConnect`

XMLSocket.onConnect

Availability

Flash Player 5.

Usage

myXMLSocket.onConnect(*success*)

Parameters

success A Boolean value indicating whether a socket connection was successfully established (true or false).

Returns

Nothing.

Description

Method; a callback function invoked by the Flash Player when a connection request initiated through the `XMLSocket.connect` method has succeeded or failed. If the connection succeeded, the *success* parameter is true; otherwise the *success* parameter is false.

The default implementation of this method performs no actions. To override the default implementation, you must assign a function containing your own actions.

Example

The following example illustrates the process of specifying a replacement function for the `onConnect` method in a simple chat application.

The function controls which screen users are taken to, depending on whether a connection is successfully established. If the connection is successfully made, users are taken to the main chat screen on the frame labeled `startChat`. If the connection is not successful, users go to a screen with troubleshooting information on the frame labeled `connectionFailed`.

```
function myOnConnect(success) {
    if (success) {
        gotoAndPlay("startChat")
    } else {
        gotoAndStop("connectionFailed")
    }
}
```

After creating the `XMLSocket` object using the constructor method, the script installs the `onConnect` method using the assignment operator:

```
socket = new XMLSocket()
socket.onConnect = myOnConnect
```

Finally, the connection is initiated. If `connect` returns false, the movie is sent directly to the frame labeled `connectionFailed`, and `onConnect` is never invoked. If `connect` returns true, the movie jumps to a frame labeled `waitForConnection`, which is the “Please wait” screen. The movie remains on the `waitForConnection` frame until the `onConnect` handler is invoked, which happens at some point in the future depending on network latency.

```
if (!socket.connect(null, 2000)) {
    gotoAndStop("connectionFailed")
} else {
    gotoAndStop("waitForConnection")
}
```

See also

function, `XMLSocket.connect`

XMLSocket.onData

Availability

Flash Player 5.

Usage

```
XMLSocket.onData()
```

Parameters

None.

Returns

Nothing.

Description

Event handler; invoked when an XML message has been downloaded from the server, terminated by a zero byte.

By default, the `XMLSocket.onData` method invokes the `XMLSocket.onXML` method. If you override `XMLSocket.onData` with your own behavior, `XMLSocket.onXML` will no longer be called unless you call it in your implementation of `XMLSocket.onData`.

```
XMLSocket.prototype.onData = function (src) {  
    this.onXML(new XML(src));  
}
```

In the above example, the *src* parameter is a string containing XML text downloaded from the server. The zero byte terminator is not included in the string.

`XMLSocket.onData` can be overridden to intercept the XML text without parsing it.

XMLSocket.onXML

Availability

Flash Player 5.

Usage

```
myXMLSocket.onXML(object)
```

Parameter

object An instance of the XML object containing a parsed XML document received from a server.

Returns

Nothing.

Description

Method; a callback function invoked by the Flash Player when the specified XML object containing an XML document arrives over an open XMLSocket connection. An XMLSocket connection may be used to transfer an unlimited number of XML documents between the client and the server. Each document is terminated with a 0 (zero) byte. When the Flash Player receives the 0 byte, it parses all of the XML received since the previous 0 byte, or since the connection was established if this is the first message received. Each batch of parsed XML is treated as a single XML document and passed to the `onXML` method.

The default implementation of this method performs no actions. To override the default implementation, you must assign a function containing actions that you define.

Example

The following function overrides the default implementation of the `onXML` method in a simple chat application. The function `myOnXML` instructs the chat application to recognize a single XML element, `MESSAGE`, in the following format.

```
<MESSAGE USER="John" TEXT="Hello, my name is John!" />.
```

The `onXML` handler must first be installed in the `XMLSocket` object as follows:

```
socket.onXML = myOnXML;
```

The function `displayMessage` is assumed to be a user-defined function that displays the message received by the user.

```
function myOnXML(doc) {  
    var e = doc.firstChild;  
    if (e != null && e.nodeName == "MESSAGE") {  
        displayMessage(e.attributes.user, e.attributes.text);  
    }  
}
```

See also

`function`

XMLSocket.send

Availability

Flash Player 5.

Usage

```
myXMLSocket.send(object)
```

Parameters

object An XML object or other data to transmit to the server.

Returns

Nothing.

Description

Method; converts the XML object or data specified in the *object* parameter to a string and transmits it to the server, followed by a zero byte. If *object* is an XML object, the string is the XML textual representation of the XML object. The send operation is asynchronous; it returns immediately, but the data may be transmitted at a later time. The `XMLSocket.send` method does not return a value indicating whether the data was successfully transmitted.

If the *myXMLSocket* object is not connected to the server (using `XMLSocket.connect`), the `XMLSocket.send` operation will fail.

Example

The following example illustrates how you could specify a user name and password to send the XML object `myXML` to the server:

```
var myXML = new XML();  
var myLogin = myXML.createElement("login");  
myLogin.attributes.username = usernameTextField;  
myLogin.attributes.password = passwordTextField;  
myXML.appendChild(myLogin);  
myXMLSocket.send(myXML);
```

See also

`XMLSocket.connect`