

# **Extending ADOBE® FLASH® PROFESSIONAL**



## **Legal notices**

For legal notices, see [http://help.adobe.com/en\\_US/legalnotices/index.html](http://help.adobe.com/en_US/legalnotices/index.html).

# Contents

## **Chapter 1: Introduction**

Working with the JavaScript API .....	1
What's new in the JavaScript API .....	4
JavaScript API objects .....	10
Sample implementations .....	16

## **Chapter 2: Top-Level Functions and Methods**

Top-level summary .....	18
activate() .....	18
alert() .....	19
configureTool() .....	19
confirm() .....	20
deactivate() .....	21
keyDown() .....	21
keyUp() .....	22
mouseDoubleClick() .....	23
mouseDown() .....	23
mouseMove() .....	24
mouseUp() .....	25
notifySettingsChanged() .....	25
prompt() .....	26
setCursor() .....	27

## **Chapter 3: actionsPanel object**

actionsPanel summary .....	28
actionsPanel.getClassForObject() - dropped .....	28
actionsPanel.getScriptAssistMode() - dropped .....	29
actionsPanel.getSelectedText() .....	30
actionsPanel.getText() .....	30
actionsPanel.hasSelection() .....	31
actionsPanel.replaceSelectedText() .....	31
actionsPanel.setScriptAssistMode() - dropped .....	32
actionsPanel.setSelection() .....	33
actionsPanel.setText() .....	34

## **Chapter 4: BitmapInstance object**

bitmapInstance summary .....	35
bitmapInstance.getBits() .....	35
bitmapInstance.hPixels .....	36
bitmapInstance.setBits() .....	37
bitmapInstance.vPixels .....	37

**Chapter 5: BitmapItem object**

bitmapItem.summary .....	39
bitmapItem.allowSmoothing .....	40
bitmapItem.compressionType .....	40
bitmapItem.exportToFile() .....	41
bitmapItem.fileLastModifiedDate .....	41
bitmapItem.hasValidAlphaLayer .....	42
bitmapItem.hPixels .....	42
bitmapItem.lastModifiedDate .....	43
bitmapItem.originalCompressionType .....	43
bitmapItem.quality .....	44
bitmapItem.sourceFileExists .....	44
bitmapItem.sourceFilesCurrent .....	45
bitmapItem.sourceFilePath .....	45
bitmapItem.useDeblocking .....	46
bitmapItem.useImportedJPEGQuality .....	46
bitmapItem.vPixels .....	46

**Chapter 6: CompiledClipInstance object**

compiledClipInstance.summary .....	48
compiledClipInstance.accName .....	49
compiledClipInstance.actionScript - dropped .....	49
compiledClipInstance.backgroundColor .....	50
compiledClipInstance.blendMode .....	50
compiledClipInstance.brightness .....	51
compiledClipInstance.cacheAsBitmap .....	51
compiledClipInstance.colorAlphaAmount .....	52
compiledClipInstance.colorAlphaPercent .....	52
compiledClipInstance.colorBlueAmount .....	53
compiledClipInstance.colorBluePercent .....	53
compiledClipInstance.colorGreenAmount .....	54
compiledClipInstance.colorGreenPercent .....	54
compiledClipInstance.colorMode .....	55
compiledClipInstance.colorRedAmount .....	55
compiledClipInstance.colorRedPercent .....	55
compiledClipInstance.description .....	56
compiledClipInstance.filters .....	56
compiledClipInstance.forceSimple .....	57
compiledClipInstance.shortcut .....	57
compiledClipInstance.silent .....	58
compiledClipInstance.tabIndex .....	58
compiledClipInstance.tintColor .....	59
compiledClipInstance.tintColorPercent .....	59
compiledClipInstance.useBackgroundColor .....	60
compiledClipInstance.visible .....	60

**Chapter 7: compilerErrors object**

compilerErrors summary .....	61
compilerErrors.clear() .....	61
compilerErrors.save() .....	62

**Chapter 8: ComponentInstance object**

componentInstance summary .....	63
componentInstance.parameters .....	63

**Chapter 9: componentsPanel object**

componentsPanel summary .....	64
componentsPanel.addItemToDocument() .....	64
componentsPanel.reload() .....	65

**Chapter 10: Contour object**

contour summary .....	66
contour.fill .....	66
contour.getHalfEdge() .....	67
contour.interior .....	68
contour.orientation .....	68

**Chapter 11: Document object**

document summary .....	70
document.accName .....	77
document.addDataToDocument() .....	78
document.addDataToSelection() .....	78
document.addFilter() .....	79
document.addItem() .....	80
document.add.NewLine() .....	80
document.addNewOval() .....	81
document.addNewPrimitiveOval() .....	82
document.addNewPrimitiveRectangle() .....	83
document.addNewPublishProfile() .....	84
document.addNewRectangle() .....	84
document.addNewScene() .....	85
document.addNewText() .....	86
document.align() .....	87
document.allowScreens() - dropped .....	87
document.arrange() .....	88
document.as3AutoDeclare .....	89
document.as3Dialect .....	89
document.as3ExportFrame .....	89
document.as3StrictMode .....	90
document.as3WarningsMode .....	90
document.asVersion .....	91
document.autoLabel .....	92
document.backgroundColor .....	92
document.breakApart() .....	92

document.canEditSymbol()	93
document.canRevert()	93
document.canTestMovie()	94
document.canTestScene()	95
document.changeFilterOrder()	95
document.clipCopy()	96
document.clipCut()	97
document.clipPaste()	97
document.close()	98
document.convertLinesToFills()	98
document.convertSelectionToBitmap()	99
document.convertToSymbol()	99
document.crop()	100
document.currentPublishProfile	101
document.currentTimeline	101
document.debugMovie()	102
document.deleteEnvelope()	102
document.deletePublishProfile()	103
document.deleteScene()	104
document.deleteSelection()	104
document.description	105
document.disableAllFilters()	105
document.disableFilter()	106
document.disableOtherFilters()	106
document.distribute()	107
document.distributeToKeyframes()	108
document.distributeToLayers()	108
document.docClass	109
document.documentHasData()	109
document.duplicatePublishProfile()	110
document.duplicateScene()	110
document.duplicateSelection()	111
document.editScene()	111
document.enableAllFilters()	112
document.enableFilter()	112
document.enterEditMode()	113
document.exitEditMode()	114
document.exportInstanceToLibrary()	114
document.exportInstanceToPNGSequence()	115
document.exportPNG()	115
document.exportPublishProfile()	116
document.exportPublishProfileString()	117
document.exportSWF()	117
document.exportVideo()	118
document.externalLibraryPath	119
document.forceSimple	119

document.frameRate .....	120
document.getAlignToDocument() .....	120
document.getBlendMode() .....	121
document.getCustomFill() .....	121
document.getCustomStroke() .....	122
document.getDataFromDocument() .....	123
document.getElementProperty() .....	124
document.getElementTextAttr() .....	124
document.getFilters() .....	125
document.getMetadata() .....	126
document.getMobileSettings() .....	126
document.getPlayerVersion() .....	127
document.getPublishDocumentData() .....	128
document.getSelectionRect() .....	129
document.getSWFPathFromProfile() .....	129
document.getTelemetryForSwf() .....	130
document.getTextString() .....	130
document.getTimeline() .....	131
document.getTransformationPoint() .....	132
document.group() .....	133
document.height .....	133
document.id .....	134
document.importFile() .....	134
document.importPublishProfile() .....	135
document.importPublishProfileString() .....	135
document.importSWF() - dropped .....	136
document.intersect() .....	136
document.library .....	137
document.libraryPath .....	137
document.livePreview .....	138
document.loadCuepointXML() - dropped .....	138
document.match() .....	139
document.mouseClick() .....	140
document.mouseDblClk() .....	140
document.moveSelectedBezierPointsBy() .....	141
document.moveSelectionBy() .....	142
document.name .....	142
document.optimizeCurves() .....	143
document.path .....	143
document.pathURI .....	144
document.publish() .....	144
document.publishProfiles .....	145
document.punch() .....	145
document.removeAllFilters() .....	146
document.removeDataFromDocument() .....	146
document.removeDataFromSelection() .....	147

document.removeFilter()	148
document.renamePublishProfile()	148
document.renameScene()	149
document.reorderScene()	149
document.resetOvalObject()	150
document.resetRectangleObject()	150
document.resetTransformation()	151
document.revert()	151
document.rotate3DSelection()	152
document.rotateSelection()	152
document.save()	153
document.saveAndCompact() - dropped	154
document.saveAsCopy()	154
document.scaleSelection()	155
document.screenOutline - dropped	156
document.selectAll()	156
document.selection	157
document.selectNone()	158
document.setAlignToDocument()	159
document.setBlendMode()	160
document.setCustomFill()	160
document.setCustomStroke()	161
document.setProperty()	161
document.setTextAttr()	162
document.setFillColor()	163
document.setFilterProperty()	164
document.setFilters()	164
document.setInstanceAlpha()	165
document.setInstanceBrightness()	166
document.setInstanceTint()	166
document.setMetadata()	167
document.setMobileSettings()	168
document.setOvalObjectProperty()	169
document.setPlayerVersion()	170
document.setPublishDocumentData()	170
document.setRectangleObjectProperty()	171
document.setSelectionBounds()	172
document.setSelectionRect()	173
document.setStageVanishingPoint()	174
document.setStageViewAngle()	174
document.setStroke()	175
document.setStrokeColor()	175
document.setStrokeSize()	176
document.setStrokeStyle()	176
document.setTextRectangle()	177
document.setTextSelection()	178

document.setTextString()	178
document.setTransformationPoint()	179
document.silent	180
document.skewSelection()	180
document.smoothSelection()	181
document.sourcePath	181
document.space()	182
document.straightenSelection()	183
document.swapElement()	183
document.swapStrokeAndFill()	184
document.swfJPEGQuality	184
document.testMovie()	185
document.testScene()	185
document.timelines	186
document.traceBitmap()	186
document.translate3DCenter()	187
document.translate3DSelection()	187
document.transformSelection()	188
document.unGroup()	189
document.union()	189
document.unlockAllElements()	190
document.viewMatrix	190
document.width	191
document.xmlPanel()	191
document.zoomFactor	192

**Chapter 12: drawingLayer object**

drawingLayersSummary	193
drawingLayer.beginDraw()	193
drawingLayer.beginFrame()	194
drawingLayer.cubicCurveTo()	194
drawingLayer.curveTo()	195
drawingLayer.drawPath()	196
drawingLayer.endDraw()	196
drawingLayer.endFrame()	197
drawingLayer.lineTo()	197
drawingLayer.moveTo()	198
drawingLayer newPath()	198
drawingLayer.setColor()	199
drawingLayer.setFill()	200
drawingLayer.setStroke()	200

**Chapter 13: Edge object**

edge summary	201
edge.cubicSegmentIndex	201
edge.getControl()	202
edge.getHalfEdge()	202

edge.id .....	203
edge.isLine .....	203
edge.setControl() .....	204
edge.splitEdge() .....	204
edge.stroke .....	205

**Chapter 14: Element object**

element summary .....	206
element.depth .....	207
element.elementType .....	208
element.getPersistentData() .....	208
element.getPublishPersistentData() .....	209
element.getTransformationPoint() .....	210
element.hasPersistentData() .....	211
element.height .....	211
element.layer .....	212
element.left .....	212
element.locked .....	213
element.matrix .....	213
element.name .....	213
element.removePersistentData() .....	214
element.rotation .....	214
element.scaleX .....	215
element.scaleY .....	215
element.selected .....	216
element.setPersistentData() .....	216
element.setPublishPersistentData() .....	217
element.setTransformationPoint() .....	218
element.skewX .....	219
element.skewY .....	219
element.top .....	220
element.transformX .....	220
element.transformY .....	220
element.width .....	221
element.x .....	221
element.y .....	222

**Chapter 15: Fill object**

fill summary .....	223
fill.bitmapIsClipped .....	223
fill.bitmapPath .....	224
fill.color .....	224
fill.colorArray .....	225
fill.focalPoint .....	225
fill.linearRGB .....	226
fill.matrix .....	226
fill.overflow .....	227

fill.posArray .....	227
fill.style .....	228

**Chapter 16: Filter object**

filter.summary .....	229
filter.angle .....	230
filter.blurX .....	230
filter.blurY .....	231
filter.brightness .....	231
filter.color .....	232
filter.contrast .....	232
filter.distance .....	233
filter.enabled .....	234
filter.hideObject .....	234
filter.highlightColor .....	235
filter.hue .....	235
filter.inner .....	236
filter.knockout .....	236
filter.name .....	237
filter.quality .....	237
filter.saturation .....	238
filter.shadowColor .....	239
filter.strength .....	239
filter.type .....	240

**Chapter 17: flash object (fl)**

fl.summary .....	241
fl.actionsPanel .....	244
fl.addEventListener() .....	244
fl.as3PackagePaths .....	245
fl/browseForFileURL() .....	246
fl/browseForFolderURL() .....	247
fl.clearPublishCache() .....	248
fl/clipCopyString() .....	248
fl/closeAll() .....	249
fl/closeAllPlayerDocuments() .....	250
fl/closeDocument() .....	250
fl/compilerErrors .....	251
fl/componentsPanel .....	251
fl/configDirectory .....	252
fl/configURI .....	252
fl/contactSensitiveSelection .....	252
fl/copyLibraryItem() .....	253
fl/createDocument() .....	253
fl/createNewDocList .....	254
fl/createNewDocListType .....	254
fl/createNewTemplateList .....	255

fl.documents .....	255
fl.drawingLayer .....	256
fl.exportPublishProfileString() .....	256
fl.externalLibraryPath .....	257
fl.fileExists() .....	257
fl.findDocumentDOM() .....	258
fl.findDocumentIndex() .....	258
fl.findObjectInDocByName() .....	259
fl.findObjectInDocByType() .....	260
fl.flexSDKPath .....	262
fl.getAppMemoryInfo() .....	262
fl.getDocumentDOM() .....	263
fl.getThemeColor() .....	263
fl.getThemeColorParameters() .....	264
fl.getThemeFontInfo() .....	266
fl.getSwfPanel() .....	266
fl.installedPlayers .....	267
fl.isFontInstalled() .....	268
fl.languageCode .....	268
fl.libraryPath .....	269
fl.mapPlayerURL() .....	269
fl.Math .....	270
fl.mruRecentFileList .....	270
fl.mruRecentFileListType .....	271
fl.objectDrawingMode .....	271
fl.openDocument() .....	272
fl.openScript() .....	272
fl.outputPanel .....	273
fl.packagePaths - dropped .....	273
fl.presetPanel .....	274
fl.publishCacheDiskSizeMax .....	274
fl.publishCacheEnabled .....	274
fl.publishCacheMemoryEntrySizeLimit .....	275
fl.publishCacheMemorySizeMax .....	275
fl.publishDocument() .....	276
fl.quit() .....	276
fl.reloadEffects() - dropped .....	277
fl.reloadTools() .....	278
fl.removeEventListener() .....	278
fl.resetAS3PackagePaths() .....	279
fl.resetPackagePaths() - dropped .....	279
fl.revertDocument() .....	280
fl.runScript() .....	280
fl.saveAll() .....	281
fl.saveDocument() .....	282
fl.saveDocumentAs() .....	283

fl.scriptURI .....	283
fl.selectElement() .....	284
fl.selectTool() .....	285
fl.setActiveWindow() .....	286
fl.showIdleMessage() .....	286
fl.setPrefBoolean() .....	287
fl.sourcePath .....	288
fl.spriteSheetExporter .....	288
fl.swfPanels .....	288
fl.toggleBreakpoint() .....	289
fl.tools .....	290
fl.trace() .....	290
fl.version .....	291
fl.xmlPanel() .....	291
fl.xmlPanelFromString() .....	292
fl.xmlui .....	292

#### **Chapter 18: FLfile object**

FLfile summary .....	293
FLfile.copy() .....	294
FLfile.createFolder() .....	295
FLfile.exists() .....	296
FLfile.getAttributes() .....	296
FLfile.getCreationDate() .....	297
FLfile.getCreationDateObj() .....	298
FLfile.getModificationDate() .....	299
FLfile.getModificationDateObj() .....	300
FLfile.getSize() .....	300
FLfile.listFolder() .....	301
FLfile.platformPathToURI() .....	302
FLfile.read() .....	302
FLfile.remove() .....	303
FLfile.setAttributes() .....	304
FLfile.uriToPlatformPath() .....	305
FLfile.write() .....	306

#### **Chapter 19: folderItem object**

folderItem summary .....	308
--------------------------	-----

#### **Chapter 20: fontItem object**

fontItem summary .....	309
fontItem.bitmap .....	309
fontItem.bold .....	310
fontItem.embeddedCharacters .....	310
fontItem.embedRanges .....	311
fontItem.embedVariantGlyphs .....	311
fontItem.font .....	313

fontItem.isDefineFont4Symbol .....	313
fontItem.italic .....	314
fontItem.size .....	314

**Chapter 21: Frame object**

frame.summary .....	315
frame.convertMotionObjectTo2D() .....	317
frame.convertMotionObjectTo3D() .....	317
frame.convertToFrameByFrameAnimation() .....	318
frame.actionScript .....	318
frame.duration .....	319
frame.elements .....	319
frame.getCustomEase() .....	320
frame.getMotionObjectXML() .....	320
frame.getSoundEnvelope() .....	321
frame.getSoundEnvelopeLimits() .....	322
frame.hasCustomEase .....	322
frame.hasMotionPath() .....	323
frame.is3DMotionObject() .....	323
frame.isEmpty() .....	324
frame.isMotionObject() .....	324
frame.labelType .....	325
frame.motionTweenOrientToPath .....	325
frame.motionTweenRotate .....	326
frame.motionTweenRotateTimes .....	326
frame.motionTweenScale .....	326
frame.motionTweenSnap .....	327
frame.motionTweenSync .....	327
frame.name .....	327
frame.selectMotionPath() .....	328
frame.setCustomEase() .....	329
frame.setMotionObjectDuration() .....	329
frame.setMotionObjectXML() .....	330
frame.setSoundEnvelope() .....	331
frame.setSoundEnvelopeLimits() .....	332
frame.shapeTweenBlend .....	332
frame.soundEffect .....	332
frame.soundLibraryItem .....	333
frame.soundLoop .....	333
frame.soundLoopMode .....	334
frame.soundName .....	334
frame.soundSync .....	334
frame.startFrame .....	335
frame.tweenEasing .....	335
frame.tweenInstanceName .....	336

frame.TweenType .....	336
frame.useSingleEaseCurve .....	337

**Chapter 22: HalfEdge object**

halfEdge summary .....	338
halfEdge.getEdge() .....	338
halfEdge.getNext() .....	339
halfEdge.getOppositeHalfEdge() .....	339
halfEdge.getPrev() .....	340
halfEdge.getVertex() .....	340
halfEdge.id .....	341

**Chapter 23: Instance object**

instance summary .....	342
instance.instanceType .....	342
instance.libraryItem .....	343

**Chapter 24: Item object**

item summary .....	344
item.addData() .....	345
item.getData() .....	345
item.getPublishData() .....	346
item.hasData() .....	347
item.itemType .....	348
item.linkageBaseClass .....	348
item.linkageClassName .....	349
item.linkageExportForAS .....	349
item.linkageExportForRS .....	350
item.linkageExportInFirstFrame .....	350
item.linkageIdentifier .....	351
item.linkageImportForRS .....	351
item.linkageURL .....	351
item.name .....	352
item.removeData() .....	352
item.setPublishData() .....	353

**Chapter 25: Layer object**

layer summary .....	355
layer.animationType .....	355
layer.color .....	356
layer.frameCount .....	356
layer.frames .....	357
layer.height .....	357
layer.layerType .....	358
layer.locked .....	358
layer.name .....	358
layer.outline .....	359

layer.parentLayer .....	359
layer.visible .....	360

**Chapter 26: library object**

library summary .....	361
library.addItemToDocument() .....	362
library.addNewItem() .....	362
library.deleteItem() .....	363
library.duplicateItem() .....	364
library.editItem() .....	364
library.findItemIndex() .....	365
library.getItemProperty() .....	365
library.getItemType() .....	366
library.getSelectedItems() .....	366
library.itemExists() .....	367
library.items .....	367
library.moveToFolder() .....	368
library.newFolder() .....	368
library.renameItem() .....	369
library.selectAll() .....	369
library.selectItem() .....	370
library.selectNone() .....	371
library.setItemProperty() .....	371
library.unusedItems .....	372
library.updateItem() .....	372

**Chapter 27: Math object**

Math summary .....	374
Math.concatMatrix() .....	374
Math.invertMatrix() .....	375
Math.pointDistance() .....	375
Math.transformPoint() .....	376

**Chapter 28: Matrix object**

matrix summary .....	377
matrix.a .....	377
matrix.b .....	378
matrix.c .....	378
matrix.d .....	379
matrix.tx .....	379
matrix_ty .....	380

**Chapter 29: outputPanel object**

outputPanel summary .....	381
outputPanel.clear() .....	381
outputPanel.save() .....	382
outputPanel.trace() .....	382

<b>Chapter 30: Oval object</b>	
OvalObject summary .....	384
OvalObject.closePath .....	384
OvalObject.endAngle .....	385
OvalObject.innerRadius .....	385
OvalObject.startAngle .....	386
<b>Chapter 31: Parameter object</b>	
parameter summary .....	387
parameter.category .....	387
parameter.insertItem() .....	388
parameter.listIndex .....	388
parameter.name .....	389
parameter.removeItem() .....	389
parameter.value .....	390
parameter.valueType .....	391
parameter.verbose .....	391
<b>Chapter 32: Path object</b>	
path summary .....	392
path.addCubicCurve() .....	392
path.addCurve() .....	393
path.addPoint() .....	394
path.clear() .....	394
path.close() .....	395
path.makeShape() .....	395
path.newContour() .....	396
path.nPts .....	397
<b>Chapter 33: presetItem object</b>	
presetItem summary .....	398
presetItem.isDefault .....	398
presetItem.isFolder .....	399
presetItem.level .....	399
presetItem.name .....	400
presetItem.open .....	400
presetItem.path .....	400
<b>Chapter 34: presetPanel object</b>	
presetPanel summary .....	402
presetPanel.addItem() .....	403
presetPanel.applyPreset() .....	403
presetPanel.deleteFolder() .....	404
presetPanel.deleteItem() .....	405
presetPanel.expandFolder() .....	405
presetPanel.exportItem() .....	406
presetPanel.findIndex() .....	407
presetPanel.getSelectedItems() .....	408

presetPanel.importItem()	408
presetPanel.items	409
presetPanel.moveToFolder()	409
presetPanel.newFolder()	410
presetPanel.renameItem()	411
presetPanel.selectItem()	411
<b>Chapter 35: Rectangle object</b>	
RectangleObject summary	413
RectangleObject.bottomLeftRadius	413
RectangleObject.bottomRightRadius	414
RectangleObject.lockFlag	414
RectangleObject.topLeftRadius	415
RectangleObject.topRightRadius	415
<b>Chapter 36: Shape object</b>	
shape summary	416
shape.beginEdit()	417
shape.contours	417
shape.deleteEdge()	418
shape.edges	418
shape.endEdit()	418
shape.getCubicSegmentPoints()	419
shape.isDrawingObject	420
shape.isFloating	420
shape.isGroup	420
shape.isOvalObject	421
shape.isRectangleObject	421
shape.members	422
shape.numCubicSegments	422
shape.vertices	423
<b>Chapter 37: SoundItem object</b>	
soundItem summary	424
soundItem.bitRate	425
soundItem.bits	425
soundItem.compressionType	426
soundItem.convertStereoToMono	426
soundItem.exportToFile()	427
soundItem.fileLastModifiedDate	427
soundItem.lastModifiedDate	428
soundItem.originalCompressionType	428
soundItem.quality	429
soundItem.sampleRate	429
soundItem.sourceFileExists	430
soundItem.sourceFilesCurrent	430

soundItem.sourceFilePath .....	431
soundItem.useImportedMP3Quality .....	431

**Chapter 38: SpriteSheetExporter object**

SpriteSheetExporter summary .....	433
SpriteSheetExporter.addBitmap() .....	434
SpriteSheetExporter.addSymbol() .....	435
SpriteSheetExporter.algorithm .....	435
SpriteSheetExporter.allowRotate .....	435
SpriteSheetExporter.allowTrimming .....	436
SpriteSheetExporter.app .....	436
SpriteSheetExporter.autoSize .....	437
SpriteSheetExporter.beginExport() .....	437
SpriteSheetExporter.borderPadding .....	437
SpriteSheetExporter.canBorderPad .....	438
SpriteSheetExporter.canRotate .....	438
SpriteSheetExporter.canTrim .....	438
SpriteSheetExporter.canShapePad .....	439
SpriteSheetExporter.canStackDuplicateFrames .....	439
SpriteSheetExporter.changeSymbol() .....	440
SpriteSheetExporter.exportSpriteSheet() .....	440
SpriteSheetExporter.format .....	441
SpriteSheetExporter.image .....	441
SpriteSheetExporter.layoutFormat .....	441
SpriteSheetExporter.maxSheetHeight .....	442
SpriteSheetExporter.maxSheetWidth .....	442
SpriteSheetExporter.overflowed .....	443
SpriteSheetExporter.removeBitmap() .....	443
SpriteSheetExporter.removeSymbol() .....	443
SpriteSheetExporter.shapePadding .....	444
SpriteSheetExporter.sheetHeight .....	444
SpriteSheetExporter.sheetWidth .....	445
SpriteSheetExporter.stackDuplicateFrames .....	445
SpriteSheetExporter.version .....	445

**Chapter 39: Stroke object**

stroke summary .....	447
stroke.breakAtCorners .....	448
stroke.capType .....	448
stroke.color .....	449
stroke.curve .....	449
stroke.dash1 .....	450
stroke.dash2 .....	450
stroke.density .....	451
stroke.dotSize .....	451
stroke.dotSpace .....	452
stroke.hatchThickness .....	452

stroke.jiggle .....	453
stroke.joinType .....	453
stroke.length .....	453
stroke.miterLimit .....	454
stroke.pattern .....	454
stroke.rotate .....	455
stroke.scaleType .....	455
stroke.shapeFill .....	456
stroke.space .....	456
stroke.strokeHinting .....	457
stroke.style .....	457
stroke.thickness .....	458
stroke.variation .....	458
stroke.waveHeight .....	459
stroke.waveLength .....	459

**Chapter 40: swfPanel object**

swfPanel summary .....	461
swfPanel.call() .....	461
swfPanel.dpiScaleFactorX .....	463
swfPanel.dpiScaleFactorY .....	464
swfPanel.name .....	464
swfPanel.path .....	465
swfPanel.reload() .....	465
swfPanel.setFocus() .....	465

**Chapter 41: SymbolInstance object**

symbolInstance summary .....	467
symbolInstance.accName .....	468
symbolInstance.actionScript - dropped .....	469
symbolInstance.backgroundColor .....	469
symbolInstance.bitmapRenderMode .....	470
symbolInstance.blendMode .....	470
symbolInstance.brightness .....	471
symbolInstance.buttonTracking .....	471
symbolInstance.cacheAsBitmap .....	472
symbolInstance.colorAlphaAmount .....	472
symbolInstance.colorAlphaPercent .....	472
symbolInstance.colorBlueAmount .....	473
symbolInstance.colorBluePercent .....	473
symbolInstance.colorGreenAmount .....	474
symbolInstance.colorGreenPercent .....	474
symbolInstance.colorMode .....	474
symbolInstance.colorRedAmount .....	475
symbolInstance.colorRedPercent .....	475
symbolInstance.description .....	475
symbolInstance.filters .....	476

symbolInstance.firstFrame .....	476
symbolInstance.forceSimple .....	477
symbolInstance.is3D .....	477
symbolInstance.loop .....	478
symbolInstance.shortcut .....	478
symbolInstance.silent .....	479
symbolInstance.symbolType .....	479
symbolInstance.tabIndex .....	479
symbolInstance.tintColor .....	480
symbolInstance.tintColorPercent .....	480
symbolInstance.useBackgroundColor .....	481
symbolInstance.visible .....	481

**Chapter 42: SymbolItem object**

symbolItem summary .....	482
symbolItem.convertToCompiledClip() .....	483
symbolItem.exportSWC() .....	483
symbolItem.exportSWF() .....	484
symbolItem.exportToLibrary() .....	484
symbolItem.exportToPNGSequence() .....	485
symbolItem.lastModifiedDate .....	485
symbolItem.scalingGrid .....	486
symbolItem.scalingGridRect .....	486
symbolItem.sourceAutoUpdate .....	487
symbolItem.sourceFilePath .....	487
symbolItem.sourceLibraryName .....	487
symbolItem.symbolType .....	488
symbolItem.timeline .....	488

**Chapter 43: Text object**

text summary .....	489
text.accName .....	490
text.antiAliasSharpness .....	491
text.antiAliasThickness .....	491
text.autoExpand .....	492
text.border .....	492
text.description .....	492
text.embeddedCharacters .....	493
text.embedRanges .....	493
text.embedVariantGlyphs .....	494
text.filters .....	495
text.fontRenderingMode .....	495
text.getTextAttr() .....	496
text.getTextString() .....	497
text.length .....	498
text.lineType .....	498
text.maxCharacters .....	498

text.orientation .....	499
text.renderAsHTML .....	499
text.scrollable .....	500
text.selectable .....	500
text.selectionEnd .....	500
text.selectionStart .....	501
text.setTextAttr() .....	501
text.setTextString() .....	502
text.shortcut .....	503
text.silent .....	504
text.tabIndex .....	504
text.textRuns .....	504
text.textType .....	505
text.useDeviceFonts .....	505
text.variableName .....	506
<b>Chapter 44: TextAttrs object</b>	
textAttrs summary .....	507
textAttrs.aliasText .....	508
textAttrs.alignment .....	508
textAttrs.autoKern .....	508
textAttrs.bold .....	509
textAttrs.characterPosition .....	509
textAttrs.characterSpacing .....	510
textAttrs.face .....	510
textAttrs.fillColor .....	510
textAttrs.indent .....	511
textAttrs.italic .....	511
textAttrs.leftMargin .....	512
textAttrs.letterSpacing .....	512
textAttrs.lineSpacing .....	512
textAttrs.rightMargin .....	513
textAttrs.rotation .....	513
textAttrs.size .....	514
textAttrs.target .....	514
textAttrs.url .....	514
<b>Chapter 45: TextRun object</b>	
textRun summary .....	516
textRun.textAttrs .....	516
textRun.characters .....	516
<b>Chapter 46: Timeline object</b>	
timeline summary .....	518
timeline.addMotionGuide() .....	520
timeline.addNewLayer() .....	521
timeline.clearFrames() .....	521

timeline.clearKeyframes()	522
timeline.convertToBlankKeyframes()	523
timeline.convertToKeyframes()	523
timeline.copyFrames()	524
timeline.copyLayers()	525
timeline.copyMotion()	525
timeline.copyMotionAsAS3()	526
timeline.createMotionObject()	527
timeline.createMotionTween()	527
timeline.currentFrame	528
timeline.currentLayer	528
timeline.cutFrames()	529
timeline.cutLayers()	530
timeline.deleteLayer()	530
timeline.duplicateLayers()	531
timeline.expandFolder()	532
timeline.findLayerIndex()	532
timeline.frameCount	533
timeline.getBounds()	533
timeline.getFrameProperty()	534
timeline.getGuidelines()	535
timeline.getLayerProperty()	535
timeline.getSelectedFrames()	536
timeline.getSelectedLayers()	537
timeline.insertBlankKeyframe()	537
timeline.insertFrames()	538
timeline.insertKeyframe()	539
timeline.layerCount	540
timeline.layers	540
timeline.libraryItem	541
timeline.name	541
timeline.pasteFrames()	541
timeline.pasteLayers()	542
timeline.pasteMotion()	543
timeline.pasteMotionSpecial()	543
timeline.removeFrames()	544
timeline.removeMotionObject()	545
timeline.reorderLayer()	546
timeline.reverseFrames()	546
timeline.selectAllFrames()	547
timeline.setFrameProperty()	547
timeline.setGuidelines()	548
timeline.setLayerProperty()	549
timeline.setSelectedFrames()	550
timeline.setSelectedLayers()	551
timeline.showLayerMasking()	551

timeline.startPlayback()	552
timeline.stopPlayback()	552

**Chapter 47: ToolObj object**

toolObj summary	554
toolObj.depth	555
toolObj.enablePIControl()	555
toolObj.iconID	556
toolObj.position	556
toolObj.setIcon()	557
toolObj.setMenuString()	557
toolObj.setOptionsFile()	558
toolObj.setPI()	559
toolObj.setToolName()	560
toolObj.setToolTip()	560
toolObj.showPIControl()	561
toolObj.showTransformHandles()	562

**Chapter 48: Tools object**

tools summary	563
tools.activeTool	564
tools.altIsDown	564
tools.constrainPoint()	564
tools.ctrlIsDown	565
tools.getKeyDown()	565
tools.mouselsDown	566
tools.penDownLoc	566
tools.penLoc	567
tools.setCursor()	567
tools.shiftIsDown	568
tools.snapPoint()	568
tools.toolObjs	569

**Chapter 49: Tween Object**

Tween object Summary	570
Tween.getColorTransform()	571
Tween.getFilters()	572
Tween.getGeometricTransform()	572
Tween.getShape()	573
Tween.duration	573
Tween.startFrame	574
Tween.tweenType	574

**Chapter 50: Vertex object**

vertex summary	575
vertex.getHalfEdge()	575
vertex.setLocation()	576

vertex.x .....	576
vertex.y .....	577

**Chapter 51: Videolitem object**

videolitem summary .....	578
videolitem.exportToFLV() .....	578
videolitem.fileLastModifiedDate .....	579
videolitem.lastModifiedDate .....	579
videolitem.sourceFileExists .....	580
videolitem.sourceFilesCurrent .....	580
videolitem.sourceFilePath .....	581
videolitem.videoType .....	581

**Chapter 52: XMLUI object**

xmlui summary .....	583
xmlui.accept() .....	583
xmlui.cancel() .....	584
xmlui.get() .....	584
xmlui.getControlItemElement() .....	585
xmlui.getEnabled() .....	586
xmlui.getVisible() .....	586
xmlui.set() .....	587
xmlui.setControlItemElement() .....	587
xmlui.setControlItemElements() .....	588
xmlui.setEnabled() .....	589
xmlui.setVisible() .....	589

**Chapter 53: C-Level Extensibility**

About extensibility .....	591
Integrating C functions .....	591
Data types .....	596
The C-level API .....	597

# Chapter 1: Introduction

## Working with the JavaScript API

As a user of Adobe® Flash® Professional CC, you may be familiar with Adobe® ActionScript®, which lets you create scripts that execute at run time in Adobe® Flash® Player. The Flash JavaScript application programming interface (JavaScript API or JSAPI) described in this document is a complementary programming tool that lets you create scripts that run in the Flash authoring environment.

This document describes the objects, methods, and properties available in the JavaScript API. It assumes that you know how to use the documented commands when working in the authoring environment. If you have a question about what a particular command does, use other documents in Flash Help, such as *Using Flash*, to find that information.

This document also assumes that you are familiar with JavaScript or ActionScript syntax and with basic programming concepts such as functions, parameters, and data types.

The Flash JavaScript API lets you write scripts to perform several actions in the Flash authoring environment (that is, while a user has the Flash program open). This functionality is different from the ActionScript language, which lets you write scripts to perform actions in the Flash Player environment (that is, while a SWF file is playing). This functionality is also different from JavaScript commands that you might use in pages displayed in a web browser.

Using the JavaScript API, you can write Flash application scripts to help streamline the authoring process. For example, you can write scripts to automate repetitive tasks or add custom tools to the Tools panel.

The Flash JavaScript API is designed to resemble the Adobe® Dreamweaver® and Adobe® Fireworks® JavaScript API (which were designed based on the Netscape JavaScript API). The Flash JavaScript API is based on a Document Object Model (DOM), which allows Flash documents to be accessed using JavaScript objects. The Flash JavaScript API includes all elements of the Netscape JavaScript API, plus the Flash DOM. These added objects and their methods and properties are described in this document. You can use any of the elements of the native JavaScript language in a Flash script, but only elements that make sense in the context of a Flash document have an effect.

The JavaScript API also contains methods that let you implement extensibility using a combination of JavaScript and custom C code. For more information, see “[C-Level Extensibility](#)” on page 591.

The JavaScript interpreter in Flash is the Mozilla SpiderMonkey engine, version 1.8, which is available on the web at [www.mozilla.org/js/spidermonkey/](http://www.mozilla.org/js/spidermonkey/). SpiderMonkey is one of the two reference implementations of the JavaScript language developed by Mozilla.org. It is the same engine that is embedded in the Mozilla browser.

SpiderMonkey implements the core JavaScript language as defined in the ECMAScript (ECMA-262) edition 3 language specification and it is fully compliant with the specification. Only the browser-specific host objects, which are not part of the ECMA-262 specification, are not supported. Similarly, many JavaScript reference guides distinguish between core JavaScript and client-side (browser-related) JavaScript. Only core JavaScript applies to the Flash JavaScript interpreter.

### Creating JSFL files

You can use Adobe Flash Professional or your preferred text editor to write and edit Flash JavaScript (JSFL) files. If you use Flash, these files have a .jsfl extension by default. To write a script, select File > New > Flash JavaScript File.

You can also create a JSFL file by selecting commands in the History panel. Then click the Save button in the History panel or select Save As Command from the panel menu. The command (JSFL) file is saved in the Commands folder (see “[Saving JSFL files](#)” on page 2). You can then open the file and edit it the same as any other script file.

The History panel provides some other useful options as well. You can copy selected commands to the Clipboard, and you can view JavaScript commands that are generated while you are working in Flash.

**To copy commands from the History panel to the clipboard:**

- 1 Select one or more commands in the History panel.
- 2 Do one of the following:
  - Click the Copy button.
  - Select Copy Steps from the panel menu.

**To view JavaScript commands in the History panel:**

- Select View > JavaScript in Panel from the panel menu.

## Saving JSFL files

You can have JSFL scripts available within the Flash authoring environment by storing them in one of several folders within the Configuration folder. By default, the Configuration folder is in the following location:

- Windows® 7™:  
*boot drive\Users\username\AppData\Local\Adobe\Flash CC\language\Configuration\*
- Windows® Vista™:  
*boot drive\Users\username\Local Settings\Application Data\Adobe\Flash CC\language\Configuration\*
- Mac OS® X:  
*Macintosh HD/Users/username/Library/Application Support/Adobe/Flash CC/language/Configuration/*

To determine the location of the Configuration folder, use `f1.configDirectory` or `f1.configURI`, as shown in the following example:

```
// store directory to a variable
var configDir = f1.configDirectory;
// display directory in the Output panel
f1.trace(f1.configDirectory);
```

Within the Configuration folder, the following folders can contain scripts that you can access in the authoring environment: Behaviors (to support the user interface for behaviors); Commands (for scripts that appear on the Commands menu); JavaScript (for scripts used by Script Assist to populate the user interface controls); Tools (for extensible tools in the Tools panel); and WindowSWF (for panels that appear in the Windows menu). This document focuses on scripts used for commands and tools.

If you edit a script in the Commands folder, the new script is immediately available in Flash. If you edit a script for an extensible tool, close and restart Flash, or else use the `f1.reloadTools()` command. However, if you used a script to add an extensible tool to the Tools panel and you then edit the script, either remove and then add the tool to the Tools panel again, or else close and restart Flash for the revised tool to be available.

There are two locations where you can store command and tool files so they can be accessed in the authoring environment.

- For scripts that appear as items in the Commands menu, save the JSFL file in the Commands folder in the following location:

Operating system	Location
Windows 7	<i>boot drive\Users\username\AppData\Local\Adobe\Flash CC\language\Configuration\Commands</i>
Windows Vista	<i>boot drive\Users\username\Local Settings\Application Data\Adobe\Flash CC\language\Configuration\Commands</i>
Mac OS X	Macintosh HD/Users/ <i>userName</i> /Library/Application Support/Adobe/Flash CC/ <i>language</i> /Configuration/Commands

- For scripts that appear as extensible tools in the Tools panel, save the JSFL file in the Tools folder in the following location:

Operating system	Location
Windows 7	<i>boot drive\Users\username\AppData\Local\Adobe\Flash CC\language\Configuration\Tools</i>
Windows Vista	<i>boot drive\Users\username\Local Settings\Application Data\Adobe\Flash CC\language\Configuration\Tools</i>
Mac OS X	Macintosh HD/Users/ <i>userName</i> /Library/Application Support/Adobe/Flash CC/ <i>language</i> /Configuration/Tools

If a JSFL file has other files that go with it, such as XML files, store them in the same directory as the JSFL file.

## Running scripts

There are several ways to run scripts. The most common ways are explained in this section.

### To run a script that you are currently viewing or editing:

- Right-click (Command-click on the Macintosh) and choose Run Script.
- Click the Run Script icon on the Script window toolbar.

This option lets you run a script before you have saved it. This option also lets you run a script even if no FLA files are open.

### To run a script that is in the Commands folder, do one of the following:

- From the authoring environment, select Commands > *Script Name*.
- Use a keyboard shortcut that you have assigned to the script. To assign a keyboard shortcut, use Edit > Keyboard Shortcuts and select Drawing Menu Commands from the Commands pop-up menu. Expand the Commands node in the menu tree to view a list of available scripts.

### To run a command script that is not in the Commands folder, do one of the following:

- From the authoring environment, select Commands > Run Command, and then select the script to run.
- From within a script, use the `f1.runScript()` command.
- From the file system, double-click the script file.

### To add a tool implemented in a JSFL file to the Tools panel:

- Copy the JSFL file for the tool and any other associated files to the Tools folder (see “[Saving JSFL files](#)” on page 2).

- 2 Select Edit > Customize Tools Panel (Windows) or Flash > Customize Tools Panel (Macintosh).
- 3 Add the tool to the list of available tools.
- 4 Click OK.

You can add individual JavaScript API commands to ActionScript files by using the `MMExecute()` function, which is documented in the *ActionScript 3.0 Language and Components Reference*. However, the `MMExecute()` function has an effect only when it is used in the context of a custom user interface element, such as a component Property inspector, or a SWF panel within the authoring environment. Even if called from ActionScript, JavaScript API commands have no effect in Flash Player or outside the authoring environment.

**To issue a command from an ActionScript script:**

- Use the following syntax (you can concatenate several commands into one string):

```
MMExecute(Javascript command string);
```

You can also run a script from the command line.

**To run a script from the command line on Windows:**

- Use the following syntax (add path information as required):

```
"flash.exe" myTestFile.jsfl [-AlwaysRunJSFL]
```

Use the `-AlwaysRunJSFL` option to bypass the dialog box that prompts you to confirm script execution.

**To run a script from the “Terminal” application on the Macintosh, use either of the following:**

- Use the following osascript syntax (add path information as required):

```
osascript -e 'tell application "flash" to open alias "Mac OS X:Users:user:myTestFile.jsfl" '
```

The `osascript` command can also run AppleScript in a file. For example, you could include the following text in a file named `myScript`:

```
tell application "flash"
open alias "Mac OS X:Users:user:myTestFile.jsfl"
end tell
```

Then, to run the script, you would use this command:

```
osascript myScript
```

- Use the `flashpro` command:

```
/Applications/Adobe\ Flash\ CC/flashpro.app/Contents/MacOS/flashpro <path of the jsfl file>
```

## What's new in the JavaScript API

The following section lists new objects, methods, and properties in Flash CC. It also lists changes in Flash CS5, CS5.5, and CS6.

If you have not used the JavaScript API before, you might want to skip this section and go directly to “[JavaScript API objects](#)” on page 10).

## New methods and properties in Flash CC

The following methods and properties for existing objects are new in Flash Professional CC:

- [Tween Object](#)
  - Tween.getColorTransform()
  - Tween.getFilters()
  - Tween.getGeometricTransform()
  - Tween.getShape()
  - Tween.duration
  - Tween.startFrame
  - Tween.tweenType
- [CompiledClipInstance object](#)
  - CompiledClipInstance.backgroundColor
  - CompiledClipInstance.blendMode
  - CompiledClipInstance.brightness
  - CompiledClipInstance.cacheAsBitmap
  - CompiledClipInstance.colorAlphaAmount
  - CompiledClipInstance.colorAlphaPercent
  - CompiledClipInstance.colorBlueAmount
  - CompiledClipInstance.colorBluePercent
  - CompiledClipInstance.colorGreenAmount
  - CompiledClipInstance.colorGreenPercent
  - CompiledClipInstance.colorMode
  - CompiledClipInstance.colorRedAmount
  - CompiledClipInstance.colorRedPercent
  - CompiledClipInstance.filters
  - CompiledClipInstance.tintColor
  - CompiledClipInstance.tintPercent
  - CompiledClipInstance.useBackgroundColor
  - CompiledClipInstance.visible
- [Document object](#)
  - Document.convertSelectionToBitmap()
  - Document.distributeToKeyframes()
  - Document.exportVideo()
  - Document.getPublishDocumentData()
  - Document.getTelemetryForSwf()
  - Document.importFile() showDialog and showImporterUI properties
  - Document.setPublishDocumenData()

- [Element object](#)
  - Element.getPublishPersistentData()
  - Element.setPublishPersistentData()
- [flash object \(fl\)](#)
  - Flash.addEventListener() prePublish, postPublish, selectionChanged, and dpiChanged events
  - fl.getThemeColor()
  - fl.getThemeColorParameters()
  - fl.getThemeFontInfo()
  - fl.setPrefBoolean()
  - fl.toggleBreakPoint()
  - fl.xmlPanel()
  - fl.xmlPanelFromString()
- [Frame object](#)
  - Frame.convertToFrameByFrameAnimation()
  - Frame.getSoundEnvelope()
  - Frame.getSoundEnvelopeLimits()
  - Frame.setSoundEnvelope()
  - Frame.setSoundEnvelopeLimits()
  - Frame.isEmpty()
- [Item object](#)
  - item.getPublishData()
  - item.setPublishData()
- [library object](#)
  - library.unusedItems
- [swfPanel object](#)
  - swfPanel.dpiScaleFactorX
  - swfPanel.dpiScaleFactorY
  - swfPanel.reload()
- [SymbolInstance object](#)
  - SymbolInstance.brightness
  - SymbolInstance.tintColor
  - SymbolInstance.tintColorPercent
- [Timeline object](#)
  - getBounds()

## Dropped methods and properties in Flash CC

The following methods and properties for existing objects are dropped in Flash Professional CC:

- ActionsPanel.getClassForObject()
- ActionsPanel.getScriptAssistMode()
- ActionsPanel.setScriptAssistMode()
- CompiledClipInstance.actionscript
- fl.reloadEffects()
- fl.resetPackagePaths()
- document.activeEffect
- document.allowScreens()
- document.drawingLayer
- document.importSWF()
- document.loadCuePointXML()
- document.packagePaths
- document.saveAndCompact()
- document.screenType
- library.importEmbeddedSWF()
- SymbolInstance.actionscript

## New objects in Flash CS6

The following object is new in Flash CS6:

- [SpriteSheetExporter object](#)

## New methods and properties in Flash CS6

The following methods and properties for existing objects are new in Flash Pro CS6:

- [BitmapItem object](#)
  - bitmapItem.exportToFile() quality parameter
  - bitmapItem.hasValidAlphaLayer
  - bitmapItem.lastModifiedDate
- [Document object](#)
  - document.exportInstanceToLibrary()
  - document.exportInstanceToPNGSequence()
  - document.getSWFPathFromProfile()
  - document.saveAsCopy()
  - document.swfJPEGQuality
- [flash object \(fl\)](#)
  - fl.spriteSheetExporter

- [Layer object](#)
  - `layer.animationType`
- [Math object](#)
  - `Math.transformPoint()`
- [Shape object](#)
  - `shape.isFloating`
- [SoundItem object](#)
  - `soundItem.lastModifiedDate`
- [SymbolInstance object](#)
  - `symbolInstance.is3D`
- [SymbolItem object](#)
  - `symbolItem.exportToLibrary()`
  - `symbolItem.exportToPNGSequence()`
  - `symbolItem.lastModifiedDate`
- [Text object](#)
  - `text.filters`
- [VideoItem object](#)
  - `videoItem.lastModifiedDate`

## New methods and properties in Flash CS5 and CS5.5

The following methods and properties for existing objects are new in Flash Pro CS5:

- [Document object](#)
  - `document.debugMovie()`
  - `document.loadCuepointXML()`
- [flash object \(fl\)](#)
  - `fl.languageCode`
  - `fl.toggleBreakpoint`
- [Frame object](#)
  - `frame.convertMotionObjectTo2D()`
  - `frame.convertMotionObjectTo3D()`
  - `frame.getMotionObjectXML()`
  - `frame.hasMotionPath()`
  - `frame.isMotionObject()`
  - `frame.is3DMotionObject()`
  - `frame.selectMotionPath()`
  - `frame.setMotionObjectDuration()`
  - `frame.setMotionObjectXML()`

- frame.TweenInstanceName
- **Timeline object**
  - timeline.createMotionObject()
  - timeline.libraryItem
  - timeline.removeMotionObject()
  - timeline.startPlayback
  - timeline.stopPlayback

The following methods and properties for existing objects are new in Flash Pro CS5.5:

- [SymbolInstance object](#)
  - symbolInstance.bitmapRenderMode
  - symbolInstance.backgroundColor
  - symbolInstance.usesBackgroundColor
  - symbolInstance.visible
- [Timeline object](#)
  - timeline.copyLayers()
  - timeline.cutLayers()
  - timeline.duplicateLayers()
  - timeline.pasteLayers()
- [flash object \(fl\)](#)
  - fl.getSwfPanel()
  - fl.installedPlayers()
  - fl.publishCacheEnabled
  - fl.publishCacheDiskSizeMax
  - fl.publishCacheMemorySizeMax
  - fl.publishCacheMemoryEntrySizeLimit
  - fl.clearPublishCache()
- [swfPanel object](#)
  - swfPanel.setFocus()

## Other changes in Flash CS5 and CS5.5

The following methods and properties are updated in Flash CS5:

- fl.openScript()
- fl.publishDocument()
- fontItem.embedRanges
- fontItem.embeddedCharacters
- fontItem.embedVariantGlyphs

The following objects and method are no longer available in Flash CS5:

- Screen object
- ScreenOutline object
- document.canSaveAVersion()
- document.revertToLastVersion()
- document.saveAVersion()
- document.synchronizeWithHeadVersion()
- fl.downloadLatestVersion()
- fl.revertDocumentToLastVersion()
- fl.saveAVersionOfDocument()
- fl.synchronizeDocumentWithHeadVersion()

## JavaScript API objects

This section provides a summary of the objects available in the Flash JavaScript API and how to begin working with them. All standard JavaScript commands are also available when working with the JavaScript API.

The following table briefly describes each of the objects in the JavaScript API. The objects are listed in alphabetical order.

Object	Description
actionsPanel object	The actionsPanel object represents the currently displayed Actions panel.
BitmapInstance object	The BitmapInstance object is a subclass of the Instance object and represents a bitmap in a frame.
BitmapItem object	A BitmapItem object refers to a bitmap in the library of a document. The BitmapItem object is a subclass of the Item object.
CompiledClipInstance object	The CompiledClipInstance object is a subclass of the Instance object.
compilerErrors object	The compilerErrors object represents the Compiler Errors panel. It is a property of the flash object (fl.compilerErrors).
ComponentInstance object	The ComponentInstance object is a subclass of the SymbolInstance object and represents a component in a frame.
componentsPanel object	The componentsPanel object, which represents the Components panel, is a property of the flash object (fl.componentsPanel).
Contour object	A Contour object represents a closed path of half edges on the boundary of a shape.
Document object	The Document object represents the Stage.
drawingLayer object	The drawingLayer object is accessible from JavaScript as a child of the flash object.
Edge object	The Edge object represents an edge of a shape on the Stage.
Element object	Everything that appears on the Stage is of the type Element.
Fill object	The Fill object contains all the properties of the Fill color setting of the Tools panel or of a selected shape.

Object	Description
Filter object	The Filter object contains all the properties for all filters.
flash object (fl)	The flash object represents the Flash application.
FLfile object	The FLfile object lets you write Flash extensions that can access, modify, and remove files and folders on the local file system.
folderItem object	The folderItem object is a subclass of the Item object.
fontItem object	The fontItem object is a subclass of the Item object.
Frame object	The Frame object represents frames in the layer.
HalfEdge object	Directed side of the edge of a Shape object.
Instance object	The Instance object is a subclass of the Element object.
Item object	The Item object is an abstract base class.
Layer object	The Layer object represents a layer in the timeline.
library object	The library object represents the Library panel.
Math object	The Math object is available as a read-only property of the flash object (fl.Math).
Matrix object	The Matrix object represents a transformation matrix.
outputPanel object	The outputPanel object represents the Output panel, which displays troubleshooting information such as syntax errors. It is a property of the flash object (fl.outputPanel).
Oval object	The Oval object is a shape that is drawn using the Oval tool. To determine if an item is an Oval object, use <code>shape.isOvalObject</code> .
Parameter object	The Parameter object type is accessed from the <code>screen.parameters</code> array (which corresponds to the screen Property inspector in the Flash authoring tool) or by the <code>componentInstance.parameters</code> array (which corresponds to the component Property inspector in the authoring tool).
Path object	The Path object defines a sequence of line segments (straight, curved, or both), which you typically use when creating extensible tools.
presetItem object	The presetItem object represents an item (preset or folder) in the Motion Presets panel.
presetPanel object	The presetPanel object represents the Motion Presets panel (Window > Motion Presets). It is a property of the flash object (fl.presetPanel).
Rectangle object	The Rectangle object is a shape that is drawn using the Rectangle tool. To determine if an item is a Rectangle object, use <code>shape.isRectangleObject</code> .
Screen object	The Screen object represents a single screen in a slide or form document.
ScreenOutline object	The ScreenOutline object represents the group of screens in a slide or form document.
Shape object	The Shape object is a subclass of the Element object. The Shape object provides more precise control than the drawing APIs for manipulating or creating geometry on the Stage.
SoundItem object	The SoundItem object is a subclass of the Item object. It represents a library item used to create a sound.
Stroke object	The Stroke object contains all the settings for a stroke, including the custom settings.
swfPanel object	The swfPanel object represents a Windows SWF panel. Windows SWF panels are SWF files that implement applications you can run from the Flash authoring environment. The array of swfPanel objects is a property of the flash object (fl.swfPanels).
SymbolInstance object	The SymbolInstance object is a subclass of the Instance object and represents a symbol in a frame.

Object	Description
SymbolItem object	The SymbolItem object is a subclass of the Item object.
Text object	The Text object represents a single text item in a document.
TextAttrs object	The TextAttrs object contains all the properties of text that can be applied to a subselection. This object is a subclass of the Text object.
TextRun object	The TextRun object represents a run of characters that have attributes that match all of the properties in the TextAttrs object.
Timeline object	The Timeline object represents the Flash timeline, which can be accessed for the current document by <code>f1.getDocumentDOM().getTimeline()</code> .
ToolObj object	A ToolObj object represents an individual tool in the Tools panel.
Tools object	The Tools object is accessible from the Flash object ( <code>f1.tools</code> ).
Vertex object	The Vertex object is the part of the shape data structure that holds the coordinate data.
VideoItem object	The VideoItem object is a subclass of the Item object.
XMLUI object	The XMLUI object provides the ability to get and set properties of an XMLUI dialog box, and accept or cancel out of one.

## The Flash Document Object Model

The Flash Document Object Model (DOM) for the Flash JavaScript API consists of a set of top-level functions (see “[Top-Level Functions and Methods](#)” on page 18) and two top-level objects—the FLfile object and the flash object (`f1`). Each object is guaranteed to be available to a script because it always exists when the Flash authoring environment is open. For more information, see [FLfile object](#) and [flash object \(f1\)](#).

When referring to the flash object, you can use `flash` or `f1`. For example, to close all open FLA files, you can use either of the following statements:

```
flash.closeAll();  
f1.closeAll();
```

The flash object contains the following *child objects*:

Object	How to access
actionsPanel object	Use <code>f1.actionsPanel</code> to access the actionsPanel object. This object corresponds to the Actions panel in the Flash authoring environment.
compilerErrors object	Use <code>f1.compilerErrors</code> to access the compilerErrors object. This object corresponds to the Compiler Errors panel in the Flash authoring environment.
componentsPanel object	Use <code>f1.componentsPanel</code> to access the componentsPanel object. This object corresponds to the Components panel in the Flash authoring environment.
Document object	Use <code>f1.documents</code> to retrieve an array of all the open documents; use <code>f1.documents [index]</code> to access a particular document; use <code>f1.getDocumentDOM()</code> to access the current document (the one with focus).
drawingLayer object	Use <code>f1.drawingLayer</code> to access the drawingLayer object.
Math object	Use <code>f1.Math</code> to access the Math object.
outputPanel object	Use <code>f1.outputPanel</code> to access the outputPanel object. This object corresponds to the Output panel in the Flash authoring environment.

Object	How to access
presetPanel object	Use <code>f1.presetPanel</code> to access the presetPanel object. This object corresponds to the Motion Presets panel (Window > Motion Presets).
swfPanel object	Use <code>f1.swfPanels</code> to access an array of swfPanel objects. These objects correspond to Window SWF panels.
Tools object	Use <code>f1.tools</code> to access an array of Tools objects.
XMLUI object	Use <code>f1.xmlui</code> to access an XML User Interface (XMLUI) object. The XMLUI object provides the ability to get and set properties of an XMLUI dialog box.

## The Document object

An important property of the top-level flash object is the `fl.documents` property. This property contains an array of Document objects, each of which represents one of the FLA files currently open in the authoring environment. The properties of each Document object represent most of the elements that a FLA file can contain. Therefore, a large portion of the DOM is composed of child objects and properties of the Document object. For more information, see [Document object](#).

To refer to the first open document, for example, use the statement `flash.documents[0]` or `f1.documents[0]`. The first document is the first Flash document that was opened during the current session in the authoring environment. When the first opened document is closed, the indexes of the other open documents are decremented.

To find a particular document's index, use `flash.findDocumentIndex(nameOfDocument)` or `f1.findDocumentIndex(nameOfDocument)`. See [f1.findDocumentIndex\(\)](#).

To access the document that is currently focused, use the statement `flash.getDocumentDOM()` or `f1.getDocumentDOM()`. See [f1.getDocumentDOM\(\)](#). The latter is the syntax used in most of the examples in this document.

To find a particular document in the `f1.documents` array, iterate through the array and test each document for its `document.name` property. See [f1.documents](#) and [document.name](#).

All the objects in the DOM that aren't listed in the previous table (see "[The Flash Document Object Model](#)" on page 12) are accessed from the Document object. For example, to access the library of a document, you use the `document.library` property, which retrieves a library object:

```
f1.getDocumentDOM().library
```

To access the array of items in the library, you use the `library.items` property; each element in the array is an Item object:

```
f1.getDocumentDOM().library.items
```

To access a particular item in the library, you specify a member of the `library.items` array:

```
f1.getDocumentDOM().library.items[0]
```

In other words, the library object is a child of the Document object, and the Item object is a child of the library object. For more information, see [document.library](#), [library object](#), [library.items](#), and [Item object](#).

## Specifying the target of an action

Unless otherwise specified, methods affect the current focus or selection. For example, the following script doubles the size of the current selection because no particular object is specified:

```
f1.getDocumentDOM().scaleSelection(2, 2);
```

In some cases, you might want an action to specifically target the currently selected item in the Flash document. To do this, use the array that the `document.selection` property contains (see [document.selection](#)). The first element in the array represents the currently selected item, as shown in the following example:

```
var accDescription = fl.getDocumentDOM().selection[0].description;
```

The following script doubles the size of the first element on the Stage that is stored in the element array, instead of the current selection:

```
var element = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];
if (element) {
    element.width = element.width*2;
    element.height = element.height*2;
}
```

You can also do something such as loop through all the elements on the Stage and increase the width and height by a specified amount, as shown in the following example:

```
var elementArray =
    fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements;
for (var i=0; i < elementArray.length; i++) {
    var offset = 10;
    elementArray[i].width += offset;
    elementArray[i].height += offset;
}
```

## Summary of the DOM structure

The following list displays the DOM structure in outline format. Numbers at the beginning of each line represent the level of an object. For example, an object preceded by “03” is a child of next highest “02” object, which, in turn, is a child of the next highest “01” object.

In some cases, an object is available by specifying a property of its parent object. For example, the `document.timelines` property contains an array of Timeline objects. These properties are noted in the following outline.

Some objects are subclasses of other objects, rather than being children of other objects. An object that is a subclass of another object has methods and/or properties of its own in addition to the methods and properties of the parent object (the superclass). Subclasses share the same level in the hierarchy as their superclass. For example, the Item object is a superclass of the BitmapItem object. These relationships are illustrated in the following outline:

```
01 Top-Level Functions and Methods
01 FLfile object
01 flash object (f1)
    02 compilerErrors object
    02 componentsPanel object
    02 Document object (f1.documents array)
        03 Filter object
        03 Matrix object
        03 Fill object
        03 Stroke object
        03 library object
            04 Item object (library.items array)
            04 BitmapItem object (subclass of Item object)
            04 folderItem object (subclass of Item object)
            04 fontItem object (subclass of Item object)
            04 SoundItem object (subclass of Item object)
            04 SymbolItem object (subclass of Item object)
            04 VideoItem object (subclass of Item object)
        03 Timeline object (document.timelines array)
            04 Layer object (timeline.layers array)
                05 Frame object (layer.frames array)
                    06 Element object (frame.elements array)
                    07 Matrix object (element.matrix)
                06 Instance object (abstract class, subclass of Element object)
                06 BitmapInstance object (subclass of Instance object)
                06 CompiledClipInstance object (subclass of Instance object)
                06 ComponentInstance object (subclass of SymbolInstance object)
                    07 Parameter object (componentInstance.parameters array)
                06 SymbolInstance object (subclass of Instance object)
                06 Text object (subclass of Element object)
                    07 TextRun object (text.textRuns array)
                        08 TextAttrs object (textRun.textAttrs array)
                06 Shape object (subclass of Element object)
                    07 Oval object
                    07 Rectangle object
                    07 Contour object (shape.contours array)
                        08 HalfEdge object
                    09 Vertex object
                    09 Edge object
```

```
07 Edge object (shape.edges array)
08 HalfEdge object
09 Vertex object
09 Edge object
07 Vertex object (shape.vertices array)
08 HalfEdge object
09 Vertex object
09 Edge object
05 Parameter object (screen.parameters array)
02 drawingLayer object
03 Path object
04 Contour object
02 Math object
02 outputPanel object
02 presetPanel object
03 presetItem object (presetPanel.items array)
02 swfPanel object
02 Tools object (fl.tools array)
03 ToolObj object (tools.toolObjs array)
02 XMLUI object
```

## Sample implementations

Several sample JSFL implementations are available for Adobe Flash Professional CS5 and CS5.5. You can review and install these files to familiarize yourself with the JavaScript API. The samples are in a folder named Samples/ExtendingFlash within the Samples.zip file located at [www.adobe.com/go/learn\\_fl\\_samples](http://www.adobe.com/go/learn_fl_samples).

### Sample Shape command

A sample JavaScript API script named Shape.jsfl is located in the ExtendingFlash/Shape folder (see “Sample implementations” above). This script displays information about the contours of the shape in the Output panel.

#### To install and run the Shape script:

- 1 Copy the Shape.jsfl file to the Configuration/Commands folder (see “[Saving JSFL files](#)” on page 2).
- 2 In a Flash document (FLA file), select a shape object.
- 3 Select Commands > Shape to run the script.

### Sample get and set filters command

A sample JavaScript API script named filtersGetSet.jsfl is located in the ExtendingFlash/filtersGetSet folder (see “Sample implementations” above). This script adds filters to a selected object and displays information about the filters being added in the Output panel.

#### To install and run the filtersGetSet script:

- 1 Copy the filtersGetSet.jsfl file to the Configuration/Commands folder (see “[Saving JSFL files](#)” on page 2).
- 2 In a Flash document (FLA file), select a text, movie clip, or button object.
- 3 Select Commands > filtersGetSet to run the script.

## Sample PolyStar tool

A sample JavaScript API script named PolyStar.jsfl is located in the ExtendingFlash/PolyStar folder (see “Sample implementations” above).

The PolyStar.jsfl replicates the PolyStar tool that can be found in the Flash Tools panel. The script demonstrates how to build the PolyStar tool using the JavaScript API and includes detailed comments describing what the code is doing. Read this file to gain a better understanding of how the JavaScript API can be used. You should also read the PolyStar.xml file in the Tools directory to learn more about how to build your own tool.

## Sample Trace Bitmap panel

A set of files named TraceBitmap.fla and TraceBitmap.swf are located in the ExtendingFlash/TraceBitmapPanel folder (see “Sample implementations” above). These files illustrate how to design and build a panel to control the functions of Flash. They also show the use of the `MMExecute()` function to call JavaScript commands from an ActionScript script.

### To run the TraceBitmap sample:

- 1 If Flash is running, exit from Flash.
- 2 Copy the TraceBitmap.swf file to the WindowSWF folder, which is a subdirectory of the Configuration folder (see “[Saving JSFL files](#)” on page 2). For example, on Windows XP, the folder is in *boot drive\Documents and Settings\user\Local Settings\Application Data\Adobe\Flash CS5\language\Configuration\WindowSWF*.
- 3 Start Flash.
- 4 Create or open a Flash document (FLA file), and import a bitmap or JPEG image into the file.  
You can use the flower.jpg file provided in the TraceBitmapPanel folder or another image of your choice.
- 5 With the imported image selected, select Window > Other Panels > TraceBitmap.
- 6 Click Submit.

The image is converted into a group of shapes.

## Sample DLL

A sample DLL implementation is located in the ExtendingFlash/dllSampleComputeSum folder (see “Sample implementations” above). For more information about building DLLs, see “[C-Level Extensibility](#)” on page 591.

# Chapter 2: Top-Level Functions and Methods

## Top-level summary

### About this section

This section describes the top-level functions and methods that are available when you use the Adobe Flash JavaScript application programming interface (JavaScript API). For information about where to store JavaScript API files, see “[Saving JSFL files](#)” on page 2.

### Global methods

The following methods can be called from any JavaScript API script:

```
alert()
confirm()
prompt()
```

### Extensible tools

The following functions are available in scripts that create extensible tools:

```
activate()
configureTool()
deactivate()
keyDown()
keyUp()
mouseDoubleClick()
mouseDown()
mouseMove()
mouseUp()
notifySettingsChanged()
setCursor()
```

## activate()

### Availability

Flash MX 2004.

### Usage

```
function activate() {
    // statements
}
```

### Parameters

None.

**Returns**

Nothing.

**Description**

Function; called when the extensible tool becomes active (that is, when the tool is selected in the Tools panel). Use this function to perform any initialization tasks the tool requires.

**Example**

The following example sets the value of `tools.activeTool` when the extensible tool is selected in the Tools panel:

```
function activate() {  
    var theTool = fl.tools.activeTool  
}
```

**See also**

[tools.activeTool](#)

## alert()

**Availability**

Flash MX 2004.

**Usage**

```
alert ( alertText )
```

**Parameters**

**alertText** A string that specifies the message you want to display in the Alert dialog box.

**Returns**

Nothing.

**Description**

Method; displays a string in a modal Alert dialog box, along with an OK button.

**Example**

The following example displays the message “Process Complete” in an Alert dialog box:

```
alert("Process Complete");
```

**See also**

[confirm\(\)](#), [prompt\(\)](#)

## configureTool()

**Availability**

Flash MX 2004.

**Usage**

```
function configureTool() {  
    // statements  
}
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Function; called when Flash opens and the extensible tool is loaded into the Tools panel. Use this function to set any information Flash needs to know about the tool.

**Example**

The following examples show two possible implementations of this function:

```
function configureTool() {  
    theTool = fl.tools.activeTool;  
    theTool.setToolName("myTool");  
    theTool.setIcon("myTool.png");  
    theTool.setMenuString("My Tool's menu string");  
    theTool.setToolTipText("my tool's tool tip");  
    theTool.setOptionsFile( "mtTool.xml" );  
}  
  
function configureTool() {  
    theTool = fl.tools.activeTool;  
    theTool.setToolName("ellipse");  
    theTool.setIcon("Ellipse.png");  
    theTool.setMenuString("Ellipse");  
    theTool.setToolTipText("Ellipse");  
    theTool.showTransformHandles( true );  
}
```

## confirm()

**Availability**

Flash 8.

**Usage**

```
confirm ( strAlert )
```

**Parameters**

**strAlert** A string that specifies the message you want to display in the Alert dialog box.

**Returns**

A Boolean value: `true` if the user clicks OK; `false` if the user clicks Cancel.

**Description**

Method; displays a string in a modal Alert dialog box, along with OK and Cancel buttons.

**Note:** If there are no documents (FLA files) open, this method fails with an error condition.

**Example**

The following example displays the message "Sort data?" in an Alert dialog box:

```
confirm("Sort data?");
```

**See also**

[alert\(\)](#), [prompt\(\)](#)

## deactivate()

**Availability**

Flash MX 2004.

**Usage**

```
function deactivate() {  
    // statements  
}
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Function; called when the extensible tool becomes inactive (that is, when the active tool changes from this tool to another one). Use this function to perform any cleanup the tool needs.

**Example**

The following example displays a message in the Output panel when the tool becomes inactive:

```
function deactivate() {  
    fl.trace( "Tool is no longer active" );  
}
```

## keyDown()

**Availability**

Flash MX 2004.

**Usage**

```
function keyDown() {  
    // statements  
}
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Function; called when the extensible tool is active and the user presses a key. The script should call [tools.getKeyDown\(\)](#) to determine which key was pressed.

**Example**

The following example displays information about which key was pressed when the extensible tool is active and the user presses a key.

```
function keyDown() {  
    fl.trace("key " + fl.tools.getKeyDown() + " was pressed");  
}
```

**See also**

[keyUp\(\)](#), [tools.getKeyDown\(\)](#)

## keyUp()

**Availability**

Flash MX 2004.

**Usage**

```
function keyUp() {  
    // statements  
}
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Function; called when the extensible tool is active and a key is released.

**Example**

The following example displays a message in the Output panel when the extensible tool is active and a key is released.

```
function keyUp() {  
    fl.trace("Key is released");  
}
```

**See also**[keyDown\(\)](#)

## mouseDoubleClick()

**Availability**

Flash MX 2004.

**Usage**

```
function mouseDoubleClick() {  
    // statements  
}
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Function; called when the extensible tool is active and the mouse button is double-clicked on the Stage.

**Example**

The following example displays a message in the Output panel when the extensible tool is active and the mouse button is double-clicked.

```
function mouseDoubleClick() {  
    fl.trace("Mouse was double-clicked");  
}
```

## mouseDown()

**Availability**

Flash MX 2004.

**Usage**

```
function mouseDown( [ pt ] ) {  
    // statements  
}
```

**Parameters**

**pt** A point that specifies the location of the mouse when the button is pressed. It is passed to the function when the mouse button is pressed. This parameter is optional.

**Returns**

Nothing.

**Description**

Function; called when the extensible tool is active and the mouse button is pressed while the pointer is over the Stage.

**Example**

The following examples show how this function can be used when the extensible tool is active. The first example displays a message in the Output panel that the mouse button was pressed. The second example displays the *x* and *y* coordinates of the mouse's location when the button was pressed.

```
function mouseDown() {  
    fl.trace("Mouse button has been pressed");  
}  
function mouseDown(pt) {  
    fl.trace("x = "+ pt.x+ " :: y = "+pt.y);  
}
```

## mouseMove()

**Availability**

Flash MX 2004.

**Usage**

```
function mouseMove( [ pt ] ) {  
    // statements  
}
```

**Parameters**

**pt** A point that specifies the current location of the mouse. It is passed to the function whenever the mouse moves, which tracks the mouse location. If the Stage is in edit or edit-in-place mode, the point coordinates are relative to the object being edited. Otherwise, the point coordinates are relative to the Stage. This parameter is optional.

**Returns**

Nothing.

**Description**

Function; called whenever the extensible tool is active and the mouse moves over a specified point on the Stage. The mouse button can be down or up.

**Example**

The following examples show how this function can be used. The first example displays a message in the Output panel that the mouse is being moved. The second example displays the *x* and *y* coordinates of the mouse's location as it moves.

```
function mouseMove() {
    fl.trace("moving");
}

function mouseMove(pt) {
    fl.trace("x = " + pt.x + " :: y = " + pt.y);
}
```

## mouseUp()

### Availability

Flash MX 2004.

### Usage

```
function mouseUp() {
    // statements
}
```

### Parameters

None.

### Returns

Nothing.

### Description

Function; called whenever the extensible tool is active and the mouse button is released after being pressed on the Stage.

### Example

The following example displays a message in the Output panel when the extensible tool is active and the mouse button is released.

```
function mouseUp() {
    fl.trace("mouse is up");
}
```

## notifySettingsChanged()

### Availability

Flash MX 2004.

### Usage

```
function notifySettingsChanged() {
    // statements
}
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Function; called when the extensible tool is active and the user changes its options in the Property inspector. You can use the `tools.activeTool` property to query the current values of the options (see [tools.activeTool](#)).

**Example**

The following example displays a message in the Output panel when the extensible tool is active and the user changes its options in the Property inspector.

```
function notifySettingsChanged() {  
    var theTool = fl.tools.activeTool;  
    var newValue = theTool.myProp;  
}
```

## **prompt()**

**Availability**

Flash MX 2004.

**Usage**

```
prompt(promptMsg [,text])
```

**Parameters**

**promptMsg** A string to display in the Prompt dialog box (limited to 256 characters in Mac OS X).

**text** An optional string to display as a default value for the text field.

**Returns**

The string the user typed if the user clicks OK; `null` if the user clicks Cancel.

**Description**

Method; displays a prompt and optional text in a modal Alert dialog box, along with OK and Cancel buttons.

**Example**

The following example prompts the user to enter a user name. If the user types a name and clicks OK, the name appears in the Output panel.

```
var userName = prompt("Enter user name", "Type user name here");  
fl.trace(userName);
```

**See also**

[alert\(\)](#), [confirm\(\)](#)

## setCursor()

### Availability

Flash MX 2004.

### Usage

```
function setCursor() {  
    // statements  
}
```

### Parameters

None.

### Returns

Nothing.

### Description

Function; called when the extensible tool is active and the mouse moves, to allow the script to set custom pointers. The script should call `tools.setCursor()` to specify the pointer to use. For a list that shows which pointers correspond to which integer values, see [tools.setCursor\(\)](#).

### Example

```
function setCursor() {  
    fl.tools.setCursor( 1 );  
}
```

# Chapter 3: actionsPanel object

## actionsPanel summary

### Availability

Flash CS3 Professional.

### Description

The actionsPanel object, which represents the currently displayed Actions panel, is a property of the flash object (see [fl.actionsPanel](#)).

### Method summary

The following methods can be used with the actionsPanel object:

Method	Description
<code>actionsPanel.getClassForObject() - dropped</code>	Dropped in Flash Professional CC.
<code>actionsPanel.getScriptAssistMode() - dropped</code>	Dropped in Flash Professional CC.
<code>actionsPanel.getSelectedText()</code>	Returns the text that is currently selected in the Actions panel.
<code>actionsPanel.getText()</code>	Returns the text in the Actions panel.
<code>actionsPanel.hasSelection()</code>	Specifies whether any text is currently selected in the Actions panel.
<code>actionsPanel.replaceSelectedText()</code>	Replaces the currently selected text with specified text.
<code>actionsPanel.setScriptAssistMode() - dropped</code>	Dropped in Flash Professional CC.
<code>actionsPanel.setSelection()</code>	Selects a specified set of characters in the Actions panel.
<code>actionsPanel.setText()</code>	Clears any text in the Actions panel and then adds specified text.

## actionsPanel.getClassForObject() - dropped

### Availability

Flash CS3 Professional. *Dropped in Flash Professional CC.*

### Usage

```
actionsPanel.getClassForObject (ASvariableName)
```

### Parameters

**ASvariableName** A string that represents the name of an ActionScript variable.

### Returns

A string that represents the class of which *ASvariableName* is a member.

**Description**

*Dropped in Flash Professional CC.*

Method; returns the class of the specified variable, which must be defined in the currently displayed Actions panel. In addition, the cursor or selected text in the Actions panel must be positioned after the variable definition.

**Example**

The following example displays the class assigned to the variable myVar, if the cursor is positioned after the statement var myVar:ActivityEvent; in the Actions panel.

```
// Place the following code in the Actions panel,  
// and position the cursor somewhere after the end of the line  
var myVar:ActivityEvent;  
// Place the following code in the JSFL file  
var theClass = fl.actionsPanel.getClassForObject("myVar");  
fl.trace(theClass); // traces: "ActivityEvent"
```

## actionsPanel.getScriptAssistMode() - dropped

**Availability**

Flash CS3 Professional. *Dropped in Flash Professional CC.*

**Usage**

```
actionsPanel.getScriptAssistMode()
```

**Parameters**

None.

**Returns**

A Boolean value that specifies whether Script Assist mode is enabled (`true`) or not (`false`).

**Description**

*Dropped in Flash Professional CC.*

Method; specifies whether Script Assist mode is enabled.

**Example**

The following example displays a message if Script Assist mode is not enabled.

```
mAssist = fl.actionsPanel.getScriptAssistMode();  
if (!mAssist) {  
    alert("For more guidance when writing ActionScript code, try Script Assist mode");  
}
```

**See also**

[actionsPanel.setScriptAssistMode\(\) - dropped](#)

## actionsPanel.getSelectedText()

### Availability

Flash CS3 Professional.

### Usage

```
actionsPanel.getSelectedText()
```

### Parameters

None.

### Returns

A string that contains the text that is currently selected in the Actions panel.

### Description

Method; returns the text that is currently selected in the Actions panel.

### Example

The following example displays the text that is currently selected in the Actions panel.

```
var apText = fl.actionsPanel.getSelectedText();
fl.trace(apText);
```

### See also

[actionsPanel.getText\(\)](#), [actionsPanel.hasSelection\(\)](#), [actionsPanel.replaceSelectedText\(\)](#),  
[actionsPanel.setSelection\(\)](#)

## actionsPanel.getText()

### Availability

Flash CS3 Professional.

### Usage

```
actionsPanel.getText()
```

### Parameters

None.

### Returns

A string that contains all the text in the Actions panel.

### Description

Method; returns the text in the Actions panel.

### Example

The following example displays the text that is in the Actions panel.

```
var apText = fl.actionsPanel.getText();  
fl.trace(apText);
```

**See also**

[actionsPanel.getSelectedText\(\)](#), [actionsPanel.setText\(\)](#)

## actionsPanel.hasSelection()

**Availability**

Flash CS3 Professional.

**Usage**

```
actionsPanel.hasSelection()
```

**Parameters**

None.

**Returns**

A Boolean value that specifies whether any text is selected in the Actions panel (`true`) or not (`false`).

**Description**

Method; specifies whether any text is currently selected in the Actions panel.

**Example**

The following example displays text that is currently selected in the Actions panel. If no text is selected, it displays all the text in the Actions panel.

```
if (fl.actionsPanel.hasSelection()) {  
    var apText = fl.actionsPanel.getSelectedText();  
}  
else {  
    var apText = fl.actionsPanel.getText();  
}  
fl.trace(apText);
```

**See also**

[actionsPanel.getSelectedText\(\)](#), [actionsPanel.getText\(\)](#), [actionsPanel.replaceSelectedText\(\)](#),  
[actionsPanel.setSelection\(\)](#)

## actionsPanel.replaceSelectedText()

**Availability**

Flash CS3 Professional.

**Usage**

```
actionsPanel.replaceSelectedText(replacementText)
```

**Parameters**

**replacementText** A string that represents text to replace selected text in the Actions panel.

**Returns**

A Boolean value of `true` if the Actions panel is found; `false` otherwise.

**Description**

Method; replaces the currently selected text with the text specified in `replacementText`. If `replacementText` contains more characters than the selected text, any characters following the selected text now follow `replacementText`; that is, they are not overwritten.

**Example**

The following example replaces currently selected text in the Actions panel.

```
if (fl.actionsPanel.hasSelection()) {  
    fl.actionsPanel.replaceSelectedText("// © 2006 Adobe Inc.");  
}
```

**See also**

[actionsPanel.getSelectedText\(\)](#), [actionsPanel.hasSelection\(\)](#), [actionsPanel.setSelection\(\)](#),  
[actionsPanel.setText\(\)](#)

## actionsPanel.setScriptAssistMode() - dropped

**Availability**

Flash CS3 Professional. *Dropped in Flash Professional CC.*

**Usage**

```
actionsPanel.setScriptAssistMode(bScriptAssist)
```

**Parameters**

**bScriptAssist** A Boolean value that specifies whether to enable or disable Script Assist mode.

**Returns**

A Boolean value that specifies whether Script Assist mode was enabled or disabled successfully.

**Description**

*Dropped in Flash Professional CC.*

Method; enables or disables Script Assist mode.

**Example**

The following example toggles the state of Script Assist mode.

```
fl.trace(f1.actionsPanel.getScriptAssistMode());
if (f1.actionsPanel.getScriptAssistMode()){
    f1.actionsPanel.setScriptAssistMode(false);
}
else {
    f1.actionsPanel.setScriptAssistMode(true);
}
fl.trace(f1.actionsPanel.getScriptAssistMode());
```

**See also**

[actionsPanel.getScriptAssistMode\(\) - dropped](#)

## actionsPanel.setSelection()

**Availability**

Flash CS3 Professional.

**Usage**

```
actionsPanel.setSelection(startIndex, numberOfChars)
```

**Parameters**

**startIndex** A zero-based integer that specifies the first character to be selected.

**numberOfChars** An integer that specifies how many characters to select.

**Returns**

A Boolean value that specifies whether the requested characters can be selected (`true`) or not (`false`).

**Description**

Method; selects a specified set of characters in the Actions panel.

**Example**

The following example replaces the characters “2006” in the Actions panel with the specified text.

```
// Type the following as the first line in the Actions panel
// 2006 - Addresses user request 40196
// Type the following in the JSFL file
f1.actionsPanel.setSelection(3,4);
f1.actionsPanel.replaceSelectedText("// Last updated: 2007");
```

**See also**

[actionsPanel.getSelectedText\(\), actionsPanel.hasSelection\(\)](#),
[actionsPanel.replaceSelectedText\(\)](#)

## actionsPanel.setText()

### Availability

Flash CS3 Professional.

### Usage

```
actionsPanel.setText(replacementText)
```

### Parameters

**replacementText** A string that represents text to place in the Actions panel.

### Returns

A Boolean value of `true` if the specified text was placed in the Actions panel; `false` otherwise.

### Description

Method; clears any text in the Actions panel and then adds the text specified in *replacementText*.

### Example

The following example replaces any text currently in the Actions panel with the specified text.

```
f1.actionsPanel.setText("// Deleted this code - no longer needed");
```

### See also

[actionsPanel.getText\(\)](#), [actionsPanel.replaceSelectedText\(\)](#)

# Chapter 4: BitmapInstance object

## bitmapInstance summary

**Inheritance** [Element object](#) > [Instance object](#) > BitmapInstance object

### Availability

Flash MX 2004.

### Description

The BitmapInstance object is a subclass of the Instance object and represents a bitmap in a frame (see [Instance object](#)).

### Method summary

In addition to the [Instance object](#) methods, you can use the following methods with the BitmapInstance object:

Method	Description
<code>bitmapInstance.getBits()</code>	Lets you create bitmap effects by getting the bits out of the bitmap, manipulating them, and then returning them to Flash.
<code>bitmapInstance.setBits()</code>	Sets the bits of an existing bitmap element.

### Property summary

In addition to the [Instance object](#) properties, you can use the following properties with the BitmapInstance object:

Property	Description
<code>bitmapInstance.hPixels</code>	Read-only; an integer that represents the width of the bitmap, in pixels.
<code>bitmapInstance.vPixels</code>	Read-only; an integer that represents the height of the bitmap, in pixels.

## bitmapInstance.getBits()

### Availability

Flash MX 2004.

### Usage

```
bitmapInstance.getBits()
```

### Parameters

None.

### Returns

An object that contains `width`, `height`, `depth`, `bits`, and, if the bitmap has a color table, `cTab` properties. The `bits` element is an array of bytes. The `cTab` element is an array of color values of the form "#RRGGBB". The length of the array is the length of the color table.

The byte array is meaningful only when referenced by a DLL or shared library. You typically use it only when creating an extensible tool or effect. For information on creating DLLs for use with Flash JavaScript, see “[C-Level Extensibility](#)” on page 591

### Description

Method; lets you create bitmap effects by getting the bits out of the bitmap, manipulating them, and then returning them to Flash.

### Example

The following code creates a reference to the currently selected object; tests whether the object is a bitmap; and traces the height, width, and bit depth of the bitmap:

```
var isBitmap = fl.getDocumentDOM().selection[0].instanceType;
if(isBitmap == "bitmap"){
    var bits = fl.getDocumentDOM().selection[0].getBits();
    fl.trace("height = " + bits.height);
    fl.trace("width = " + bits.width);
    fl.trace("depth = " + bits.depth);
}
```

### See also

[bitmapInstance.setBits\(\)](#)

## bitmapInstance.hPixels

### Availability

Flash MX 2004.

### Usage

`bitmapInstance.hPixels`

### Description

Read-only property; an integer that represents the width of the bitmap—that is, the number of pixels in the horizontal dimension.

### Example

The following code retrieves the width of the bitmap in pixels:

```
// Get the number of pixels in the horizontal dimension.
var bmObj = fl.getDocumentDOM().selection[0];
var isBitmap = bmObj.instanceType;
if(isBitmap == "bitmap"){
    var numHorizontalPixels = bmObj.hPixels;
}
```

### See also

[bitmapInstance.vPixels](#)

## bitmapInstance.setBits()

### Availability

Flash MX 2004.

### Usage

```
bitmapInstance.setBits(bitmap)
```

### Parameters

**bitmap** An object that contains height, width, depth, bits, and cTab properties. The height, width, and depth properties are integers. The bits property is a byte array. The cTab property is required only for bitmaps with a bit depth of 8 or less and is a string that represents a color value in the form "#RRGGBB".

**Note:** The byte array is meaningful only when referenced by an external library. You typically use it only when creating an extensible tool or effect.

### Returns

Nothing.

### Description

Method; sets the bits of an existing bitmap element. This lets you create bitmap effects by getting the bits out of the bitmap, manipulating them, and then returning the bitmap to Flash.

### Example

The following code tests whether the current selection is a bitmap and then sets the height of the bitmap to 150 pixels:

```
var isBitmap = fl.getDocumentDOM().selection[0].instanceType;
if(isBitmap == "bitmap"){
    var bits = fl.getDocumentDOM().selection[0].getBits();
    bits.height = 150;
    fl.getDocumentDOM().selection[0].setBits(bits);
}
```

### See also

[bitmapInstance.getBits\(\)](#)

## bitmapInstance.vPixels

### Availability

Flash MX 2004.

### Usage

```
bitmapInstance.vPixels
```

### Description

Read-only property; an integer that represents the height of the bitmap—that is, the number of pixels in the vertical dimension.

**Example**

The following code gets the height of the bitmap in pixels:

```
// Get the number of pixels in the vertical dimension.  
var bmObj = fl.getDocumentDOM().selection[0];  
var isBitmap = bmObj.instanceType;  
if(isBitmap == "bitmap") {  
    var numVerticalPixels = bmObj.vPixels;  
}
```

**See also**

[bitmapInstance.hPixels](#)

# Chapter 5: BitmapItem object

## bitmapItem summary

**Inheritance** [Item object](#) > BitmapItem object

### Availability

Flash MX 2004.

### Description

A BitmapItem object refers to a bitmap in the library of a document. The BitmapItem object is a subclass of the Item object (see [Item object](#)).

### Property summary

In addition to the [Item object](#) properties, the BitmapItem object has following properties:

Property	Description
<code>bitmapItem.allowSmoothing</code>	A Boolean value that specifies whether to allow smoothing of a bitmap.
<code>bitmapItem.compressionType</code>	A string that determines the type of image compression applied to the bitmap.
<code>bitmapItem.fileLastModifiedDate</code>	The number of seconds that have elapsed between January 1, 1970 and the modification date of the original file.
<code>bitmapItem.hasValidAlphaLayer</code>	A Boolean value indicating whether the bitmap has an alpha channel.
<code>bitmapItem.hPixels</code>	Specifies the width of the bitmap, in pixels.
<code>bitmapItem.lastModifiedDate</code>	The modification date of the bitmap item in the Library.
<code>bitmapItem.originalCompressionType</code>	Specifies whether the item was imported as an jpeg file.
<code>bitmapItem.sourceFileExists</code>	Specifies whether the file that was imported to the Library still exists in the location from where it was imported.
<code>bitmapItem.sourceFileIsCurrent</code>	Specifies whether the file modification date of the Library item is the same as the modification date on disk of the file that was imported.
<code>bitmapItem.sourceFilePath</code>	The path and name of the file that was imported into the Library.
<code>bitmapItem.useDeblocking</code>	Specifies whether deblocking is enabled.
<code>bitmapItem.useImportedJPEGQuality</code>	A Boolean value that specifies whether to use the default imported JPEG quality.
<code>bitmapItem.vPixels</code>	Specifies the height of the bitmap, in pixels.

### Method summary

In addition to the [Item object](#) properties, the BitmapItem object has following methods:

Method	Description
<code>bitmapItem.exportToFile()</code>	Exports the specified item to a PNG or JPG file.

## bitmapItem.allowSmoothing

### Availability

Flash MX 2004.

### Usage

```
bitmapItem.allowSmoothing
```

### Description

Property; a Boolean value that specifies whether to allow smoothing of a bitmap (`true`) or not (`false`).

### Example

The following code sets the `allowSmoothing` property of the first item in the library of the current document to `true`:

```
f1.getDocumentDOM().library.items[0].allowSmoothing = true;  
alert(f1.getDocumentDOM().library.items[0].allowSmoothing);
```

## bitmapItem.compressionType

### Availability

Flash MX 2004.

### Usage

```
bitmapItem.compressionType
```

### Description

Property; a string that determines the type of image compression applied to the bitmap. Acceptable values are "photo" or "lossless". If the value of `bitmapItem.useImportedJPEGQuality` is `false`, "photo" corresponds to JPEG with a quality from 0 to 100; if `bitmapItem.useImportedJPEGQuality` is `true`, "photo" corresponds to JPEG using the default document quality value. The value "lossless" corresponds to GIF or PNG format (see `bitmapItem.useImportedJPEGQuality`).

### Example

The following code sets the `compressionType` property of the first item in the library of the current document to "photo":

```
f1.getDocumentDOM().library.items[0].compressionType = "photo";  
alert(f1.getDocumentDOM().library.items[0].compressionType);
```

## bitmapItem.exportToFile()

### Availability

Flash CS4 Professional.

### Usage

```
bitmapItem.exportToFile(fileURI, quality)
```

### Parameters

**fileURI** A string, expressed as a file:/// URI, that specifies the path and name of the exported file.

**quality** A number, from 1-100, that determines the quality of the exported image file. A higher number indicates higher quality. The default is 80. New in Flash CS6 Professional.

### Returns

A Boolean value of `true` if the file was exported successfully; `false` otherwise.

### Description

Method; exports the specified item to a PNG or JPG file.

### Example

Assuming the first item in the Library is a bitmap item, the following code exports it as a JPG file:

```
var imageFileURL = "file:///C|/exportTest/out.jpg";
var libItem = fl.getDocumentDOM().library.items[0];
libItem.exportToFile(imageFileURL);
```

## bitmapItem.fileLastModifiedDate

### Availability

Flash CS4 Professional.

### Usage

```
bitmapItem.fileLastModifiedDate
```

### Description

Read-only property; a string containing a hexadecimal number that represents the number of seconds that have elapsed between January 1, 1970 and the modification date of the original file at the time the file was imported to the library. If the file no longer exists, this value is "00000000".

### Example

Assuming the first item in the Library is a bitmap item, the following code displays a hex number as described above.

```
var libItem = fl.getDocumentDOM().library.items[0];
fl.trace("Mod date when imported = " + libItem.fileLastModifiedDate);
```

**See also**

`bitmapItem.sourceFileExists`, `bitmapItem.sourceFileIsCurrent`, `bitmapItem.sourceFilePath`,  
`FLfile.getModificationDate()`

## bitmapItem.hasValidAlphaLayer

**Availability**

Flash CS6 Professional.

**Usage**

```
bitmapItem.hasValidAlphaLayer
```

**Description**

Read-only property; a boolean indicating if a bitmap in the library has a valid/useful alpha channel. This flag will help you decide if you should export the bitmap item as a PNG instead of a JPEG using the `bitmapItem.exportToFile()` function.

**Example**

The following code exports a library item with the proper file name extension depending on whether it has a valid alpha layer.

```
var bitmapItem = fl.getDocumentDOM().library.items[0];
var uri = fl.browseForFileURI("open");
if (bitmapItem.hasValidAlphaLayer) uri += ".png";
else uri += ".jpg";
bitmapItem.exportToFile(uri);
```

**See also**

`bitmapItem.sourceFileExists`, `bitmapItem.sourceFileIsCurrent`, `bitmapItem.sourceFilePath`,  
`FLfile.getModificationDate()`

## bitmapItem.hPixels

**Availability**

Flash CS6 Professional.

**Usage**

```
bitmapItem.hPixels
```

**Description**

Read-only property; an int that specifies the width of the bitmap, in pixels.

**Example**

The following code illustrates use of this property.

```
// get the number of pixels in the horizontal dimension.  
var bmItemObj = fl.getDocumentDOM().library.items[0];  
var numHorizontalPixels = bmItemObj.hPixels;
```

**See also**

[bitmapItem.vPixels](#)

## bitmapItem.lastModifiedDate

**Availability**

Flash Pro CS6.

**Usage**

`bitmapItem.lastModifiedDate`

**Description**

Read-only property; a hexadecimal value indicating the modification date and time of the bitmap item. This value is incremented every time the bitmap item is imported. For example, selecting the Update button from the Bitmap Properties dialog will trigger an import.

**Example**

Assuming the first item in the Library is a bitmap item, the following code displays a hex number as described above.

```
var libItem = fl.getDocumentDOM().library.items[0];  
fl.trace("Mod date when imported = " + libItem.lastModifiedDate);
```

**See also**

[bitmapItem.sourceFileExists](#), [bitmapItem.sourceFileIsCurrent](#), [bitmapItem.sourceFilePath](#),  
[FLfile.getModificationDate\(\)](#)

## bitmapItem.originalCompressionType

**Availability**

Flash CS4 Professional.

**Usage**

`bitmapItem.originalCompressionType`

**Description**

Read-only property; a string that specifies whether the specified item was imported as an jpeg file. Possible values for this property are "photo" (for jpeg files) and "lossless" (for uncompressed file types such as GIF and PNG).

**Example**

Assuming that the first item in the Library is a bitmap item, the following code displays "photo" if the file was imported into the Library as a jpeg file, or "lossless" if it was not:

```
var libItem = fl.getDocumentDOM().library.items[0];
fl.trace("Imported compression type = "+ libItem.originalCompressionType);
```

**See also**

[bitmapItem.compressionType](#)

## bitmapItem.quality

**Availability**

Flash MX 2004.

**Usage**

`bitmapItem.quality`

**Description**

Property; an integer that specifies the quality of the bitmap. To use the default document quality, specify -1; otherwise, specify an integer from 0 to 100. Available only for JPEG compression.

**Example**

The following code sets the `quality` property of the first item in the library of the current document to 65:

```
fl.getDocumentDOM().library.items[0].quality = 65;
alert(fl.getDocumentDOM().library.items[0].quality);
```

## bitmapItem.sourceFileExists

**Availability**

Flash CS4 Professional.

**Usage**

`bitmapItem.sourceFileExists`

**Description**

Read-only property; a Boolean value of `true` if the file that was imported to the Library still exists in the location from where it was imported; `false` otherwise.

**Example**

Assuming the first item in the Library is a bitmap item, the following code displays "true" if the file that was imported into the Library still exists.

```
var libItem = fl.getDocumentDOM().library.items[0];
fl.trace("sourceFileExists = "+ libItem.sourceFileExists);
```

**See also**

[bitmapItem.sourceFileIsCurrent](#),

[bitmapItem.sourceFilePath](#)

## bitmapItem.sourceFileIsCurrent

### Availability

Flash CS4 Professional.

### Usage

```
bitmapItem.sourceFileIsCurrent
```

### Description

Read-only property; a Boolean value of `true` if the file modification date of the Library item is the same as the modification date on disk of the file that was imported; `false` otherwise.

### Example

Assuming the first item in the Library is a bitmap item, the following code displays "true" if the file that was imported has not been modified on disk since it was imported:

```
var libItem = fl.getDocumentDOM().library.items[0];
fl.trace("fileIsCurrent = " + libItem.sourceFileIsCurrent);
```

### See also

[bitmapItem.fileLastModifiedDate](#), [bitmapItem.sourceFilePath](#)

## bitmapItem.sourceFilePath

### Availability

Flash CS4 Professional.

### Usage

```
bitmapItem.sourceFilePath
```

### Description

Read-only property; a string, expressed as a file:/// URI, that represents the path and name of the file that was imported into the Library.

### Example

The following example displays the name and source file path of any items in the library that are of type "bitmap":

```
for (idx in fl.getDocumentDOM().library.items) {
if (fl.getDocumentDOM().library.items[idx].itemType == "bitmap") {
    var myItem = fl.getDocumentDOM().library.items[idx];
    fl.trace(myItem.name + " source is " + myItem.sourceFilePath);
}
}
```

### See also

[bitmapItem.sourceFileExists](#)

## bitmapItem.useDeblocking

### Availability

Flash CS4 Professional.

### Usage

```
bitmapItem.useDeblocking
```

### Description

Property; a Boolean value that specifies whether deblocking is enabled (`true`) or not (`false`).

### Example

Assuming the first item in the Library is a bitmap item, the following code enables deblocking for the item:

```
var libItem = fl.getDocumentDOM().library.items[0];
libItem.useDeblocking = true;
```

## bitmapItem.useImportedJPEGQuality

### Availability

Flash MX 2004.

### Usage

```
bitmapItem.useImportedJPEGQuality
```

### Description

Property; a Boolean value that specifies whether to use the default imported JPEG quality (`true`) or not (`false`). Available only for JPEG compression.

### Example

The following code sets the `useImportedJPEGQuality` property of the first item in the library of the current document to `true`:

```
fl.getDocumentDOM().library.items[0].useImportedJPEGQuality = true;
alert(fl.getDocumentDOM().library.items[0].useImportedJPEGQuality);
```

## bitmapItem.vPixels

### Availability

Flash CS6 Professional.

### Usage

```
bitmapItem.vPixels
```

**Description**

Read-only property; an int that specifies the height of the bitmap, in pixels.

**Example**

The following code illustrates use of this property.

```
// get the number of pixels in the vertical dimension
var bmitemObj = fl.getDocumentDOM().library.items[0];
var numHorizontalPixels = bmitemObj.vPixels;
```

**See also**

[bitmapItem.hPixels](#)

# Chapter 6: CompiledClipInstance object

## compiledClipInstance summary

**Inheritance** [Element object](#) > [Instance object](#) > CompiledClipInstance object

### Availability

Flash MX 2004.

### Description

The CompiledClipInstance object is a subclass of the Instance object. It is essentially an instance of a movie clip that has been converted to a compiled clip library item (see [Instance object](#)).

### Property summary

In addition to the properties of the [Instance object](#), the CompiledClipInstance object has the following properties:

Property	Description
<code>compiledClipInstance.accName</code>	A string that is equivalent to the Name field in the Accessibility panel.
<code>compiledClipInstance.actionScript - dropped</code>	Dropped in Flash Professional CC.
<code>compiledClipInstance.backgroundColor</code>	A string that specifies the matte color when Opaque is selected.
<code>compiledClipInstance.description</code>	A string that is equivalent to the Description field in the Accessibility panel.
<code>compiledClipInstance.blendMode</code>	A string that specifies the blend mode.
<code>compiledClipInstance.brightness</code>	An int that contains the value set in the Color Effect Property Inspector for brightness.
<code>compiledClipInstance.cacheAsBitmap</code>	A boolean that indicates whether to cache bitmaps.
<code>compiledClipInstance.colorAlphaAmount</code>	An int that reduces or increases the tint and alpha values by a constant amount.
<code>compiledClipInstance.colorAlphaPercent</code>	An int that reduces the tint and alpha values by a specified percentage.
<code>compiledClipInstance.colorBlueAmount</code>	An int that reduces or increases the blue tint value by a constant amount.
<code>compiledClipInstance.colorBluePercent</code>	An int that reduces the blue tint value by a specified percentage.
<code>compiledClipInstance.colorGreenAmount</code>	An int that reduces or increases the green tint value by a constant amount.
<code>compiledClipInstance.colorGreenPercent</code>	An int that reduces the green tint value by a specified percentage.
<code>compiledClipInstance.colorMode</code>	A string that specifies the color mode, as identified in the Symbol Properties dialog.
<code>compiledClipInstance.colorRedAmount</code>	An int that reduces or increases the red tint value by a constant amount.
<code>compiledClipInstance.colorRedPercent</code>	An int that reduces the red tint value by a specified percentage.

Property	Description
<code>compiledClipInstance.description</code>	A string that is equivalent to the Description field in the Accessibility panel.
<code>compiledClipInstance.filters</code>	An array of Filter objects.
<code>compiledClipInstance.forceSimple</code>	A Boolean value that enables and disables the children of the object to be accessible.
<code>compiledClipInstance.shortcut</code>	A string that is equivalent to the Shortcut field in the Accessibility panel.
<code>compiledClipInstance.silent</code>	A Boolean value that enables or disables the accessibility of the object; equivalent to the inverse logic of the Make Object Accessible setting in the Accessibility panel.
<code>compiledClipInstance.tabIndex</code>	An integer that is equivalent to the Tab Index field in the Accessibility panel.
<code>compiledClipInstance.tintColor</code>	Aa Color object that, when the Color Effect Property Inspector is using style tint, returns the color applied to the tint.
<code>compiledClipInstance.tintColorPercent</code>	A string that, when the Color Effect Property Inspector is using style tint, returns the tint percentage.
<code>compiledClipInstance.useBackgroundColor</code>	A boolean that sets the background color.
<code>compiledClipInstance.visible</code>	A boolean that sets visibility.

## compiledClipInstance.accName

### Availability

Flash MX 2004.

### Usage

`compiledClipInstance.accName`

### Description

Property; a string that is equivalent to the Name field in the Accessibility panel. Screen readers identify objects by reading the name aloud.

### Example

The following example gets and sets the accessibility name of the first selected object:

```
// Get the name of the object.  
var theName = fl.getDocumentDOM().selection[0].accName;  
// Set the name of the object.  
fl.getDocumentDOM().selection[0].accName = 'Home Button';
```

## compiledClipInstance.actionScript - dropped

### Availability

Flash MX 2004. *Dropped in Flash Professional CC.*

**Usage**

```
compiledClipInstance.actionScript
```

**Description**

*Dropped in Flash Professional CC.*

Property; a string that represents the ActionScript for this instance; equivalent to [symbolInstance.actionScript - dropped](#).

**Example**

The following code assigns ActionScript to specified elements:

```
// Assign some ActionScript to a specified Button compiled clip instance.  
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0]  
    .actionScript = "on(click) {trace('button is clicked')}";  
// Assign some ActionScript to the currently selected Button compiled clip instance.  
fl.getDocumentDOM().selection[0].actionScript =  
    "on(click) {trace('button is clicked')}";
```

## compiledClipInstance.backgroundColor

**Availability**

Flash Professional CC.

**Usage**

```
compiledClipInstance.backgroundColor
```

**Description**

Property; a string that specifies the matte color when Opaque is selected. This is a string in hexadecimal #rrggbba format or an integer containing the value.

**Example**

The following example illustrates getting the backgroundColor property:

```
var bitmapInstance = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];  
bitmapInstance.backgroundColor = "#000000";
```

## compiledClipInstance.blendMode

**Availability**

Flash Professional CC.

**Usage**

```
compiledClipInstance.blendMode
```

**Description**

Property; a string that specifies the blend mode. Valid blend modes are: normal, layer, darken, multiply, lighten, screen, overlay, hardlight, add, subtract, difference, invert, alpha, and erase.

**Example**

The following example illustrates getting and setting the `blendMode` property:

```
//if the blend mode is 'add', change it to 'subtract'  
var blend = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].blendMode;  
fl.trace(blend);  
if (blend == 'add') {  
    fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].blendMode = 'subtract';  
}
```

## compiledClipInstance.brightness

**Availability**

Flash Professional CC.

**Usage**

```
compiledClipInstance.brightness
```

**Description**

Read-only property; an int that contains the value set in the Color Effect Property Inspector for brightness when `colorMode == 'brightness'`. Specify a percentage between -100 and 100. Returns an error if `colorMode` is a different setting.

**Example**

The following example illustrates use of this property:

```
var elem = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];  
if (elem.colorMode == 'brightness') {  
    fl.trace(elem.brightness);  
}
```

## compiledClipInstance.cacheAsBitmap

**Availability**

Flash Professional CC.

**Usage**

```
compiledClipInstance.cacheAsBitmap
```

**Description**

Property; a boolean that indicates whether to cache bitmaps. (Equivalent to Use runtime bitmap caching in the Property Inspector). The default is `false`.

**Example**

The following example illustrates use of this property:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].cacheAsBitmap = true;
```

## **compiledClipInstance.colorAlphaAmount**

**Availability**

Flash Professional CC.

**Usage**

```
compiledClipInstance.colorAlphaAmount
```

**Description**

Property; an int that reduces or increases the tint and alpha values by a constant amount. This value is added to the current value. This setting is most useful if used in conjunction with `colorAlphaPercent`. Valid values are -255 to 255.

This setting is the same as selecting Color > Advanced in the Instance Property Inspector and adjusting the controls on the right of the dialog.

**Example**

The following example illustrates use of this property:

```
//change the colorAlphaAmount of the first element in the first frame, top layer
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].colorAlphaAmount = 100;
//change the colorAlphaAmount of the selected symbol instance
f1.getDocumentDOM().selection[0].colorAlphaAmount = -100;
```

## **compiledClipInstance.colorAlphaPercent**

**Availability**

Flash Professional CC.

**Usage**

```
compiledClipInstance.colorAlphaPercent
```

**Description**

Property; an int that reduces or increases the tint and alpha values by a specified percentage. The current values are multiplied by this percentage. Valid values are -100 to 100.

This setting is the same as selecting Color > Advanced in the Instance Property Inspector and adjusting the controls on the left of the dialog

**Example**

The following example illustrates use of this property:

```
//change the colorAlphaPercent of the first element in the first frame, top layer
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].colorAlphaPercent = -100;
//change the colorAlphaPercent of the selected symbol instance
fl.getDocumentDOM().selection[0].colorAlphaPercent = 90;
```

## compiledClipInstance.colorBlueAmount

### Availability

Flash Professional CC.

### Usage

```
compiledClipInstance.colorBlueAmount
```

### Description

Property; an int that either reduces or increases the blue tint by a constant amount. This value is added to the current value. Valid values are -255 to 255.

This setting is the same as selecting Color > Advanced in the Instance Property Inspector.

### Example

The following example illustrates use of this property:

```
// Change the colorBlueAmount of the first element in the first frame, top layer
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].colorBlueAmount = 100;
//change the colorBlueAmount of the selected symbol instance
fl.getDocumentDOM().selection[0].colorBlueAmount = 255;
```

## compiledClipInstance.colorBluePercent

### Availability

Flash Professional CC.

### Usage

```
compiledClipInstance.colorBluePercent
```

### Description

Property; an int that reduces or increases the blue tint values by a specified percentage. The current values are multiplied by this percentage. Valid values are -100 to 100.

This setting is the same as selecting Color > Advanced in the Instance Property Inspector.

### Example

The following example illustrates use of this property:

```
//change the colorBluePercent of the first element in the first frame, top layer
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].colorBluePercent = 100;
//change the colorBluePercent of the selected symbol instance
fl.getDocumentDOM().selection[0].colorBluePercent = 80;
```

## compiledClipInstance.colorGreenAmount

### Availability

Flash Professional CC.

### Usage

```
compiledClipInstance.colorGreenAmount
```

### Description

Property; an int that either reduces or increases the green tint by a constant amount. This value is added to the current value. Valid values are -255 to 255.

This setting is the same as selecting Color > Advanced in the Instance Property Inspector.

### Example

The following example illustrates use of this property:

```
// change the colorGreenAmount of the first element in the first frame, top layer
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].colorGreenAmount = 100;
//change the colorGreenAmount of the selected symbol instance
fl.getDocumentDOM().selection[0].colorGreenAmount = 255;
```

## compiledClipInstance.colorGreenPercent

### Availability

Flash Professional CC.

### Usage

```
compiledClipInstance.colorGreenPercent
```

### Description

Property; an int that reduces or increases the green tint values by a specified percentage. The current values are multiplied by this percentage. Valid values are -100 to 100.

This setting is the same as selecting Color > Advanced in the Instance Property Inspector.

### Example

The following example illustrates use of this property:

```
/change the colorGreenPercent of the first element in the first frame, top layer
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].colorGreenPercent = 100;
//change the colorGreenPercent of the selected symbol instance
fl.getDocumentDOM().selection[0].colorGreenPercent = 80;
```

## compiledClipInstance.colorMode

### Availability

Flash Professional CC.

### Usage

```
compiledClipInstance.colorMode
```

### Description

Property; a string that specifies the color mode, as identified in the Symbol Properties dialog. Valid values are “none”, “brightness”, “tint”, “alpha”, and “advanced”.

### Example

The following example illustrates use of this property:

```
//change the colorMode of the first element in the first frame, top layer
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].colorMode = 'advanced';
```

## compiledClipInstance.colorRedAmount

### Availability

Flash Professional CC.

### Usage

```
compiledClipInstance.colorRedAmount
```

### Description

Property; an int that either reduces or increases the red tint by a constant amount. This value is added to the current value. Valid values are -255 to 255.

This setting is the same as selecting Color > Advanced in the Instance Property Inspector.

### Example

The following example illustrates use of this property:

```
// change the colorRedAmount of the first element in the first frame, top layer
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].colorRedAmount = 100;
//change the colorRedAmount of the selected symbol instance
fl.getDocumentDOM().selection[0].colorRedAmount = 255;
```

## compiledClipInstance.colorRedPercent

### Availability

Flash Professional CC.

**Usage**

```
compiledClipInstance.colorRedPercent
```

**Description**

Property; an int that reduces or increases the red tint values by a specified percentage. The current values are multiplied by this percentage. Valid values are -100 to 100.

This setting is the same as selecting Color > Advanced in the Instance Property Inspector.

**Example**

The following example illustrates use of this property:

```
//change the colorRedPercent of the first element in the first frame, top layer
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].colorRedPercent = 100;
//change the colorRedPercent of the selected symbol instance
fl.getDocumentDOM().selection[0].colorRedPercent = 80;
```

## **compiledClipInstance.description**

**Availability**

Flash MX 2004.

**Usage**

```
compiledClipInstance.description
```

**Description**

Property; a string that is equivalent to the Description field in the Accessibility panel. The description is read by the screen reader.

**Example**

The following example illustrates getting and setting the description property:

```
// Get the description of the current selection.
var theDescription = fl.getDocumentDOM().selection[0].description;
// Set the description of the current selection.
fl.getDocumentDOM().selection[0].description =
    "This is compiled clip number 1";
```

## **compiledClipInstance.filters**

**Availability**

Flash Professional CC.

**Usage**

```
compiledClipInstance.filters
```

**Description**

Property; an array of Filter objects. The properties of Filter object in the filters array can be read but cannot be written directly by accessing the filters array. To set the properties of the filter objects in the filters array, first retrieve the array, set the properties, set it back to the filters array.

**Example**

The following example illustrates use of this property:

```
//trace the name of the filter at index 0, if == glow filter, set its blurX to 100
var filterName =
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].filters[0].name;
f1.trace(filterName);
var filterArray = f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].filters;
if (filterName == 'glowFilter'){
    filterArray[0].blurX = 100;
}
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].filters = filterArray;
```

## compiledClipInstance.forceSimple

**Availability**

Flash MX 2004.

**Usage**

```
compiledClipInstance.forceSimple
```

**Description**

Property; a Boolean value that enables and disables the children of the object to be accessible. This is equivalent to the inverse logic of the Make Child Objects Accessible setting in the Accessibility panel. If forceSimple is true, it is the same as the Make Child Objects Accessible option being unchecked. If forceSimple is false, it is the same as the Make Child Object Accessible option being checked.

**Example**

The following example illustrates getting and setting the forceSimple property:

```
// Query if the children of the object are accessible.
var areChildrenAccessible = f1.getDocumentDOM().selection[0].forceSimple;
// Allow the children of the object to be accessible.
f1.getDocumentDOM().selection[0].forceSimple = false;
```

## compiledClipInstance.shortcut

**Availability**

Flash MX 2004.

**Usage**

```
compiledClipInstance.shortcut
```

**Description**

Property; a string that is equivalent to the Shortcut field in the Accessibility panel. The shortcut is read by the screen reader. This property is not available for dynamic text fields.

**Example**

The following example illustrates getting and setting the `shortcut` property:

```
// Get the shortcut key of the object.  
var theShortcut = fl.getDocumentDOM().selection[0].shortcut;  
// Set the shortcut key of the object.  
fl.getDocumentDOM().selection[0].shortcut = "Ctrl+I";
```

## **compiledClipInstance.silent**

**Availability**

Flash MX 2004.

**Usage**

```
compiledClipInstance.silent
```

**Description**

Property; a Boolean value that enables or disables the accessibility of the object; equivalent to the inverse logic of Make Object Accessible setting in the Accessibility panel. That is, if `silent` is `true`, then Make Object Accessible is unchecked. If `silent` is `false`, then Make Object Accessible is checked.

**Example**

The following example illustrates getting and setting the `silent` property:

```
// Query if the object is accessible.  
var isSilent = fl.getDocumentDOM().selection[0].silent;  
// Set the object to be accessible.  
fl.getDocumentDOM().selection[0].silent = false;
```

## **compiledClipInstance.tabIndex**

**Availability**

Flash MX 2004.

**Usage**

```
compiledClipInstance.tabIndex
```

**Description**

Property; an integer that is equivalent to the Tab Index field in the Accessibility panel. Creates a tab order in which objects are accessed when the user presses the Tab key.

**Example**

The following example illustrates getting and setting the `tabIndex` property:

```
// Get the tabIndex of the object.  
var theTabIndex = fl.getDocumentDOM().selection[0].tabIndex;  
// Set the tabIndex of the object.  
fl.getDocumentDOM().selection[0].tabIndex = 1;
```

## compiledClipInstance.tintColor

### Availability

Flash Professional CC.

### Usage

```
compiledClipInstance.tintColor
```

### Description

Read-only property; a Color object that, when the Color Effect Property Inspector is using style tint (colorMode == 'tint'), returns the color applied to the tint. Otherwise, using this property results in an error.

### Example

The following example illustrates use of this property:

```
var elem = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];  
if (elem.colorMode == 'tint') {  
    fl.trace(elem.tintColor);  
    fl.trace(elem.tintColorPercent);  
}
```

## compiledClipInstance.tintColorPercent

### Availability

Flash Professional CC.

### Usage

```
compiledClipInstance.tintColorPercent
```

### Description

Read-only property; a string that, when the Color Effect Property Inspector is using style tint (colorMode == 'tint'), returns the tint percentage from -100 to 100. Otherwise, using this property results in an error.

### Example

The following example illustrates use of this property:

```
var elem = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];  
if (elem.colorMode == 'tint') {  
    fl.trace(elem.tintColor);  
    fl.trace(elem.tintColorPercent);  
}
```

## compiledClipInstance.useBackgroundColor

### Availability

Flash Professional CC.

### Usage

```
compiledClipInstance.useBackgroundColor
```

### Description

Property; a boolean that sets the background color:

- true - Use 32-bit with alpha.
- false - Use the background color.

### Example

The following example illustrates use of this property:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].useBackgroundColor = true;
```

## compiledClipInstance.visible

### Availability

Flash Professional CC.

### Usage

```
compiledClipInstance.visible
```

### Description

Property; a boolean that sets visibility. Equivalent to the visible checkbox in the Display section of the Property Inspector for symbols.

### Example

The following example illustrates use of this property:

```
// change visible to false on currently selected symbol instance
f1.getDocumentDOM().selection[0].visible = false;
```

# Chapter 7: compilerErrors object

## compilerErrors summary

### Availability

Flash CS3 Professional.

### Description

The compilerErrors object, which represents the Compiler Errors panel, is a property of the flash object (fl) and can be accessed by `f1.compilerErrors` (see [flash object \(fl\)](#)).

### Method summary

The following methods can be used with the compilerErrors object:

Method	Description
<code>compilerErrors.clear()</code>	Clears the contents of the Compiler Errors panel.
<code>compilerErrors.save()</code>	Saves the contents of the Compiler Errors panel to a local text file.

## compilerErrors.clear()

### Availability

Flash CS3 Professional.

### Usage

```
compilerErrors.clear()
```

### Parameters

None.

### Returns

Nothing.

### Description

Method; clears the contents of the Compiler Errors panel.

### Example

The following example clears the contents of the Compiler Errors panel:

```
f1.compilerErrors.clear();
```

### See also

[compilerErrors.save\(\)](#)

## compilerErrors.save()

### Availability

Flash CS3 Professional.

### Usage

```
compilerErrors.save(fileURI [, bAppendToFile [, bUseSystemEncoding]])
```

### Parameters

**fileURI** A string, expressed as a file:/// URI, that specifies the filename for the saved file. If *fileURI* already exists, and you haven't specified a value of `true` for *bAppendToFile*, *fileURI* is overwritten without warning.

**bAppendToFile** An optional Boolean value that specifies whether the contents of the Compiler Errors panel should be appended to *fileURI* (`true`) or not (`false`). The default value is `false`.

**bUseSystemEncoding** An optional Boolean value that specifies whether to save the Compiler Errors panel text using the system encoding. If this value is `false` (the default), the Compiler Errors panel text is saved using UTF-8 encoding, with Byte Order Mark characters at the beginning of the text. The default value is `false`.

### Returns

Nothing.

### Description

Method; saves the contents of the Compiler Errors panel to a local text file.

### Example

The following example saves the contents of the Compiler Errors panel to a file named errors.log in the C:\tests folder:

```
f1.compilerErrors.save("file:///c|/tests/errors.log");
```

### See also

[compilerErrors.clear\(\)](#)

# Chapter 8: ComponentInstance object

## componentInstance summary

**Inheritance** [Element object](#) > [Instance object](#) > [SymbolInstance object](#) > ComponentInstance object

### Availability

Flash MX 2004.

### Description

The ComponentInstance object is a subclass of the SymbolInstance object and represents a component in a frame (see [SymbolInstance object](#)).

### Property summary

In addition to all the properties of the [SymbolInstance object](#), the ComponentInstance object has the following property:

Property	Description
<code>componentInstance.parameters</code>	Read-only; an array of ActionScript 2.0 properties that are accessible from the component Property inspector.

## componentInstance.parameters

### Availability

Flash MX 2004.

### Usage

```
componentInstance.parameters
```

### Description

Read-only property; an array of ActionScript 2.0 properties that are accessible from the component Property inspector. See [Parameter object](#).

### Example

The following example illustrates getting and setting the `parameters` property:

```
var parms = fl.getDocumentDOM().selection[0].parameters;
parms[0].value = "some value";
```

### See also

[Parameter object](#)

# Chapter 9: componentsPanel object

## componentsPanel summary

### Availability

Flash MX 2004.

### Description

The componentsPanel object, which represents the Components panel, is a property of the flash object (fl) and can be accessed by `fl.componentsPanel` (see [flash object \(fl\)](#)).

### Method summary

You can use the following methods with the componentsPanel object:

Method	Description
<code>componentsPanel.addItemToDocument()</code>	Adds the specified component to the document at the specified position.
<code>componentsPanel.reload()</code>	Refreshes the Components panel's list of components.

## componentsPanel.addItemToDocument()

### Availability

Flash MX 2004.

### Usage

```
componentsPanel.addItemToDocument(position, categoryName, componentName)
```

### Parameters

**position** A point (for example, `{x:0, y:100}`) that specifies the location at which to add the component. Specify *position* relative to the center point of the component—not the component's registration point (also *origin point* or *zero point*).

**categoryName** A string that specifies the name of the component category (for example, "Data"). The valid category names are listed in the Components panel.

**componentName** A string that specifies the name of the component in the specified category (for example, "WebServiceConnector"). The valid component names are listed in the Components panel.

### Returns

Nothing.

### Description

Adds the specified component to the document at the specified position.

**Example**

The following examples illustrate some ways to use this method:

```
f1.componentsPanel.addItemToDocument({x:0, y:0}, "User Interface", "CheckBox");
f1.componentsPanel.addItemToDocument({x:0, y:100}, "Data", "WebServiceConnector");
f1.componentsPanel.addItemToDocument({x:0, y:200}, "User Interface", "Button");
```

## **componentsPanel.reload()**

**Availability**

Flash 8.

**Usage**

```
componentsPanel.reload()
```

**Parameters**

None.

**Returns**

A Boolean value of `true` if the Component panel list is refreshed, `false` otherwise.

**Description**

Method; refreshes the Components panel's list of components.

**Example**

The following example refreshes the Components panel:

```
f1.componentsPanel.reload();
```

# Chapter 10: Contour object

## contour summary

### Availability

Flash MX 2004.

### Description

A Contour object represents a closed path of half edges on the boundary of a shape.

### Method summary

You can use the following method with the Contour object:

Method	Description
<code>contour.getHalfEdge()</code>	Returns a <a href="#">HalfEdge object</a> on the contour of the selection.

### Property summary

You can use the following properties with the Contour object:

Property	Description
<code>contour.fill</code>	A <a href="#">Fill object</a> .
<code>contour.interior</code>	Read-only; the value is <code>true</code> if the contour encloses an area; <code>false</code> otherwise.
<code>contour.orientation</code>	Read-only; an integer indicating the orientation of the contour.

## contour.fill

### Availability

Flash CS4 Professional.

### Usage

```
contour.fill
```

### Description

Property; a [Fill object](#).

### Example

Assuming that you have a contour with a fill selected, the following example displays the contour's fill color in the Output panel:

```
var insideContour = fl.getDocumentDOM().selection[0].contours[1];
var insideFill = insideContour.fill;
fl.trace(insideFill.color);
```

## contour.getHalfEdge()

### Availability

Flash MX 2004.

### Usage

```
contour.getHalfEdge()
```

### Parameters

None.

### Returns

A [HalfEdge object](#).

### Description

Method; returns a [HalfEdge object](#) on the contour of the selection.

### Example

This example traverses all the contours of the selected shape and shows the coordinates of the vertices in the Output panel:

```
// with a shape selected

var elt = fl.getDocumentDOM().selection[0];
elt.beginEdit();

var contourArray = elt.contours;
var contourCount = 0;
for (i=0;i<contourArray.length;i++)
{
    var contour = contourArray[i];
    contourCount++;
    var he = contour.getHalfEdge();

    var iStart = he.id;
    var id = 0;
    while (id != iStart)
    {
        // Get the next vertex.
        var vrt = he.getNext();

        var x = vrt.x;
        var y = vrt.y;
        fl.trace("vrt: " + x + ", " + y);

        he = he.getNext();
        id = he.id;
    }
}
elt.endEdit();
```

## contour.interior

### Availability

Flash MX 2004.

### Usage

```
contour.interior
```

### Description

Read-only property; the value is `true` if the contour encloses an area; `false` otherwise.

### Example

This example traverses all the contours of the selected shape and shows the value of the `interior` property for each contour in the Output panel:

```
var elt = fl.getDocumentDOM().selection[0];
elt.beginEdit();

var contourArray = elt.contours;

var contourCount = 0;
for (i=0;i<contourArray.length;i++) {
    var contour = contourArray[i];
    fl.trace("Next Contour, interior:" + contour.interior );
    contourCount++;
}
elt.endEdit();
```

## contour.orientation

### Availability

Flash MX 2004.

### Usage

```
contour.orientation
```

### Description

Read-only property; an integer indicating the orientation of the contour. The value of the integer is -1 if the orientation is counterclockwise, 1 if it is clockwise, and 0 if it is a contour with no area.

### Example

The following example traverses all the contours of the selected shape and shows the value of the `orientation` property of each contour in the Output panel:

```
var elt = fl.getDocumentDOM().selection[0];
elt.beginEdit();

var contourArray = elt.contours;

var contourCount = 0;
for (i=0;i<contourArray.length;i++) {
    var contour = contourArray[i];
    fl.trace("Next Contour, orientation:" + contour.orientation);
    contourCount++;
}
elt.endEdit();
```

# Chapter 11: Document object

## document summary

### Availability

Flash MX 2004.

### Description

The Document object represents the Stage. That is, only FLA files are considered documents. To return the Document object for the current document, use `f1.getDocumentDOM()`.

### Method summary

You can use the following methods with the Document object:

Method	Description
<code>document.addDataToDocument()</code>	Stores specified data with a document.
<code>document.addDataToSelection()</code>	Stores specified data with the selected objects.
<code>document.addFilter()</code>	Applies a filter to the selected objects.
<code>document.addItem()</code>	Adds an item from any open document or library to the specified Document object.
<code>document.add.NewLine()</code>	Adds a new path between two points.
<code>document.addNewOval()</code>	Adds a new Oval object in the specified bounding rectangle.
<code>document.addNewPrimitiveOval()</code>	Adds a new oval primitive fitting into the specified bounds.
<code>document.addNewPrimitiveRectangle()</code>	Adds a new rectangle primitive fitting into the specified bounds.
<code>document.addNewPublishProfile()</code>	Adds a new publish profile and makes it the current one.
<code>document.addNewRectangle()</code>	Adds a new rectangle or rounded rectangle, fitting it into the specified bounds.
<code>document.addNewScene()</code>	Adds a new scene ( <a href="#">Timeline object</a> ) as the next scene after the currently selected scene and makes the new scene the currently selected scene.
<code>document.addNewText()</code>	Inserts a new empty text field.
<code>document.align()</code>	Aligns the selection.
<code>document.allowScreens() - dropped</code>	Dropped in Flash Professional CC.
<code>document.arrange()</code>	Arranges the selection on the Stage.
<code>document.breakApart()</code>	Performs a break-apart operation on the current selection.
<code>document.canEditSymbol()</code>	Indicates whether the Edit Symbols menu and functionality are enabled.
<code>document.canRevert()</code>	Determines whether you can use the <code>document.revert()</code> or <code>f1.revertDocument()</code> method successfully.

Method	Description
<code>document.canTestMovie()</code>	Determines whether you can use the <code>document.testMovie()</code> method successfully.
<code>document.canTestScene()</code>	Determines whether you can use the <code>document.testScene()</code> method successfully.
<code>document.changeFilterOrder()</code>	Changes the index of the filter in the Filters list.
<code>document.clipCopy()</code>	Copies the current selection from the document to the Clipboard.
<code>document.clipCut()</code>	Cuts the current selection from the document and writes it to the Clipboard.
<code>document.clipPaste()</code>	Pastes the contents of the Clipboard into the document.
<code>document.close()</code>	Closes the specified document.
<code>document.convertLinesToFills()</code>	Converts lines to fills on the selected objects.
<code>document.convertSelectionToBitmap()</code>	Converts selected objects in the current frame to a bitmap and inserts the bitmap into the library.
<code>document.convertToSymbol()</code>	Converts the selected Stage items to a new symbol.
<code>document.crop()</code>	Uses the top selected drawing object to crop all selected drawing objects underneath it.
<code>document.debugMovie()</code>	Initiates a debug session with the document.
<code>document.deleteEnvelope()</code>	Deletes the envelope (bounding box that contains one or more objects) from the selected object.
<code>document.deletePublishProfile()</code>	Deletes the currently active profile, if there is more than one.
<code>document.deleteScene()</code>	Deletes the current scene ( <a href="#">Timeline object</a> ), and if the deleted scene was not the last one, sets the next scene as the current Timeline object.
<code>document.deleteSelection()</code>	Deletes the current selection on the Stage.
<code>document.disableAllFilters()</code>	Disables all filters on the selected objects.
<code>document.disableFilter()</code>	Disables the specified filter in the Filters list.
<code>document.disableOtherFilters()</code>	Disables all filters except the one at the specified position in the Filters list.
<code>document.distribute()</code>	Distributes the selection.
<code>document.distributeToKeyframes()</code>	Performs a distribute-to-keyframes operation on the current selection; equivalent to selecting Distribute to Keyframes.
<code>document.distributeToLayers()</code>	Performs a distribute-to-layers operation on the current selection; equivalent to selecting Distribute to Layers.
<code>document.documentHasData()</code>	Checks the document for persistent data with the specified name.
<code>document.duplicatePublishProfile()</code>	Duplicates the currently active profile and gives the duplicate version focus.
<code>document.duplicateScene()</code>	Makes a copy of the currently selected scene, giving the new scene a unique name and making it the current scene.
<code>document.duplicateSelection()</code>	Duplicates the selection on the Stage.

Method	Description
<code>document.editScene()</code>	Makes the specified scene the currently selected scene for editing.
<code>document.enableAllFilters()</code>	Enables all the filters on the Filters list for the selected objects.
<code>document.enableFilter()</code>	Enables the specified filter for the selected objects.
<code>document.enterEditMode()</code>	Switches the authoring tool into the editing mode specified by the parameter.
<code>document.exitEditMode()</code>	Exits from symbol-editing mode and returns focus to the next level up from the editing mode.
<code>document.exportInstanceToLibrary()</code>	Export a selected movie clip, graphic, or button instance on the Stage to a new bitmap in the Library.
<code>document.exportInstanceToPNGSequence()</code>	Export a selected movie clip, graphic, or button instance on the Stage to a series of PNG files on disk.
<code>document.exportPNG()</code>	Exports the document as one or more PNG files.
<code>document.exportPublishProfile()</code>	Exports the currently active profile to an XML file.
<code>document.exportPublishProfileString()</code>	Returns a string that specifies, in XML format, the specified profile.
<code>document.exportSWF()</code>	Exports the document in the Flash SWF format.
<code>document.exportVideo()</code>	Exports a video from the document and optionally sends it to Adobe Media Encoder to convert the video.
<code>document.getAlignToDocument()</code>	Identical to retrieving the value of the To Stage button in the Align panel.
<code>document.getBlendMode()</code>	Returns a string that specifies the blending mode for the selected objects.
<code>document.getCustomFill()</code>	Retrieves the fill object of the selected shape, or the Tools panel and Property inspector if specified.
<code>document.getCustomStroke()</code>	Returns the stroke object of the selected shape, or the Tools panel and Property inspector if specified.
<code>document.getDataFromDocument()</code>	Retrieves the value of the specified data.
<code>document.getElementProperty()</code>	Gets the specified Element property for the current selection.
<code>document.getTextTextAttrs()</code>	Gets a specified TextAttrs property of the selected Text objects.
<code>document.getFilters()</code>	Returns an array that contains the list of filters applied to the currently selected objects.
<code>document.getMetadata()</code>	Returns a string containing the XML metadata associated with the document.
<code>document.getMobileSettings()</code>	Returns the string passed to <code>document.setMobileSettings()</code> .
<code>document.getPlayerVersion()</code>	Returns a string that represents the targeted player version for the specified document.
<code>document.getPublishDocumentData()</code>	Indicates whether publishing of the specified persistent data is enabled for the specified format in the document.
<code>document.getSelectionRect()</code>	Gets the bounding rectangle of the current selection.

Method	Description
<code>document.getSWFPathFromProfile()</code>	Gets the full path to the SWF file that is set in the current Publish profile.
<code>document.getTelemetryForSwf()</code>	Indicates whether the “Enable detailed telemetry” checkbox is selected in the Publish Settings dialog.
<code>document.getTextString()</code>	Gets the currently selected text.
<code>document.getTimeline()</code>	Retrieves the current <a href="#">Timeline object</a> in the document.
<code>document.getTransformationPoint()</code>	Gets the location of the transformation point of the current selection.
<code>document.group()</code>	Converts the current selection to a group.
<code>document.importFile()</code>	Imports a file into the document.
<code>document.importPublishProfile()</code>	Imports a profile from a file.
<code>document.importPublishProfileString()</code>	Imports an XML string that represents a publish profile and sets it as the current profile.
<code>document.importSWF() - dropped</code>	Dropped in Flash Professional CC.
<code>document.intersect()</code>	Creates an intersection drawing object from all selected drawing objects.
<code>document.loadCuepointXML() - dropped</code>	Dropped in Flash Professional CC.
<code>document.match()</code>	Makes the size of the selected objects the same.
<code>document.mouseClick()</code>	Performs a mouse click from the Selection tool.
<code>document.mouseDblClk()</code>	Performs a double mouse click from the Selection tool.
<code>document.moveSelectedBezierPointsBy()</code>	If the selection contains at least one path with at least one Bézier point selected, this method moves all selected Bézier points on all selected paths by the specified amount.
<code>document.moveSelectionBy()</code>	Moves selected objects by a specified distance.
<code>document.optimizeCurves()</code>	Optimizes smoothing for the current selection, allowing multiple passes, if specified, for optimal smoothing; equivalent to selecting Modify > Shape > Optimize.
<code>document.publish()</code>	Publishes the document according to the active publish settings (File > Publish Settings); equivalent to selecting File > Publish.
<code>document.punch()</code>	Uses the top selected drawing object to punch through all selected drawing objects underneath it.
<code>document.removeAllFilters()</code>	Removes all filters from the selected objects.
<code>document.removeDataFromDocument()</code>	Removes persistent data with the specified name that has been attached to the document.
<code>document.removeDataFromSelection()</code>	Removes persistent data with the specified name that has been attached to the selection.
<code>document.removeFilter()</code>	Removes the specified filter from the Filters list of the selected objects.
<code>document.renamePublishProfile()</code>	Renames the current profile.
<code>document.renameScene()</code>	Renames the currently selected scene in the Scenes panel.

Method	Description
<code>document.reorderScene()</code>	Moves the specified scene before another specified scene.
<code>document.resetOvalObject()</code>	Sets all values in the Property inspector to default Oval object settings.
<code>document.resetRectangleObject()</code>	Sets all values in the Property inspector to default Rectangle object settings.
<code>document.resetTransformation()</code>	Resets the transformation matrix; equivalent to selecting Modify > Transform > Remove Transform.
<code>document.revert()</code>	Reverts the specified document to its previously saved version; equivalent to selecting File > Revert.
<code>document.rotate3DSelection()</code>	Applies a 3D rotation to the selection.
<code>document.rotateSelection()</code>	Rotates the selection by a specified number of degrees.
<code>document.save()</code>	Saves the document in its default location; equivalent to selecting File > Save.
<code>document.saveAsCopy()</code>	Saves a copy of the document to a specified location.
<code>document.saveAndCompact() - dropped</code>	Dropped in Flash Professional CS6.
<code>document.scaleSelection()</code>	Scales the selection by a specified amount; equivalent to using the Free Transform tool to scale the object.
<code>document.selectAll()</code>	Selects all items on the Stage; equivalent to pressing Control+A (Windows) or Command+A (Macintosh) or selecting Edit > Select All.
<code>document.selectNone()</code>	Deselects any selected items.
<code>document.setAlignToDocument()</code>	Sets the preferences for <code>document.align()</code> , <code>document.distribute()</code> , <code>document.match()</code> , and <code>document.space()</code> to act on the document; equivalent to enabling the To Stage button in the Align panel.
<code>document.setBlendMode()</code>	Sets the blending mode for the selected objects.
<code>document.setCustomFill()</code>	Sets the fill settings for the Tools panel, Property inspector, and any selected shapes.
<code>document.setCustomStroke()</code>	Sets the stroke settings for the Tools panel, Property inspector, and any selected shapes.
<code>document.setProperty()</code>	Sets the specified Element property on selected objects in the document.
<code>document.setTextTextAttr()</code>	Sets the specified <code>TextAttrs</code> property of the selected text items to the specified value.
<code>document.setFillColor()</code>	Changes the selection and the tools panel to the specified color.
<code>document.setFilterProperty()</code>	Sets a specified filter property for the currently selected objects.
<code>document.setFilters()</code>	Applies filters to the selected objects.
<code>document.setInstanceAlpha()</code>	Sets the opacity of the instance.
<code>document.setInstanceBrightness()</code>	Sets the brightness for the instance.
<code>document.setInstanceTint()</code>	Sets the tint for the instance.

Method	Description
<code>document.setMetadata()</code>	Sets the XML metadata for the specified document, overwriting any existing metadata.
<code>document.setMobileSettings()</code>	Sets the value of an XML settings string in a mobile FLA file.
<code>document.setOvalObjectProperty()</code>	Specifies a value for a specified property of primitive Oval objects.
<code>document.setPlayerVersion()</code>	Sets the version of the Flash Player targeted by the specified document.
<code>document.setRectangleObjectProperty()</code>	Specifies a value for a specified property of primitive Rectangle objects.
<code>document.setSelectionBounds()</code>	Moves and resizes the selection in a single operation.
<code>document.setSelectionRect()</code>	Draws a rectangular selection marquee relative to the Stage, using the specified coordinates.
<code>document.setStageVanishingPoint()</code>	Specifies the vanishing point for viewing 3D objects.
<code>document.setStageViewAngle()</code>	Specifies the perspective angle for viewing 3D objects.
<code>document.setStroke()</code>	Sets the color, width, and style of the selected strokes.
<code>document.setStrokeColor()</code>	Changes the stroke color of the selection to the specified color.
<code>document.setStrokeSize()</code>	Changes the stroke size of the selection to the specified size.
<code>document.setStrokeStyle()</code>	Changes the stroke style of the selection to the specified style.
<code>document.setTextRectangle()</code>	Changes the bounding rectangle for the selected text item to the specified size.
<code>document.setTextSelection()</code>	Sets the text selection of the currently selected text field to the values specified by the <code>startIndex</code> and <code>endIndex</code> values.
<code>document.setTextString()</code>	Inserts a string of text.
<code>document.setTransformationPoint()</code>	Moves the transformation point of the current selection.
<code>document.skewSelection()</code>	Skews the selection by a specified amount.
<code>document.smoothSelection()</code>	Smooths the curve of each selected fill outline or curved line.
<code>document.space()</code>	Spaces the objects in the selection evenly.
<code>document.straightenSelection()</code>	Straightens the currently selected strokes; equivalent to using the Straighten button in the Tools panel.
<code>document.swapElement()</code>	Swaps the current selection with the specified one.
<code>document.swapStrokeAndFill()</code>	Swaps the Stroke and Fill colors.
<code>document.testMovie()</code>	Executes a Test Movie operation on the document.
<code>document.testScene()</code>	Executes a Test Scene operation on the current scene of the document.
<code>document.traceBitmap()</code>	Performs a trace bitmap on the current selection; equivalent to selecting Modify > Bitmap > Trace Bitmap.
<code>document.transformSelection()</code>	Performs a general transformation on the current selection by applying the matrix specified in the arguments.
<code>document.translate3DCenter()</code>	Sets the XYZ position around which the selection is translated or rotated.

Method	Description
<code>document.translate3DSelection()</code>	Applies a 3D translation to the selection.
<code>document.unGroup()</code>	Ungroups the current selection.
<code>document.union()</code>	Combines all selected shapes into a drawing object.
<code>document.unlockAllElements()</code>	Unlocks all locked elements on the currently selected frame.
<code>document.xmlPanel()</code>	Posts a XMLUI dialog box.

### Property summary

You can use the following properties with the Document object.

Property	Description
<code>document.accName</code>	A string that is equivalent to the Name field in the Accessibility panel.
<code>document.as3AutoDeclare</code>	A Boolean value that describes whether the instances placed on the Stage are automatically added to user-defined timeline classes.
<code>document.as3Dialect</code>	A string that describes the ActionScript 3.0 "dialect" being used in the specified document.
<code>document.as3ExportFrame</code>	An integer that specifies in which frame to export ActionScript 3.0 classes.
<code>document.as3StrictMode</code>	A Boolean value that specifies whether the ActionScript 3.0 compiler should compile with the Strict Mode option turned on or off.
<code>document.as3WarningsMode</code>	A Boolean value that specifies whether the ActionScript 3.0 compiler should compile with the Warnings Mode option turned on or off.
<code>document.asVersion</code>	An integer that specifies which version of ActionScript is being used in the specified file.
<code>document.autoLabel</code>	A Boolean value that is equivalent to the Auto Label check box in the Accessibility panel.
<code>document.backgroundColor</code>	A string, hexadecimal value, or integer that represents the background color.
<code>document.currentPublishProfile</code>	A string that specifies the name of the active publish profile for the specified document.
<code>document.currentTimeline</code>	An integer that specifies the index of the active timeline.
<code>document.description</code>	A string that is equivalent to the Description field in the Accessibility panel.
<code>document.docClass</code>	Specifies the top-level ActionScript 3.0 class associated with the document.
<code>document.externalLibraryPath</code>	A string that contains a list of items in the document's ActionScript 3.0 External library path, which specifies the location of SWC files used as runtime shared libraries.
<code>document.forceSimple</code>	A Boolean value that specifies whether the children of the specified object are accessible.
<code>document.frameRate</code>	A float value that specifies the number of frames displayed per second when the SWF file plays; the default is 12.
<code>document.height</code>	An integer that specifies the height of the document (Stage) in pixels.

Property	Description
<code>document.id</code>	A unique integer (assigned automatically) that identifies a document during a Flash session.
<code>document.library</code>	Read-only; the <a href="#">library object</a> for a document.
<code>document.libraryPath</code>	A string that contains a list of items in the document's ActionScript 3.0 Library path, which specifies the location of SWC files or folders containing SWC files.
<code>document.livePreview</code>	A Boolean value that specifies whether Live Preview is enabled.
<code>document.name</code>	Read-only; a string that represents the name of a document (FLA file).
<code>document.path</code>	Read-only; a string that represents the path of the document, in a platform-specific format.
<code>document.pathURI</code>	Read-only; a string that represents the path of the document, expressed as a file:/// URI.
<code>document.publishProfiles</code>	Read-only; an array of the publish profile names for the document.
<code>document.screenOutline - dropped</code>	Dropped in Flash Professional CC.
<code>document.selection</code>	An array of the selected objects in the document.
<code>document.silent</code>	A Boolean value that specifies whether the object is accessible.
<code>document.sourcePath</code>	A string that contains a list of items in the document's ActionScript 3.0 Source path, which specifies the location of ActionScript class files.
<code>document.swfJPEGQuality</code>	An integer that returns the JPEG Quality setting from the current Publish Profile in the document.
<code>document.timelines</code>	Read-only; an array of Timeline objects (see <a href="#">Timeline object</a> ).
<code>document.viewMatrix</code>	Read-only; a <a href="#">Matrix object</a> .
<code>document.width</code>	An integer that specifies the width of the document (Stage) in pixels.
<code>document.zoomFactor</code>	Specifies the zoom percent of the Stage at authoring time.

## document.accName

### Availability

Flash MX 2004.

### Usage

```
document.accName
```

### Description

Property; a string that is equivalent to the Name field in the Accessibility panel. Screen readers identify objects by reading the name aloud.

### Example

The following example sets the accessibility name of the document to "Main Movie":

```
f1.getDocumentDOM().accName = "Main Movie";
```

The following example gets the accessibility name of the document:

```
fl.trace(f1.getDocumentDOM().accName);
```

## document.addDataToDocument()

### Availability

Flash MX 2004.

### Usage

```
document.addDataToDocument(name, type, data)
```

### Parameters

**name** A string that specifies the name of the data to add.

**type** A string that defines the type of data to add. Acceptable values are "integer", "integerArray", "double", "doubleArray", "string", and "byteArray".

**data** The value to add. Valid types depend on the *type* parameter.

### Returns

Nothing.

### Description

Method; stores specified data with a document. Data is written to the FLA file and is available to JavaScript when the file reopens.

### Example

The following example adds an integer value of 12 to the current document:

```
f1.getDocumentDOM().addDataToDocument("myData", "integer", 12);
```

The following example returns the value of the data named "myData" and displays the result in the Output panel:

```
fl.trace(f1.getDocumentDOM().getDataFromDocument("myData"));
```

### See also

[document.getDataFromDocument\(\)](#), [document.removeDataFromDocument\(\)](#)

## document.addDataToSelection()

### Availability

Flash MX 2004.

### Usage

```
document.addDataToSelection(name, type, data)
```

**Parameters**

**name** A string that specifies the name of the persistent data.

**type** Defines the type of data. Acceptable values are "integer", "integerArray", "double", "doubleArray", "string", and "byteArray".

**data** The value to add. Valid types depend on the *type* parameter.

**Returns**

Nothing.

**Description**

Method; stores specified data with the selected objects. Data is written to the FLA file and is available to JavaScript when the file reopens. Only symbols and bitmaps support persistent data.

**Example**

The following example adds an integer value of 12 to the selected object:

```
f1.getDocumentDOM().addDataToSelection("myData", "integer", 12);
```

**See also**

[document.removeDataFromSelection\(\)](#)

## document.addFilter()

**Availability**

Flash 8.

**Usage**

```
document.addFilter(filterName)
```

**Parameters**

**filterName** A string specifying the filter to be added to the Filters list and enabled for the selected objects. Acceptable values are "adjustColorFilter", "bevelFilter", "blurFilter", "dropShadowFilter", "glowFilter", "gradientBevelFilter", and "gradientGlowFilter".

**Returns**

Nothing.

**Description**

Method; applies a filter to the selected objects and places the filter at the end of the Filters list.

**Example**

The following example applies a glow filter to the selected objects:

```
f1.getDocumentDOM().addFilter("glowFilter");
```

**See also**

`document.changeFilterOrder()`, `document.disableFilter()`, `document.enableFilter()`,  
`document.getFilters()`, `document.removeFilter()`, `document.setBlendMode()`,  
`document.setFilterProperty()`

## document.addItem()

**Availability**

Flash MX 2004.

**Usage**

```
document.addItem(position, item)
```

**Parameters**

**position** A point that specifies the *x* and *y* coordinates of the location at which to add the item. It uses the center of a symbol or the upper left corner of a bitmap or video.

**item** An Item object that specifies the item to add and the library from which to add it (see [Item object](#)).

**Returns**

A Boolean value: `true` if successful; `false` otherwise.

**Description**

Method; adds an item from any open document or library to the specified Document object.

**Example**

The following example adds the first item from the library to the first document at the specified location for the selected symbol, bitmap, or video:

```
var item = fl.documents[0].library.items[0];  
fl.documents[0].addItem({x:0,y:0}, item);
```

The following example adds the symbol `myMovieClip` from the current document's library to the current document:

```
var itemIndex = fl.getDocumentDOM().library.findItemIndex("myMovieClip");  
var theItem = fl.getDocumentDOM().library.items[itemIndex];  
fl.getDocumentDOM().addItem({x:0,y:0}, theItem);
```

The following example adds the symbol `myMovieClip` from the second document in the documents array to the third document in the documents array:

```
var itemIndex = fl.documents[1].library.findItemIndex("myMovieClip");  
var theItem = fl.documents[1].library.items[itemIndex];  
fl.documents[2].addItem({x:0,y:0}, theItem);
```

## document.add.NewLine()

**Availability**

Flash MX 2004.

**Usage**

```
document.addNewLine(startPoint, endpoint)
```

**Parameters**

**startPoint** A pair of floating-point numbers that specify the *x* and *y* coordinates where the line starts.

**endpoint** A pair of floating-point numbers that specify the *x* and *y* coordinates where the line ends.

**Returns**

Nothing.

**Description**

Method; adds a new path between two points. The method uses the document's current stroke attributes and adds the path on the current frame and current layer. This method works in the same way as clicking on the line tool and drawing a line.

**Example**

The following example adds a line between the specified starting point and ending point:

```
f1.getDocumentDOM().addNewLine({x:216.7, y:122.3}, {x:366.8, y:165.8});
```

## document.addNewOval()

**Availability**

Flash MX 2004.

**Usage**

```
document.addNewOval(boundingRectangle [, bSuppressFill [, bSuppressStroke ]])
```

**Parameters**

**boundingRectangle** A rectangle that specifies the bounds of the oval to be added. For information on the format of *boundingRectangle*, see [document.addNewRectangle\(\)](#).

**bSuppressFill** A Boolean value that, if set to `true`, causes the method to create the shape without a fill. The default value is `false`. This parameter is optional.

**bSuppressStroke** A Boolean value that, if set to `true`, causes the method to create the shape without a stroke. The default value is `false`. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; adds a new Oval object in the specified bounding rectangle. This method performs the same operation as the Oval tool. The method uses the document's current default stroke and fill attributes and adds the oval on the current frame and layer. If both *bSuppressFill* and *bSuppressStroke* are set to `true`, the method has no effect.

### Example

The following example adds a new oval within the specified coordinates; it is 164 pixels in width and 178 pixels in height:

```
f1.getDocumentDOM().addNewOval({left:72,top:50,right:236,bottom:228});
```

The following example draws the oval without a fill:

```
f1.getDocumentDOM().addNewOval({left:72,top:50,right:236,bottom:228}, true);
```

The following example draws the oval without a stroke:

```
f1.getDocumentDOM().addNewOval({left:72,top:50,right:236,bottom:228}, false, true);
```

### See also

[document.addNewPrimitiveOval\(\)](#)

## document.addNewPrimitiveOval()

### Availability

Flash CS4 Professional.

### Usage

```
document.addNewPrimitiveOval( boundingRectangle [, bSuppressFill [, bSuppressStroke ]] )
```

### Parameters

**boundingRectangle** A rectangle that specifies the bounds within which the new oval primitive is added. For information on the format of *boundingRectangle*, see [document.addNewRectangle\(\)](#).

**bSuppressFill** A Boolean value that, if set to `true`, causes the method to create the oval without a fill. The default value is `false`. This parameter is optional.

**bSuppressStroke** A Boolean value that, if set to `true`, causes the method to create the oval without a stroke. The default value is `false`. This parameter is optional.

### Returns

Nothing.

### Description

Method; adds a new oval primitive fitting into the specified bounds. This method performs the same operation as the Oval Primitive tool. The oval primitive uses the document's current default stroke and fill attributes and is added on the current frame and layer. If both *bSuppressFill* and *bSuppressStroke* are set to `true`, the method has no effect.

### Example

The following example adds oval primitives within the specified coordinates, with and without fill and stroke:

```
// Add an oval primitive with fill and stroke
f1.getDocumentDOM().addNewPrimitiveOval({left:0,top:0,right:100,bottom:100});
// Add an oval primitive without a fill
f1.getDocumentDOM().addNewPrimitiveOval({left:100,top:100,right:200,bottom:200}, true);
// Add an oval primitive without a stroke
f1.getDocumentDOM().addNewPrimitiveOval({left:200,top:200,right:300,bottom:300},false,true);
```

**See also**[document.addNewOval\(\)](#)

## document.addNewPrimitiveRectangle()

**Availability**

Flash CS4 Professional.

**Usage**

```
document.addNewPrimitiveRectangle( boundingRectangle, roundness, [, bSuppressFill [, bSuppressStroke ]] )
```

**Parameters**

**rect** A rectangle that specifies the bounds within which the new rectangle primitive is added. For information on the format of *boundingRectangle*, see [document.addNewRectangle\(\)](#).

**roundness** An integer between 0 and 999 that represents the number of points used to specify how much the corners should be rounded.

**bSuppressFill** A Boolean value that, if set to `true`, causes the method to create the rectangle without a fill. The default value is `false`. This parameter is optional.

**bSuppressStroke** A Boolean value that, if set to `true`, causes the method to create the rectangle without a stroke. The default value is `false`. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; adds a new rectangle primitive fitting into the specified bounds. This method performs the same operation as the Rectangle Primitive tool. The rectangle primitive uses the document's current default stroke and fill attributes and is added on the current frame and layer. If both *bSuppressFill* and *bSuppressStroke* are set to `true`, the method has no effect.

**Example**

The following example adds rectangle primitives within the specified coordinates, with and without fill and stroke, and with different amounts of roundness:

```
// Add a rectangle primitive with fill and stroke
fl.getDocumentDOM().addNewPrimitiveRectangle({left:0,top:0,right:100,bottom:100}, 0);
// Add a rectangle primitive without a fill
fl.getDocumentDOM().addNewPrimitiveRectangle({left:100,top:100,right:200,bottom:200}, 20,
true);
// Add a rectangle primitive without a stroke
fl.getDocumentDOM().addNewPrimitiveRectangle({left:200,top:200,right:300,bottom:300},
50,false,true);
```

**See also**[document.addNewRectangle\(\)](#)

## document.addNewPublishProfile()

### Availability

Flash MX 2004.

### Usage

```
document.addNewPublishProfile([profileName])
```

### Parameters

**profileName** The unique name of the new profile. If you do not specify a name, a default name is provided. This parameter is optional.

### Returns

An integer that is the index of the new profile in the profiles list. Returns -1 if a new profile cannot be created.

### Description

Method; adds a new publish profile and makes it the current one.

### Example

The following example adds a new publish profile with a default name and then displays the name of the profile in the Output panel:

```
f1.getDocumentDOM().addNewPublishProfile();
f1.outputPanel.trace(f1.getDocumentDOM().currentPublishProfile);
```

The following example adds a new publish profile with the name "my\_profile":

```
f1.getDocumentDOM().addNewPublishProfile("my_profile");
```

### See also

[document.deletePublishProfile\(\)](#)

## document.addNewRectangle()

### Availability

Flash MX 2004.

### Usage

```
document.addNewRectangle(boundingRectangle, roundness
[, bSuppressFill [, bSuppressStroke]])
```

### Parameters

**boundingRectangle** A rectangle that specifies the bounds within which the new rectangle is added, in the format {left:value1,top:value2,right:value3,bottom:value4}. The **left** and **top** values specify the location of the upper left corner (e.g., **left:0,top:0** represents the upper left corner of the Stage) and the **right** and **bottom** values specify the location of the lower-right corner. Therefore, the width of the rectangle is the difference in value between **left** and **right**, and the height of the rectangle is the difference in value between **top** and **bottom**.

In other words, the rectangle bounds do not all correspond to the values shown in the Property inspector. The `left` and `top` values correspond to the X and Y values in the Property inspector, respectively. However, the `right` and `bottom` values don't correspond to the W and H values in the Property inspector. For example, consider a rectangle with the following bounds:

```
{left:10,top:10,right:50,bottom:100}
```

This rectangle would display the following values in the Property inspector:

`X = 10, Y = 10, W = 40, H = 90`

**roundness** An integer value from 0 to 999 that specifies the roundness to use for the corners. The value is specified as number of points. The greater the value, the greater the roundness.

**bSuppressFill** A Boolean value that, if set to `true`, causes the method to create the shape without a fill. The default value is `false`. This parameter is optional.

**bSuppressStroke** A Boolean value that, if set to `true`, causes the method to create the rectangle without a stroke. The default value is `false`. This parameter is optional.

### Returns

Nothing.

### Description

Method; adds a new rectangle or rounded rectangle, fitting it into the specified bounds. This method performs the same operation as the Rectangle tool. The method uses the document's current default stroke and fill attributes and adds the rectangle on the current frame and layer. If both `bSuppressFill` and `bSuppressStroke` are set to `true`, the method has no effect.

### Example

The following example adds a new rectangle with no rounding on the corners within the specified coordinates; it is 100 pixels in width and in height:

```
f1.getDocumentDOM().addNewRectangle({left:0,top:0,right:100,bottom:100},0);
```

The following example adds a new rectangle with no rounding on the corners and without a fill; it is 100 pixels in width and 200 in height:

```
f1.getDocumentDOM().addNewRectangle({left:10,top:10,right:110,bottom:210},0, true);
```

The following example adds a new rectangle with no rounding on the corners and without a stroke; it is 200 pixels in width and 100 in height:

```
f1.getDocumentDOM().addNewRectangle({left:20,top:20,right:220,bottom:120},0, false, true);
```

### See also

[document.addNewPrimitiveRectangle\(\)](#)

## document.addNewScene()

### Availability

Flash MX 2004.

**Usage**

```
document.addNewScene( [name] )
```

**Parameters**

**name** Specifies the name of the scene. If you do not specify a name, a new scene name is generated.

**Returns**

A Boolean value: `true` if the scene is added successfully; `false` otherwise.

**Description**

Method; adds a new scene ([Timeline object](#)) as the next scene after the currently selected scene and makes the new scene the currently selected scene. If the specified scene name already exists, the scene is not added and the method returns an error.

**Example**

The following example adds a new scene named `myScene` after the current scene in the current document. The variable `success` will be `true` when the new scene is created; `false` otherwise.

```
var success = fl.getDocumentDOM().addNewScene("myScene");
```

The following example adds a new scene using the default naming convention. If only one scene exists, the newly created scene is named "Scene 2".

```
fl.getDocumentDOM().addNewScene();
```

## document.addNewText()

**Availability**

Flash MX 2004; optional `text` parameter added in Flash CS3 Professional.

**Usage**

```
document.addNewText(boundingRectangle [, text ])
```

**Parameters**

**boundingRectangle** Specifies the size and location of the text field. For information on the format of `boundingRectangle`, see [document.addNewRectangle\(\)](#).

**text** An optional string that specifies the text to place in the field. If you omit this parameter, the selection in the Tools panel switches to the Text tool. Therefore, if you don't want the selected tool to change, pass a value for `text`.

**Returns**

Nothing.

**Description**

Method; inserts a new text field and optionally places text into the field. If you omit the `text` parameter, you can call [document.setTextString\(\)](#) to populate the text field.

**Example**

The following example creates a new text field in the upper left corner of the Stage and sets the text string to "Hello World":

```
f1.getDocumentDOM().addNewText({left:0, top:0, right:100, bottom:100} , "Hello World!" );
f1.getDocumentDOM().setTextString('Hello World!');
```

**See also**

[document.setTextString\(\)](#)

## document.align()

**Availability**

Flash MX 2004.

**Usage**

```
document.align(alignmode [, bUseDocumentBounds])
```

**Parameters**

**alignmode** A string that specifies how to align the selection. Acceptable values are "left", "right", "top", "bottom", "vertical center", and "horizontal center".

**bUseDocumentBounds** A Boolean value that, if set to `true`, causes the method to align to the bounds of the document. Otherwise, the method uses the bounds of the selected objects. The default is `false`. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; aligns the selection.

**Example**

The following example aligns objects to the left and to the Stage. This is equivalent to turning on the To Stage setting in the Align panel and clicking the Align to Left button:

```
f1.getDocumentDOM().align("left", true);
```

**See also**

[document.distribute\(\)](#), [document.getAlignToDocument\(\)](#), [document.setAlignToDocument\(\)](#)

## document.allowScreens() - dropped

**Availability**

Flash MX 2004. *Dropped in Flash Professional CC.*

**Usage**

```
document.allowScreens()
```

**Parameters**

None.

**Returns**

A Boolean value: `true` if `document.screenOutline` can be used safely; `false` otherwise.

**Description**

*Dropped in Flash Professional CC.*

**Example**

The following example determines whether `screens` methods can be used in the current document:

```
if(f1.getDocumentDOM().allowScreens()) {  
    f1.trace("screen outline is available.");  
}  
else {  
    f1.trace("whoops, no screens.");  
}
```

**See also**

[document.screenOutline - dropped](#)

## document.arrange()

**Availability**

Flash MX 2004.

**Usage**

```
document.arrange(arrangeMode)
```

**Parameters**

**arrangeMode** Specifies the direction in which to move the selection. Acceptable values are "back", "backward", "forward", and "front". It provides the same capabilities as these options provide on the Modify > Arrange menu.

**Returns**

Nothing.

**Description**

Method; arranges the selection on the Stage. This method applies only to non-shape objects.

**Example**

The following example moves the current selection to the front:

```
f1.getDocumentDOM().arrange("front");
```

## document.as3AutoDeclare

### Availability

Flash CS3 Professional.

### Usage

```
document.as3AutoDeclare
```

### Description

Property; a Boolean value that describes whether the instances placed on the Stage are automatically added to user-defined timeline classes. The default value is `true`.

### Example

The following example specifies that instances placed on the Stage in the current document must be manually added to user-defined timeline classes.

```
f1.getDocumentDOM().as3AutoDeclare=false;
```

## document.as3Dialect

### Availability

Flash CS3 Professional.

### Usage

```
document.as3Dialect
```

### Description

Property; a string that describes the ActionScript 3.0 “dialect” being used in the specified document. The default value is `"AS3"`. If you wish to allow prototype classes, as permitted in earlier ECMAScript specifications, set this value to `"ES"`.

### Example

The following example specifies that the dialect being used in the current document is ECMAScript:

```
f1.getDocumentDOM().as3Dialect="ES";
```

### See also

[document.asVersion](#)

## document.as3ExportFrame

### Availability

Flash CS3 Professional.

### Usage

```
document.as3ExportFrame
```

**Description**

Property; an integer that specifies in which frame to export ActionScript 3.0 classes. By default, classes are exported in Frame 1.

**Example**

The following example changes the frame in which classes are exported from 1 (the default) to 5.

```
var myDocument = fl.getDocumentDOM();
fl.outputPanel.trace("'Export classes in frame:' value before modification is " +
myDocument.as3ExportFrame);
myDocument.as3ExportFrame = 5;
fl.outputPanel.trace("'Export classes in frame:' value after modification is " +
myDocument.as3ExportFrame);
```

## document.as3StrictMode

**Availability**

Flash CS3 Professional.

**Usage**

```
document.as3StrictMode
```

**Description**

Property; a Boolean value that specifies whether the ActionScript 3.0 compiler should compile with the Strict Mode option turned on (`true`) or off (`false`). Strict Mode causes warnings to be reported as errors, which means that compilation will not succeed if those errors exist. The default value is `true`.

**Example**

The following example turns off the Strict Mode compiler option.

```
var myDocument = fl.getDocumentDOM();
fl.outputPanel.trace("Strict Mode value before modification is " + myDocument.as3StrictMode);
myDocument.as3StrictMode = false;
fl.outputPanel.trace("Strict Mode value after modification is " + myDocument.as3StrictMode);
```

**See also**

[document.as3WarningsMode](#)

## document.as3WarningsMode

**Availability**

Flash CS3 Professional.

**Usage**

```
document.as3WarningsMode
```

**Description**

Property; a Boolean value that specifies whether the ActionScript 3.0 compiler should compile with the Warnings Mode option turned on (`true`) or off (`false`). Warnings Mode causes extra warnings to be reported that are useful for discovering incompatibilities when updating ActionScript 2.0 code to ActionScript 3.0. The default value is `true`.

**Example**

The following example turns off the Warnings Mode compiler option.

```
var myDocument = fl.getDocumentDOM();
fl.outputPanel.trace("Warnings Mode value before modification is " +
myDocument.as3WarningsMode);
myDocument.as3WarningsMode = false;
fl.outputPanel.trace("Warnings Mode value after modification is " +
myDocument.as3WarningsMode);
```

**See also**

[document.as3StrictMode](#)

## document.asVersion

**Availability**

Flash CS3 Professional.

**Usage**

`document.asVersion`

**Description**

Property; an integer that specifies which version of ActionScript is being used in the specified document. Acceptable values are 1, 2, and 3.

To determine the targeted player version for the specified document, use `document.getPlayerVersion()`; this method returns a string, so it can be used by Flash® Lite™ players.

**Example**

The following example sets the version of ActionScript in the current document to ActionScript 2.0 if it is currently set as ActionScript 1.0.

```
if(f1.getDocumentDOM().asVersion == 1){
    f1.getDocumentDOM().asVersion = 2;
}
```

**See also**

[document.as3Dialect](#), [document.getPlayerVersion\(\)](#)

## document.autoLabel

### Availability

Flash MX 2004.

### Usage

```
document.autoLabel
```

### Description

Property; a Boolean value that is equivalent to the Auto Label check box in the Accessibility panel. You can use this property to tell Flash to automatically label objects on the Stage with the text associated with them.

### Example

The following example gets the value of the `autoLabel` property and displays the result in the Output panel:

```
var isAutoLabel = fl.getDocumentDOM().autoLabel;  
fl.trace(isAutoLabel);
```

The following example sets the `autoLabel` property to `true`, telling Flash to automatically label objects on the Stage:

```
fl.getDocumentDOM().autoLabel = true;
```

## document.backgroundColor

### Availability

Flash MX 2004.

### Usage

```
document.backgroundColor
```

### Description

Property; the color of the background, in one of the following formats:

- A string in the format "#RRGGBB" or "#RRGGBAA"
- A hexadecimal number in the format 0xRRGGBB
- An integer that represents the decimal equivalent of a hexadecimal number

### Example

The following example sets the background color to black:

```
fl.getDocumentDOM().backgroundColor = '#000000';
```

## document.breakApart()

### Availability

Flash MX 2004.

**Usage**

```
document.breakApart()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; performs a break-apart operation on the current selection.

**Example**

The following example breaks apart the current selection:

```
f1.getDocumentDOM().breakApart();
```

## document.canEditSymbol()

**Availability**

Flash MX 2004.

**Usage**

```
document.canEditSymbol()
```

**Parameters**

None.

**Returns**

A Boolean value: `true` if the Edit Symbols menu and functionality are available for use; `false` otherwise.

**Description**

Method; indicates whether the Edit Symbols menu and functionality are enabled. This is not related to whether the selection can be edited. This method should not be used to test whether `f1.getDocumentDOM().enterEditMode()` is allowed.

**Example**

The following example displays in the Output panel the state of the Edit Symbols menu and functionality:

```
f1.trace("f1.getDocumentDOM().canEditSymbol() returns: " +  
f1.getDocumentDOM().canEditSymbol());
```

## document.canRevert()

**Availability**

Flash MX 2004.

**Usage**

```
document.canRevert()
```

**Parameters**

None.

**Returns**

A Boolean value: `true` if you can use the `document.revert()` or `f1.revertDocument()` methods successfully; `false` otherwise.

**Description**

Method; determines whether you can use the `document.revert()` or `f1.revertDocument()` method successfully.

**Example**

The following example checks whether the current document can revert to the previously saved version. If so, `f1.getDocumentDOM().revert()` restores the previously saved version.

```
if(f1.getDocumentDOM().canRevert()){
    f1.getDocumentDOM().revert();
}
```

## document.canTestMovie()

**Availability**

Flash MX 2004.

**Usage**

```
document.canTestMovie()
```

**Parameters**

None.

**Returns**

A Boolean value: `true` if you can use the `document.testMovie()` method successfully; `false` otherwise.

**Description**

Method; determines whether you can use the `document.testMovie()` method successfully.

**Example**

The following example tests whether `f1.getDocumentDOM().testMovie()` can be used. If so, it calls the method.

```
if(f1.getDocumentDOM().canTestMovie()){
    f1.getDocumentDOM().testMovie();
}
```

**See also**

`document.canTestScene()`, `document.testScene()`

## document.canTestScene()

### Availability

Flash MX 2004.

### Usage

```
document.canTestScene()
```

### Parameters

None.

### Returns

A Boolean value: `true` if you can use the `document.testScene()` method successfully; `false` otherwise.

### Description

Method; determines whether you can use the `document.testScene()` method successfully.

### Example

The following example first tests whether `fl.getDocumentDOM().testScene()` can be used successfully. If so, it calls the method.

```
if(f1.getDocumentDOM().canTestScene()) {
    f1.getDocumentDOM().testScene();
}
```

### See also

[document.canTestMovie\(\)](#), [document.testMovie\(\)](#)

## document.changeFilterOrder()

### Availability

Flash 8.

### Usage

```
document.changeFilterOrder(oldIndex, newIndex)
```

### Parameters

**oldIndex** An integer that represents the current zero-based index position of the filter you want to reposition in the Filters list.

**newIndex** An integer that represents the new index position of the filter in the list.

### Returns

Nothing.

**Description**

Method; changes the index of the filter in the Filters list. Any filters above or below *newIndex* are shifted up or down accordingly. For example, using the filters shown below, if you issue the command `fl.getDocumentDOM().changeFilterOrder(3, 0)`, the filters are rearranged as follows:

Before	After
<code>blurFilterdropShadowFilterglowFiltergradientBevelFilter</code>	<code>gradientBevelFilterblurFilterdropShadowFilterglowFilter</code>

If you then issue the command `fl.getDocumentDOM().changeFilterOrder(0, 2)`, the filters are rearranged as follows:

Before	After
<code>gradientBevelFilterblurFilterdropShadowFilterglowFilter</code>	<code>blurFilterdropShadowFiltergradientBevelFilterglowFilter</code>

**Example**

The following example moves the filter that is currently in the second position in the Filters list to the first position:

```
fl.getDocumentDOM().changeFilterOrder(1, 0);
```

**See also**

[document.addFilter\(\)](#), [document.disableFilter\(\)](#), [document.enableFilter\(\)](#), [document.getFilters\(\)](#), [document.removeFilter\(\)](#), [Filter object](#)

## document.clipCopy()

**Availability**

Flash MX 2004.

**Usage**

```
document.clipCopy()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; copies the current selection from the document to the Clipboard.

To copy a string to the Clipboard, use [f1.clipCopyString\(\)](#).

**Example**

The following example copies the current selection from the document to the Clipboard:

```
fl.getDocumentDOM().clipCopy();
```

**See also**

[document.clipCut\(\)](#), [document.clipPaste\(\)](#)

## document.clipCut()

**Availability**

Flash MX 2004.

**Usage**

`document.clipCut()`

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; cuts the current selection from the document and writes it to the Clipboard.

**Example**

The following example cuts the current selection from the document and writes it to the Clipboard:

```
f1.getDocumentDOM().clipCut();
```

**See also**

[document.clipCopy\(\)](#), [document.clipPaste\(\)](#), [f1.clipCopyString\(\)](#)

## document.clipPaste()

**Availability**

Flash MX 2004.

**Usage**

`document.clipPaste([bInPlace])`

**Parameters**

**bInPlace** A Boolean value that, when set to `true`, causes the method to perform a paste-in-place operation. The default value is `false`, which causes the method to perform a paste operation to the center of the document. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; pastes the contents of the Clipboard into the document.

**Example**

The following example pastes the Clipboard contents to the center of the document:

```
f1.getDocumentDOM().clipPaste();
```

The following example pastes the Clipboard contents in place in the current document:

```
f1.getDocumentDOM().clipPaste(true);
```

**See also**

[document.clipCopy\(\)](#), [document.clipCut\(\)](#), [f1.clipCopyString\(\)](#)

## document.close()

**Availability**

Flash MX 2004.

**Usage**

```
document.close([bPromptToSaveChanges])
```

**Parameters**

**bPromptToSaveChanges** A Boolean value that, when set to `true`, causes the method to prompt the user with a dialog box if there are unsaved changes in the document. If `bPromptToSaveChanges` is set to `false`, the user is not prompted to save any changed documents. The default value is `true`. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; closes the specified document.

**Example**

The following example closes the current document and prompts the user with a dialog box to save changes:

```
f1.getDocumentDOM().close();
```

The following example closes the current document without saving changes:

```
f1.getDocumentDOM().close(false);
```

## document.convertLinesToFills()

**Availability**

Flash MX 2004.

**Usage**

```
document.convertLinesToFills()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; converts lines to fills on the selected objects.

**Example**

The following example converts the current selected lines to fills:

```
f1.getDocumentDOM().convertLinesToFills();
```

## document.convertSelectionToBitmap()

**Availability**

Flash Professional CC.

**Usage**

```
document.convertSelectionToBitmap()
```

**Parameters**

None

**Returns**

Boolean.

**Description**

Method; converts selected objects in the current frame to a bitmap and inserts the bitmap into the library.

**Example**

The following example illustrates use of this method:

```
f1.getDocumentDOM().convertSelectionToBitmap();
```

## document.convertToSymbol()

**Availability**

Flash MX 2004.

**Usage**

```
document.convertToSymbol(type, name, registrationPoint)
```

**Parameters**

**type** A string that specifies the type of symbol to create. Acceptable values are "movie clip", "button", and "graphic".

**name** A string that specifies the name for the new symbol, which must be unique. You can submit an empty string to have this method create a unique symbol name for you.

**registration point** Specifies the point that represents the 0,0 location for the symbol. Acceptable values are: "top left", "top center", "top right", "center left", "center", "center right", "bottom left", "bottom center", and "bottom right".

**Returns**

An object for the newly created symbol, or null if it cannot create the symbol.

**Description**

Method; converts the selected Stage item(s) to a new symbol. For information on defining linkage and shared asset properties for a symbol, see [Item object](#).

**Example**

The following examples create a movie clip symbol with a specified name, a button symbol with a specified name, and a movie clip symbol with a default name:

```
newMc = fl.getDocumentDOM().convertToSymbol("movie clip", "mcSymbolName", "top left");
newButton = fl.getDocumentDOM().convertToSymbol("button", "btnSymbolName", "bottom right");
newClipWithDefaultName = fl.getDocumentDOM().convertToSymbol("movie clip", "", "top left");
```

## document.crop()

**Availability**

Flash 8.

**Usage**

```
document.crop()
```

**Parameters**

None.

**Returns**

None.

**Description**

Method; uses the top selected drawing object to crop all selected drawing objects underneath it. If no objects are selected, calling this method results in an error and the script breaks at that point.

**Example**

The following example crops the currently selected objects:

```
f1.getDocumentDOM().crop();
```

**See also**

[document.deleteEnvelope\(\)](#), [document.intersect\(\)](#), [document.punch\(\)](#), [document.union\(\)](#),  
[shape.isDrawingObject](#)

## document.currentPublishProfile

**Availability**

Flash MX 2004.

**Usage**

```
document.currentPublishProfile
```

**Description**

Property; a string that specifies the name of the active publish profile for the specified document.

**Example**

The following example adds a new publish profile with the default name and then displays the name of the profile in the Output panel:

```
f1.getDocumentDOM().addNewPublishProfile();
f1.outputPanel.trace(f1.getDocumentDOM().currentPublishProfile);
```

The following example changes the selected publish profile to "Default":

```
f1.getDocumentDOM().currentPublishProfile = "Default";
```

## document.currentTimeline

**Availability**

Flash MX 2004.

**Usage**

```
document.currentTimeline
```

**Description**

Property; an integer that specifies the index of the active timeline. You can set the active timeline by changing the value of this property; the effect is almost equivalent to calling [document.editScene\(\)](#). The only difference is that you don't get an error message if the index of the timeline is not valid; the property is simply not set, which causes silent failure.

**Example**

The following example displays the index of the current timeline:

```
var myCurrentTL = fl.getDocumentDOM().currentTimeline;
fl.trace("The index of the current timeline is: "+ myCurrentTL);
```

The following example changes the active timeline from the main timeline to a scene named "myScene":

```
var i = 0;
var curTimelines = fl.getDocumentDOM().timelines;
while(i < fl.getDocumentDOM().timelines.length){
    if(curTimelines[i].name == "myScene"){
        fl.getDocumentDOM().currentTimeline = i;
    }
    ++i;
}
```

#### See also

[document.getTimeline\(\)](#)

## document.debugMovie()

#### Availability

Flash Professional CS5.

#### Usage

```
document.DebugMovie( [Boolean abortIfExistsExist] )
```

#### Description

Method; Invokes the Debug Movie command on the document.

#### Parameters

**abortIfExistsExist** Boolean; the default value is false. If set to true, the debug session will not start and the .swf window will not open if there are compiler errors. Compiler warnings will not abort the command.

#### Example

The following example opens the current document in debug mode, but aborts the operation if compiler errors exist:

```
fl.getDocumentDOM().debugMovie(1);
```

## document.deleteEnvelope()

#### Availability

Flash 8.

#### Usage

```
document.deleteEnvelope()
```

#### Parameters

None.

**Returns**

None.

**Description**

Method; deletes the envelope (bounding box that contains one or more objects) from the selected objects. If no objects are selected, calling this method results in an error and the script breaks at that point.

**Example**

The following example deletes the envelope from the selected objects:

```
f1.getDocumentDOM().deleteEnvelope();
```

**See also**

[document.crop\(\)](#), [document.intersect\(\)](#), [document.punch\(\)](#), [document.union\(\)](#), [shape.isDrawingObject](#)

## document.deletePublishProfile()

**Availability**

Flash MX 2004.

**Usage**

```
document.deletePublishProfile()
```

**Parameters**

None.

**Returns**

An integer that is the index of the new current profile. If a new profile is not available, the method leaves the current profile unchanged and returns its index.

**Description**

Method; deletes the currently active profile, if there is more than one. There must be at least one profile left.

**Example**

The following example deletes the currently active profile, if there is more than one, and displays the index of the new currently active profile:

```
alert(f1.getDocumentDOM().deletePublishProfile());
```

**See also**

[document.addNewPublishProfile\(\)](#)

## document.deleteScene()

### Availability

Flash MX 2004.

### Usage

```
document.deleteScene()
```

### Parameters

None.

### Returns

A Boolean value: `true` if the scene is successfully deleted; `false` otherwise.

### Description

Method; deletes the current scene ([Timeline object](#)) and, if the deleted scene was not the last one, sets the next scene as the current Timeline object. If the deleted scene was the last one, it sets the first object as the current Timeline object. If only one Timeline object (scene) exists, it returns the value `false`.

### Example

Assuming there are three scenes (`Scene0`, `Scene1`, and `Scene2`) in the current document, the following example makes `Scene2` the current scene and then deletes it:

```
f1.getDocumentDOM().editScene(2);
var success = f1.getDocumentDOM().deleteScene();
```

## document.deleteSelection()

### Availability

Flash MX 2004.

### Usage

```
document.deleteSelection()
```

### Parameters

None.

### Returns

Nothing.

### Description

Method; deletes the current selection on the Stage. Displays an error message if there is no selection.

### Example

The following example deletes the current selection in the document:

```
f1.getDocumentDOM().deleteSelection();
```

## document.description

### Availability

Flash MX 2004.

### Usage

```
document.description
```

### Description

Property; a string that is equivalent to the Description field in the Accessibility panel. The description is read by the screen reader.

### Example

The following example sets the description of the document:

```
f1.getDocumentDOM().description= "This is the main movie";
```

The following example gets the description of the document and displays it in the Output panel:

```
f1.trace(f1.getDocumentDOM().description);
```

## document.disableAllFilters()

### Availability

Flash 8.

### Usage

```
document.disableAllFilters()
```

### Parameters

None.

### Returns

Nothing.

### Description

Method; disables all filters on the selected objects.

### Example

The following example disables all filters on the selected objects:

```
f1.getDocumentDOM().disableAllFilters();
```

### See also

[document.addFilter\(\)](#), [document.changeFilterOrder\(\)](#), [document.disableFilter\(\)](#),  
[document.disableOtherFilters\(\)](#), [document.enableAllFilters\(\)](#), [document.getFilters\(\)](#),  
[document.removeAllFilters\(\)](#), [Filter object](#)

## document.disableFilter()

### Availability

Flash 8.

### Usage

```
document.disableFilter(filterIndex)
```

### Parameters

**filterIndex** An integer representing the zero-based index of the filter in the Filters list.

### Returns

Nothing.

### Description

Method; disables the specified filter in the Filters list.

### Example

The following example disables the first and third filters (index values of 0 and 2) in the Filters list from the selected objects:

```
f1.getDocumentDOM().disableFilter(0);  
f1.getDocumentDOM().disableFilter(2);
```

### See also

[document.addFilter\(\)](#), [document.changeFilterOrder\(\)](#), [document.disableAllFilters\(\)](#),  
[document.disableOtherFilters\(\)](#), [document.enableFilter\(\)](#), [document.getFilters\(\)](#),  
[document.removeFilter\(\)](#), [Filter object](#)

## document.disableOtherFilters()

### Availability

Flash 8.

### Usage

```
document.disableOtherFilters(enabledFilterIndex)
```

### Parameters

**enabledFilterIndex** An integer representing the zero-based index of the filter that should remain enabled after other filters are disabled.

### Returns

Nothing.

### Description

Method; disables all filters except the one at the specified position in the Filters list.

**Example**

The following example disables all filters except the second filter in the list (index value of 1):

```
f1.getDocumentDom().disableOtherFilters(1);
```

**See also**

[document.addFilter\(\)](#), [document.changeFilterOrder\(\)](#), [document.disableAllFilters\(\)](#),  
[document.disableFilter\(\)](#), [document.enableFilter\(\)](#), [document.getFilters\(\)](#),  
[document.removeFilter\(\)](#), [Filter object](#)

## document.distribute()

**Availability**

Flash MX 2004.

**Usage**

```
document.distribute(distributemode [, bUseDocumentBounds])
```

**Parameters**

**distributemode** A string that specifies where to distribute the selected objects. Acceptable values are "left edge", "horizontal center", "right edge", "top edge", "vertical center", and "bottom edge".

**bUseDocumentBounds** A Boolean value that, when set to `true`, distributes the selected objects using the bounds of the document. Otherwise, the method uses the bounds of the selected objects. The default is `false`.

**Returns**

Nothing.

**Description**

Method; distributes the selection.

**Example**

The following example distributes the selected objects by their top edges:

```
f1.getDocumentDOM().distribute("top edge");
```

The following example distributes the selected objects by their top edges and expressly sets the *bUseDocumentBounds* parameter:

```
f1.getDocumentDOM().distribute("top edge", false);
```

The following example distributes the selected objects by their top edges, using the bounds of the document:

```
f1.getDocumentDOM().distribute("top edge", true);
```

**See also**

[document.getAlignToDocument\(\)](#), [document.setAlignToDocument\(\)](#)

## document.distributeToKeyframes()

### Availability

Flash Professional CC.

### Usage

```
document.distributeToKeyframes()
```

### Parameters

None.

### Returns

Nothing.

### Description

Method; performs a distribute-to-keyframes operation on the current selection—equivalent to selecting Distribute to KeyFrames. A new keyframe is created for every object. New keyframes are created on the active layer immediately after the active frame

### Example

The following example distributes the current selection to keyframes:

```
if(f1.getDocumentDOM().canDistributeToKeyframes())
f1.getDocumentDOM().distributeToKeyframes();
```

## document.distributeToLayers()

### Availability

Flash MX 2004.

### Usage

```
document.distributeToLayers()
```

### Parameters

None.

### Returns

Nothing.

### Description

Method; performs a distribute-to-layers operation on the current selection—equivalent to selecting Distribute to Layers. This method displays an error if there is no selection.

### Example

The following example distributes the current selection to layers:

```
f1.getDocumentDOM().distributeToLayers();
```

## document.docClass

### Availability

Flash CS3 Professional.

### Usage

```
document.docClass
```

### Description

Property; a string that specifies the top-level ActionScript 3.0 class associated with the document. If the document isn't configured to use ActionScript 3.0, this property is ignored.

### Example

The following example specifies that the ActionScript 3.0 class associated with the document is com.mycompany.ManagerClass, which is defined in com/mycompany/ManagerClass.as:

```
var myDocument = fl.getDocumentDOM();
// set the property
myDocument.docClass = "com.mycompany.ManagerClass";
// get the property
fl.outputPanel.trace("document.docClass has been set to " + myDocument.docClass);
```

### See also

[item.linkageBaseClass](#)

## document.documentElementHasData()

### Availability

Flash MX 2004.

### Usage

```
document.documentElementHasData(name)
```

### Parameters

**name** A string that specifies the name of the data to check.

### Returns

A Boolean value: `true` if the document has persistent data; `false` otherwise.

### Description

Method; checks the document for persistent data with the specified name.

### Example

The following example checks the document for persistent data with the name "myData":

```
var hasData = fl.getDocumentDOM().documentHasData("myData");
```

**See also**

`document.addDataToDocument()`, `document.getDataFromDocument()`,  
`document.removeDataFromDocument()`

## document.duplicatePublishProfile()

**Availability**

Flash MX 2004.

**Usage**

```
document.duplicatePublishProfile([profileName])
```

**Parameters**

**profileName** A string that specifies the unique name of the duplicated profile. If you do not specify a name, the method uses the default name. This parameter is optional.

**Returns**

An integer that is the index of the new profile in the profile list. Returns -1 if the profile cannot be duplicated.

**Description**

Method; duplicates the currently active profile and gives the duplicate version focus.

**Example**

The following example duplicates the currently active profile and displays the index of the new profile in the Output panel:

```
f1.trace(f1.getDocumentDOM().duplicatePublishProfile("dup_profile"));
```

## document.duplicateScene()

**Availability**

Flash MX 2004.

**Usage**

```
document.duplicateScene()
```

**Parameters**

None.

**Returns**

A Boolean value: `true` if the scene is duplicated successfully; `false` otherwise.

**Description**

Method; makes a copy of the currently selected scene, giving the new scene a unique name and making it the current scene.

**Example**

The following example duplicates the second scene in the current document:

```
f1.getDocumentDOM().editScene(1); //Set the middle scene to current scene.  
var success = f1.getDocumentDOM().duplicateScene();
```

## document.duplicateSelection()

**Availability**

Flash MX 2004.

**Usage**

```
document.duplicateSelection()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; duplicates the selection on the Stage.

**Example**

The following example duplicates the current selection, which is similar to Alt-clicking and then dragging an item:

```
f1.getDocumentDOM().duplicateSelection();
```

## document.editScene()

**Availability**

Flash MX 2004.

**Usage**

```
document.editScene(index)
```

**Parameters**

**index** A zero-based integer that specifies which scene to edit.

**Returns**

Nothing.

**Description**

Method; makes the specified scene the currently selected scene for editing.

**Example**

Assuming that there are three scenes (`Scene0`, `Scene1`, and `Scene2`) in the current document, the following example makes `Scene2` the current scene and then deletes it:

```
f1.getDocumentDOM().editScene(2);  
f1.getDocumentDOM().deleteScene();
```

## document.enableAllFilters()

**Availability**

Flash 8.

**Usage**

```
document.enableAllFilters()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; enables all the filters on the Filters list for the selected objects.

**Example**

The following example enables all the filters on the Filters list for the selected objects:

```
f1.getDocumentDOM().enableAllFilters();
```

**See also**

[document.addFilter\(\)](#), [document.changeFilterOrder\(\)](#), [document.disableAllFilters\(\)](#),  
[document.enableFilter\(\)](#), [document.getFilters\(\)](#), [document.removeAllFilters\(\)](#), [Filter object](#)

## document.enableFilter()

**Availability**

Flash 8.

**Usage**

```
document.enableFilter(filterIndex)
```

**Parameters**

**filterIndex** An integer specifying the zero-based index of the filter in the Filters list to enable.

**Returns**

Nothing.

**Description**

Method; enables the specified filter for the selected objects.

**Example**

The following example enables the second filter of the selected objects:

```
f1.getDocumentDOM().enableFilter(1);
```

**See also**

[document.addFilter\(\)](#), [document.changeFilterOrder\(\)](#), [document.disableFilter\(\)](#),  
[document.enableAllFilters\(\)](#), [document.getFilters\(\)](#), [document.removeFilter\(\)](#), [Filter object](#)

## document.enterEditMode()

**Availability**

Flash MX 2004.

**Usage**

```
document.enterEditMode([editMode])
```

**Parameters**

**editMode** A string that specifies the editing mode. Acceptable values are "inPlace" or "newWindow". If no parameter is specified, the default is symbol-editing mode. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; switches the authoring tool into the editing mode specified by the parameter. If no parameter is specified, the method defaults to symbol-editing mode, which has the same result as right-clicking the symbol to invoke the context menu and selecting Edit.

**Example**

The following example puts Flash in edit-in-place mode for the currently selected symbol:

```
f1.getDocumentDOM().enterEditMode('inPlace');
```

The following example puts Flash in edit-in-new-window mode for the currently selected symbol:

```
f1.getDocumentDOM().enterEditMode('newWindow');
```

**See also**

[document.exitEditMode\(\)](#)

## document.exitEditMode()

### Availability

Flash MX 2004.

### Usage

```
document.exitEditMode()
```

### Parameters

None.

### Returns

Nothing.

### Description

Method; exits from symbol-editing mode and returns focus to the next level up from the editing mode. For example, if you are editing a symbol inside another symbol, this method takes you up a level from the symbol you are editing, into the parent symbol.

### Example

The following example exits symbol-editing mode:

```
f1.getDocumentDOM().exitEditMode();
```

### See also

[document.enterEditMode\(\)](#)

## document.exportInstanceToLibrary()

### Availability

Flash CS6.

### Usage

```
document.exportInstanceToLibrary(frameNumber, bitmapName)
```

### Parameters

**frameNumber** Integer indicating the frame to be exported.

**bitmapName** A string representing the name of the bitmap to be added to the Library.

### Returns

Nothing.

### Description

Method; Exports a selected instance of a movie clip, graphic, or button symbol on the Stage to a bitmap in Library.

**Example**

The following example exports the selected item on frame 1 to the library and assigns the new bitmap the name "myTestBitmap":

```
f1.getDocumentDOM().exportInstanceToLibrary(1, "myTestBitmap");
```

## document.exportInstanceToPNGSequence()

**Availability**

Flash CS6.

**Usage**

```
document.exportInstanceToPNGSequence(outputURI, startFrameNum, endFrameNum, matrix)
```

**Parameters**

**outputURI** String: The URI to export the PNG Sequence files to. This URI must reference a local file. Example: file:///c|/tests/mytest.png.

**startFrameNum** Optional. An integer indicating the first frame to be exported. The default is 1.

**endFrameNum** Optional. An Integer indicating the last frame to be exported. The default is 99999.

**matrix** Optional. A matrix to be appended to the exported PNG sequence.

**Returns**

Nothing.

**Description**

Method; Exports a selected instance of a movie clip, graphic, or button symbol on the Stage to a series of PNG files on disk. If no **startFrameNum** or **endFrameNum** is given, the output includes all frames in the Timeline.

**Example**

The following example exports the entire Timeline to a numbered PNG sequence starting with the file name "myTest.png":

```
f1.getDocumentDOM().exportInstanceToPNGSequence("file:///c|/tests/mytest.png");
```

## document.exportPNG()

**Availability**

Flash 8.

**Usage**

```
document.exportPNG([fileURI [, bCurrentPNGSettings [, bCurrentFrame]]])
```

**Parameters**

**fileURI** A string, expressed as a file:/// URI, that specifies the filename for the exported file. If *fileURI* is an empty string or is not specified, Flash displays the Export Movie dialog box.

**bCurrentPNGSettings** A Boolean value that specifies whether to use the current PNG publish settings (`true`) or to display the Export PNG dialog box (`false`). This parameter is optional. The default value is `false`.

**bCurrentFrame** A Boolean value that specifies whether to export only the current frame (`true`) or to export all frames, with each frame as a separate PNG file (`false`). This parameter is optional. The default value is `false`.

### Returns

A Boolean value of `true` if the file is successfully exported as a PNG file; `false` otherwise.

### Description

Method; exports the document as one or more PNG files. If `fileURI` is specified and the file already exists, it is overwritten without warning.

**Note:** If `fileURI` is empty and `bCurrentFrame` is `true`, the Export Movie dialog box does not display and Flash saves the exported PNG file in the same location as the `.fla` file.

**Note:** If `bCurrentPNGSettings` is `false` and `bCurrentFrame` is `true`, the Export PNG dialog box does not display and Flash uses the current PNG publish settings.

### Example

The following example exports the current frame in the current document to `myFile.png`, using the current PNG publish settings:

```
f1.getDocumentDOM().exportPNG("file:///C|/myProject/myFile.png", true, true);
```

## document.exportPublishProfile()

### Availability

Flash MX 2004.

### Usage

```
document.exportPublishProfile(fileURI)
```

### Parameters

**fileURI** A string, expressed as a file:/// URI, that specifies the path of the XML file to which the profile is exported.

### Returns

Nothing.

### Description

Method; exports the currently active profile to an XML file.

### Example

The following example exports the currently active profile to the file named `profile.xml` in the folder /Documents and Settings/username/Desktop on the C drive:

```
f1.getDocumentDOM().exportPublishProfile('file:///C|/Documents and Settings/username/Desktop/profile.xml');
```

**See also**

[document.exportPublishProfileString\(\)](#), [document.importPublishProfile\(\)](#)

## document.exportPublishProfileString()

**Availability**

Flash CS4 Professional.

**Usage**

```
document.exportPublishProfileString( [profileName] )
```

**Parameters**

**profileName** A string that specifies the name of the profile to export to an XML string. This parameter is optional.

**Returns**

An XML string.

**Description**

Method: returns a string that specifies, in XML format, the specified profile. If you don't pass a value for *profileName*, the current profile is exported.

**Example**

The following example stores an XML string that represents the current profile in a variable named `profileXML` and then displays it in the Output panel:

```
var profileXML=f1.getDocumentDOM().exportPublishProfileString();  
f1.trace(profileXML);
```

**See also**

[document.exportPublishProfile\(\)](#), [document.importPublishProfileString\(\)](#)

## document.exportSWF()

**Availability**

Flash MX 2004.

**Usage**

```
document.exportSWF([fileURI [, bCurrentSettings]])
```

**Parameters**

**fileURI** A string, expressed as a file:/// URI, that specifies the name of the exported file. If *fileURI* is empty or not specified, Flash displays the Export Movie dialog box. This parameter is optional.

**bCurrentSettings** A Boolean value that, when set to `true`, causes Flash to use current SWF publish settings. Otherwise, Flash displays the Export Flash Player dialog box. The default is `false`. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; exports the document in the Flash SWF format.

**Example**

The following example exports the document to the specified file location with the current publish settings:

```
f1.getDocumentDOM().exportSWF("file:///C|/Documents and  
Settings/joe_user/Desktop/qwerty.swf", true);
```

The following example displays the Export Movie dialog box and the Export Flash Player dialog box and then exports the document based on the specified settings:

```
f1.getDocumentDOM().exportSWF("", false);
```

The following example displays the Export Movie dialog box and the Export Flash Player dialog box and then exports the document based on the specified settings:

```
f1.getDocumentDOM().exportSWF();
```

## document.exportVideo()

**Availability**

Flash Professional CC.

**Usage**

```
exportVideo( fileURI [, convertInAdobeMediaEncoder] [, transparent] [, stopAtFrame]  
[, stopAtFrameOrTime] )
```

**Parameters**

**fileURI** A string, expressed as a file:/// URI, that specifies the fully qualified path to which the video is saved.

**convertInAdobeMediaEncoder** A boolean value that specifies whether or not to send the recorded video to Adobe Media Encoder. The default value is `true`, which sends the video to Adobe Media Encoder. This parameter is optional.

**transparent** A boolean value that specifies whether or not the background should be included in the video. The default value is `false`, which includes the movie background in the video. This parameter is optional.

**stopAtFrame** A boolean value that specifies whether the video should be recorded until it reaches a certain frame or a specific time. The default value is `true`, stop when a certain frame is reached. This parameter is optional.

**stopAtFrameOrTime** If `stopAtFrame` is `true`, this is an int specifying the number of frames to record. If `stopAtFrame` is `false`, this is the number of milliseconds to record. The default value is 0 which, if `stopAtFrame` is `true`, will record all the frames in the main timeline. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; exports a video from the document and optionally sends it to Adobe Media Encoder to convert the video.

**Example**

The following example illustrates the use of this method:

```
f1.getDocumentDOM().exportVideo("file:///C|/myProject/myVideo.mov");
```

## document.externalLibraryPath

**Availability**

Flash CS4 Professional.

**Usage**

```
document.externalLibraryPath
```

**Description**

Property; a string that contains a list of items in the document's ActionScript 3.0 External library path, which specifies the location of SWC files used as runtime shared libraries. Items in the string are delimited by semi-colons. In the authoring tool, the items are specified by choosing File > Publish Settings and then choosing ActionScript 3.0 Script Settings on the Flash tab.

**Example**

The following example sets the document's External library path to "." and "../mySWCLibrary":

```
var myDocument = f1.getDocumentDOM();
myDocument.externalLibraryPath = ".;../mySWCLibrary";
f1.trace(myDocument.externalLibraryPath);
```

**See also**

[document.libraryPath](#), [document.sourcePath](#), [f1.externalLibraryPath](#)

## document.forceSimple

**Availability**

Flash MX 2004.

**Usage**

```
document.forceSimple
```

**Description**

Property; a Boolean value that specifies whether the children of the specified object are accessible. This is equivalent to the inverse logic of the Make Child Objects Accessible setting in the Accessibility panel. That is, if `forceSimple` is `true`, it is the same as the Make Child Object Accessible option being unchecked. If `forceSimple` is `false`, it is the same as the Make Child Object Accessible option being checked.

**Example**

The following example sets the `areChildrenAccessible` variable to the value of the `forceSimple` property. A value of `false` means the children are accessible.

```
var areChildrenAccessible = fl.getDocumentDOM().forceSimple;
```

The following example sets the `forceSimple` property to allow the children of the document to be accessible:

```
fl.getDocumentDOM().forceSimple = false;
```

## document.frameRate

### Availability

Flash MX 2004.

### Usage

```
document.frameRate
```

### Description

Property; a float value that specifies the number of frames displayed per second when the SWF file plays; the default is 12. Setting this property is the same as setting the default frame rate in the Document Properties dialog box (Modify > Document) in the FLA file.

### Example

The following example sets the frame rate to 25.5 frames per second:

```
fl.getDocumentDOM().frameRate = 25.5;
```

## document.getAlignToDocument()

### Availability

Flash MX 2004.

### Usage

```
document.getAlignToDocument()
```

### Parameters

None.

### Returns

A Boolean value: `true` if the preference is set to align the objects to the Stage; `false` otherwise.

### Description

Method; identical to retrieving the value of the To Stage button in the Align panel. Gets the preference that can be used for `document.align()`, `document.distribute()`, `document.match()`, and `document.space()` methods on the document.

### Example

The following example retrieves the value of the To Stage button in the Align panel. If the return value is `true`, the To Stage button is active; otherwise, it is not.

```
var isAlignToDoc = fl.getDocumentDOM().getAlignToDocument();
fl.getDocumentDOM().align("left", isAlignToDoc);
```

**See also**

[document.setAlignToDocument\(\)](#)

## document.getBlendMode()

**Availability**

Flash 8.

**Usage**

```
document.getBlendMode()
```

**Parameters**

None.

**Returns**

A string that specifies the blending mode for the selected objects. If more than one object is selected and they have different blending modes, the string reflects the blending mode of the object with the highest depth.

**Note:** The return value is unpredictable if the selection contains objects that don't support blending modes, or that have a blending mode value of "normal".

**Description**

Method; returns a string that specifies the blending mode for the selected objects.

**Example**

The following example displays the name of the blending mode in the Output panel:

```
fl.trace(fl.getDocumentDom().getBlendMode());
```

## document.getCustomFill()

**Availability**

Flash MX 2004.

**Usage**

```
document.getCustomFill([objectToFill])
```

**Parameters**

**objectToFill** A string that specifies the location of the fill object. The following values are valid:

- "toolbar" returns the fill object of the Tools panel and Property inspector.
- "selection" returns the fill object of the selection.

If you omit this parameter, the default value is "selection". If there is no selection, the method returns undefined. This parameter is optional.

### Returns

The [Fill object](#) specified by the *objectToFill* parameter, if successful; otherwise, it returns undefined.

### Description

Method; retrieves the fill object of the selected shape or, if specified, of the Tools panel and Property inspector.

### Example

The following example gets the fill object of the selection and then changes the selection's color to white:

```
var fill = fl.getDocumentDOM().getCustomFill();
fill.color = '#FFFFFF';
fill.style = "solid";
fl.getDocumentDOM().setCustomFill(fill);
```

The following example returns the fill object of the Tools panel and Property inspector and then changes the color swatch to a linear gradient:

```
var fill = fl.getDocumentDOM().getCustomFill("toolbar");
fill.style = "linearGradient";
fill.colorArray = [ 0x00ff00, 0xff0000, 0x0000ff ];
fill.posArray = [0, 100, 200];
fl.getDocumentDOM().setCustomFill( fill );
```

### See also

[document.setCustomFill\(\)](#)

## document.getCustomStroke()

### Availability

Flash MX 2004.

### Usage

```
document.getCustomStroke([locationOfStroke])
```

### Parameters

**locationOfStroke** A string that specifies the location of the stroke object. The following values are valid:

- "toolbar", if set, returns the stroke object of the Tools panel and Property inspector.
- "selection", if set, returns the stroke object of the selection.

If you omit this parameter, it defaults to "selection". If there is no selection, it returns undefined. This parameter is optional.

### Returns

The [Stroke object](#) specified by the *locationOfStroke* parameter, if successful; otherwise, it returns undefined.

**Description**

Returns the stroke object of the selected shape or, if specified, of the Tools panel and Property inspector.

**Example**

The following example returns the current stroke settings of the selection and changes the stroke thickness to 2:

```
var stroke = fl.getDocumentDOM().getCustomStroke("selection");
stroke.thickness = 2;
fl.getDocumentDOM().setCustomStroke(stroke);
```

The following example returns the current stroke settings of the Tools panel and Property inspector and sets the stroke color to red:

```
var stroke = fl.getDocumentDOM().getCustomStroke("toolbar");
stroke.color = "#FF0000";
fl.getDocumentDOM().setCustomStroke(stroke);
```

**See also**

[document.setCustomStroke\(\)](#)

## document.getDataFromDocument()

**Availability**

Flash MX 2004.

**Usage**

```
document.getDataFromDocument(name)
```

**Parameters**

**name** A string that specifies the name of the data to return.

**Returns**

The specified data.

**Description**

Method; retrieves the value of the specified data. The type returned depends on the type of data that was stored.

**Example**

The following example adds an integer value of 12 to the current document and uses this method to display the value in the Output panel:

```
fl.getDocumentDOM().addDataToDocument("myData", "integer", 12);
fl.trace(fl.getDocumentDOM().getDataFromDocument("myData"));
```

**See also**

[document.addDataToDocument\(\)](#), [document.documentHasData\(\)](#), [document.removeDataFromDocument\(\)](#)

## document.getElementProperty()

### Availability

Flash MX 2004.

### Usage

```
document.getElementProperty(propertyName)
```

### Parameters

**propertyName** A string that specifies the name of the Element property for which to retrieve the value.

### Returns

The value of the specified property. Returns `null` if the property is an indeterminate state, as when multiple elements are selected with different property values. Returns `undefined` if the property is not a valid property of the selected element.

### Description

Method; gets the specified `Element` property for the current selection. For a list of acceptable values, see the Property summary table for the [Element object](#).

### Example

The following example gets the name of the Element property for the current selection:

```
// elementName = the instance name of the selected object.  
var elementName = fl.getDocumentDOM().getElementProperty("name");
```

### See also

[document.createElementProperty\(\)](#)

## document.getElementTextAttr()

### Availability

Flash MX 2004.

### Usage

```
document.getElementTextAttr(attrName [, startIndex [, endIndex]])
```

### Parameters

**attrName** A string that specifies the name of the `TextAttrs` property to be returned. For a list of property names and expected values, see the Property summary table for the [TextAttrs object](#).

**startIndex** An integer that specifies the index of first character, with 0 (zero) specifying the first position. This parameter is optional.

**endIndex** An integer that specifies the index of last character. This parameter is optional.

**Returns**

If one text field is selected, the property is returned if there is only one value used within the text. Returns `undefined` if there are several values used inside the text field. If several text fields are selected, and all the text alignment values are equal, the method returns this value. If several text fields are selected, but all the text alignment values are not equal, the method returns `undefined`. If the optional arguments are not passed, these rules apply to the range of text currently selected or the whole text field if the text is not currently being edited. If only `startIndex` is passed, the property of the character to the right of the index is returned, if all the selected Text objects match values. If `startIndex` and `endIndex` are passed, the value returned reflects the entire range of characters from `startIndex` up to, but not including, `endIndex`.

**Description**

Method; gets a specific `TextAttrs` property of the selected Text objects. Selected objects that are not text fields are ignored. For a list of property names and expected values, see the Property summary table for the [TextAttrs object](#). See also [document.setTextAttr\(\)](#).

**Example**

The following example gets the size of the selected text fields:

```
f1.getDocumentDOM().getElementTextAttr("size");
```

The following example gets the color of the character at index 3 in the selected text fields:

```
f1.getDocumentDOM().getElementTextAttr("fillColor", 3);
```

The following example gets the font name of the text from index 2 up to, but not including, index 10 of the selected text fields:

```
f1.getDocumentDOM().getElementTextAttr("face", 2, 10);
```

## document.getFilters()

**Availability**

Flash 8.

**Usage**

```
document.getFilters()
```

**Parameters**

None.

**Returns**

An array that contains a list of filters applied to the currently selected objects.

**Description**

Method; returns an array that contains the list of filters applied to the currently selected objects. If multiple objects are selected and they don't have identical filters, this method returns the list of filters applied to the first selected object.

**Example**

See [document.setFilters\(\)](#).

**See also**

[document.addFilter\(\)](#), [document.changeFilterOrder\(\)](#), [document.setFilters\(\)](#), [Filter object](#)

## document.getMetadata()

**Availability**

Flash 8.

**Usage**

```
document.getMetadata()
```

**Parameters**

None.

**Returns**

A string containing the XML metadata associated with the document or an empty string if there is no metadata.

**Description**

Method; returns a string containing the XML metadata associated with the document, or an empty string if there is no metadata.

**Example**

The following example displays XML metadata from the current document in the Output panel:

```
f1.trace("XML Metadata is :" + f1.getDocumentDOM().getMetadata());
```

**See also**

[document.setMetadata\(\)](#)

## document.getMobileSettings()

**Availability**

Flash CS3 Professional.

**Usage**

```
document.getMobileSettings()
```

**Parameters**

None.

**Returns**

A string that represents the XML settings for the document. If no value has been set, returns an empty string.

**Description**

Method; returns the mobile XML settings for the document.

**Example**

The following example displays the XML settings string for the current document:

```
f1.trace(f1.getDocumentDOM().getMobileSettings());  
//traces a string like the following "<? xml version="1.0" encoding="UTF-16" standalone="no"  
?><mobileSettings> <contentType id="standalonePlayer" name="Standalone Player"/>  
<testDevices> <testDevice id="1170" name="Generic Phone" selected="yes"/> </testDevices>  
<outputMsgFiltering info="no" trace="yes" warning="yes"/> <testWindowState height="496"  
splitterClosed="No" splitterXPos="400" width="907"/> </mobileSettings>"
```

**See also**

[document.setMobileSettings\(\)](#)

## document.getPlayerVersion()

**Availability**

Flash CS3 Professional.

**Usage**

[document.getPlayerVersion\(\)](#)

**Parameters**

None.

**Returns**

A string that represents the Flash Player version specified by using [document.setPlayerVersion\(\)](#). If no value has been set, returns the value specified in the Publish Settings dialog box.

**Description**

Method; returns a string that represents the targeted player version for the specified document. For a list of values that this method can return, see [document.setPlayerVersion\(\)](#).

To determine which version of ActionScript is being targeted in the specified file, use [document.asVersion](#).

**Example**

The following example illustrates targeting specified player versions for the current document and then retrieving those values:

```
f1.getDocumentDOM().setPlayerVersion("6");  
var version = f1.getDocumentDOM().getPlayerVersion();  
f1.trace(version) // displays "6"  
f1.getDocumentDOM().setPlayerVersion("FlashPlayer10");  
var version = f1.getDocumentDOM().getPlayerVersion();  
f1.trace(version) // displays "FlashPlayer10"
```

**See also**

[document.setPlayerVersion\(\)](#)

# document.getPublishDocumentData()

## Availability

Flash Professional CC.

## Usage

```
document.getPublishDocumentData(format)
```

## Parameters

**format** A string that specifies the publishing format.

**Note:** `_EMBED_SWF_` is a special built-in publishing format for persistent data. If set, the persistent data will be embedded in the SWF file every time a document is published. The persistent data can then be accessed via ActionScript with the `.metaData` property. This requires SWF version 19 (Flash Player 11.6) and above and is only for symbol instances onstage. Other custom publishing formats may be specified for custom JSFL scripts if this method is called with the same format.

## Returns

Boolean; True if publishing of the specified persistent data is enabled for the specified format in this document. Otherwise False.

## Description

Method; Indicates whether publishing of the specified persistent data is enabled for the specified format in this document.

## Example

The following example illustrates the use of this method:

```
var doc = fl.getDocumentDOM();
// set the data
if (doc) {
    // get the first selected element
    var elem = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];
    if (elem) {
        // add persistent data "myAlign" of "left"
        elem.setPersistentData( "myAlign", "string", "left" );
        // enable publishing of persistent data to SWF for the element
        elem.setPublishPersistentData("myAlign", "_EMBED_SWF_", true);
        // enable publishing to SWF for entire document
        doc.setPublishDocumentData("_EMBED_SWF_", true);
    }
}
// example getting data
if (doc && doc.getPublishDocumentData("_EMBED_SWF_")) {
    // get the first selected element
    var elem = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];
    if (elem && elem.hasPersistentData("myAlign") && elem.getPublishPersistentData("myAlign", "_EMBED_SWF_")) {
        alert("elem.metaData.myAlign = '" + elem.getPersistentData("myAlign") + "' will be
embedded in SWF when published.");
    }
}
```

**See also**[document.setPublishDocumentData\(\)](#)

## document.getSelectionRect()

**Availability**

Flash MX 2004.

**Usage**[document.getSelectionRect\(\)](#)**Parameters**

None.

**Returns**

The bounding rectangle of the current selection, or 0 if nothing is selected. For information on the format of the return value, see [document.addNewRectangle\(\)](#).

**Description**

Method; gets the bounding rectangle of the current selection. If a selection is non-rectangular, the smallest rectangle encompassing the entire selection is returned. The rectangle is based on the document space or, when in edit mode, the registration point (also *origin point* or *zero point*) of the symbol being edited.

**Example**

The following example gets the bounding rectangle for the current selection and then displays its properties:

```
var newRect = fl.getDocumentDOM().getSelectionRect();
var outputStr = "left: " + newRect.left + " top: " + newRect.top + " right: " + newRect.right
+ " bottom: " + newRect.bottom;
alert(outputStr);
```

**See also**[document.selection](#), [document.setSelectionRect\(\)](#)

## document.getSWFPathFromProfile()

**Availability**

Flash Professional CS6.

**Usage**[document.getSWFPathFromProfile\(\)](#)**Parameters**

None.

**Returns**

The full path to the SWF file that is set in the current Publish profile.

**Description**

Method; gets the path to the SWF file that is set in the current Publish profile.

**Example**

The following example displays the full path to the SWF file as defined in the Publish profile:

```
fl.trace("the current Publish Setting's SWF file path is: " +  
fl.getDocumentDOM().getSWFPathFromProfile());
```

## document.getTelemetryForSwf()

**Availability**

Flash Professional CC.

**Usage**

```
document.getTelemetryForSwf( )
```

**Parameters**

None.

**Returns**

Returns boolean. Returns `true` if the “Enable detailed telemetry” checkbox is selected; otherwise `false`.

**Description**

Method; Indicates whether if the “Enable detailed telemetry” checkbox is selected in the Publish Settings dialog.

**Example**

The following example calls `getTelemetryFromSwf()`:

```
fl.trace("is detailed telemetry enabled for this doc: " +  
fl.getDocumentDOM().getTelemetryForSwf());
```

## document.getTextString()

**Availability**

Flash MX 2004.

**Usage**

```
document.getTextString([startIndex [, endIndex]])
```

**Parameters**

**startIndex** An integer that is an index of first character to get. This parameter is optional.

**endIndex** An integer that is an index of last character to get. This parameter is optional.

#### Returns

A string that contains the selected text.

#### Description

Method; gets the currently selected text. If the optional parameters are not passed, the current text selection is used. If text is not currently opened for editing, the whole text string is returned. If only *startIndex* is passed, the string starting at that index and ending at the end of the field is returned. If *startIndex* and *endIndex* are passed, the string starting from *startIndex* up to, but not including, *endIndex* is returned.

If there are several text fields selected, the concatenation of all the strings is returned.

#### Example

The following example gets the string in the selected text fields:

```
f1.getDocumentDOM().getTextString();
```

The following example gets the string at character index 5 in the selected text fields:

```
f1.getDocumentDOM().getTextString(5);
```

The following example gets the string from character index 2 up to, but not including, character index 10:

```
f1.getDocumentDOM().getTextString(2, 10);
```

#### See also

[document.setTextString\(\)](#)

## document.getTimeline()

#### Availability

Flash MX 2004.

#### Usage

```
document.getTimeline()
```

#### Parameters

None.

#### Returns

The current Timeline object.

#### Description

Method; retrieves the current [Timeline object](#) in the document. The current timeline can be the current scene, the current symbol being edited, or the current screen.

#### Example

The following example gets the Timeline object and returns the number of frames in the longest layer:

```
var longestLayer = fl.getDocumentDOM().getTimeline().frameCount;
fl.trace("The longest layer has" + longestLayer + "frames");
```

The following example enters edit-in-place mode for the selected symbol on the Stage and inserts a frame on the symbol's timeline.

```
fl.getDocumentDOM().enterEditMode("inPlace");
fl.getDocumentDOM().getTimeline().insertFrames();
```

The following example gets the Timeline object and displays its name:

```
var timeline = fl.getDocumentDOM().getTimeline();
alert(timeline.name);
```

#### See also

[document.currentTimeline](#), [document.timelines](#), [symbolItem.timeline](#)

## document.getTransformationPoint()

#### Availability

Flash MX 2004.

#### Usage

```
document.getTransformationPoint()
```

#### Parameters

None.

#### Returns

A point (for example, {x:10, y:20}, where x and y are floating-point numbers) that specifies the position of the transformation point (also *origin point* or *zero point*) within the selected element's coordinate system.

#### Description

Method; gets the location of the transformation point of the current selection. You can use the transformation point for commutations such as rotate and skew.

**Note:** Transformation points are relative to different locations, depending on the type of item selected. For more information, see [document.setTransformationPoint\(\)](#).

#### Example

The following example gets the transformation point for the current selection. The `transPoint.x` property gives the x coordinate of the transformation point. The `transPoint.y` property gives the y coordinate of the transformation point.

```
var transPoint = fl.getDocumentDOM().getTransformationPoint();
```

#### See also

[document.setTransformationPoint\(\)](#), [element.getTransformationPoint\(\)](#)

## document.group()

**Availability**

Flash MX 2004.

**Usage**

```
document.group()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; converts the current selection to a group.

**Example**

The following example converts the objects in the current selection to a group:

```
f1.getDocumentDOM().group();
```

**See also**

[document.unGroup\(\)](#)

## document.height

**Availability**

Flash MX 2004.

**Usage**

```
document.height
```

**Description**

Property; an integer that specifies the height of the document (Stage) in pixels.

**Example**

The following example sets the height of the Stage to 400 pixels:

```
f1.getDocumentDOM().height = 400;
```

**See also**

[document.width](#)

## document.id

### Availability

Flash CS3 Professional.

### Usage

`document.id`

### Description

Read-only property; a unique integer (assigned automatically) that identifies a document during a Flash session. Use this property in conjunction with `f1.findDocumentDOM()` to specify a particular document for an action.

### Example

The following example displays the document ID for the current document:

```
f1.trace("Current doc's internal ID is: " + f1.getDocumentDOM().id);
```

### See also

`f1.findDocumentDOM()`

## document.importFile()

### Availability

Flash 8. The `showDialog` and `showImporterUI` parameters are new in Flash Professional CC.

### Usage

```
document.importFile(fileURI [, importToLibrary [, showDialog [, showImporterUI ]]])
```

### Parameters

**fileURI** A string, expressed as a file:/// URI, that specifies the path of the file to import.

**importToLibrary** A Boolean value that specifies whether to import the file only into the document's library (`true`) or to also place a copy on the Stage (`false`). The default value is `false`.

**showDialog** A Boolean value that specifies whether to display the Import dialog box. Specifying `true` displays the import dialog. If you specify `false`, the function imports the file using specifications set in the Preferences dialog. The default is `true`.

**showImporterUI** A Boolean value that specifies whether to display errors visually (for example, using the Library Conflict dialog box). The default is `false`.

### Returns

Nothing.

### Description

Method; imports a file into a document. This method performs the same operation as the Import To Library or Import To Stage menu command. To import a publish profile, use `document.importPublishProfile()`.

**Example**

The following example lets the user browse for a file to import onto the Stage:

```
var dom = fl.getDocumentDOM();
var URI = fl.browseForFileURL("select", "Import File");
dom.importFile(URI);
```

**See also**

[fl.browseForFileURL\(\)](#)

## document.importPublishProfile()

**Availability**

Flash MX 2004.

**Usage**

```
document.importPublishProfile( fileURI )
```

**Parameters**

**fileURI** A string, expressed as a file:/// URI, that specifies the path of the XML file defining the profile to import.

**Returns**

An integer that is the index of the imported profile in the profiles list. Returns -1 if the profile cannot be imported.

**Description**

Method; imports a profile from a file.

**Example**

The following example imports the profile contained in the profile.xml file and displays its index in the profiles list:

```
alert(f1.getDocumentDOM().importPublishProfile('file:///C|/Documents and
Settings/janeUser/Desktop/profile.xml'));
```

## document.importPublishProfileString()

**Availability**

Flash CS4 Professional.

**Usage**

```
document.importPublishProfileString(xmlString)
```

**Parameters**

**xmlString** A string that contains the XML data to be imported as the current profile.

**Returns**

A Boolean value of `true` if the string was successfully imported; `false` otherwise.

**Description**

Method: imports an XML string that represents a publish profile and sets it as the current profile. To generate an XML string to import, use [document.exportPublishProfileString\(\)](#) before using this method.

**Example**

In the following example, the default profile is exported as an XML string. The standard JavaScript `replace` command is used to modify the XML string. The string is then imported, and the default ActionScript 3 output setting is set to ActionScript 1.

```
var profileXML=f1.getDocumentDOM().exportPublishProfileString('Default');
f1.trace(profileXML);
var newProfileXML = profileXML.replace("<ActionScriptVersion>3</ActionScriptVersion>",
"<ActionScriptVersion>1</ActionScriptVersion>");
f1.getDocumentDOM().importPublishProfileString(newProfileXML);
```

## document.importSWF() - dropped

**Availability**

Flash MX 2004. *Dropped in Flash Professional CC.*

**Usage**

```
document.importSWF(fileURI)
```

**Parameters**

**fileURI** A string, expressed as a file:/// URI, that specifies the file for the SWF file to import.

**Returns**

Nothing.

**Description**

*Dropped in Flash Professional CC.*

**Example**

The following example imports the "mySwf.swf" file from the Flash Configuration folder:

```
f1.getDocumentDOM().importSWF(f1.configURI+"mySwf.swf");
```

**See also**

[document.importFile\(\)](#)

## document.intersect()

**Availability**

Flash 8.

**Usage**

```
document.intersect()
```

**Parameters**

None.

**Returns**

None.

**Description**

Method; creates an intersection drawing object from all selected drawing objects. If no objects are selected, calling this method results in an error and the script breaks at that point.

**Example**

The following example creates an intersection drawing object from all selected drawing objects:

```
f1.getDocumentDOM().intersect();
```

**See also**

[document.crop\(\)](#), [document.deleteEnvelope\(\)](#), [document.punch\(\)](#), [document.union\(\)](#),  
[shape.isDrawingObject](#)

## document.library

**Availability**

Flash MX 2004.

**Usage**

```
document.library
```

**Description**

Read-only property; the [library object](#) for a document.

**Example**

The following example gets the library for the currently focused document:

```
var myCurrentLib = f1.getDocumentDOM().library;
```

Assuming the currently focused document is not `f1.documents[1]`, the following example gets the library for a nonfocused library or for a library you opened using File > Open as external library:

```
var externalLib = f1.documents[1].library;
```

## document.libraryPath

**Availability**

Flash CS4 Professional.

**Usage**

```
document.libraryPath
```

**Description**

Property; a string that contains a list of items in the document's ActionScript 3.0 Library path, which specifies the location of SWC files or folders containing SWC files. Items in the string are delimited by semi-colons. In the authoring tool, the items are specified by choosing File > Publish Settings and then choosing ActionScript 3.0 Script Settings on the Flash tab.

**Example**

The following adds the ..Files folder to the document's Library path and then displays the path Library path in the Output panel:

```
var myDoc = fl.getDocumentDOM();
fl.trace(myDoc.libraryPath);
myDoc.libraryPath = "..Files;" + myDoc.libraryPath;
fl.trace(myDoc.libraryPath);
```

**See also**

[document.externalLibraryPath](#), [document.sourcePath](#), [fl.libraryPath](#)

## document.livePreview

**Availability**

Flash MX 2004.

**Usage**

```
document.livePreview
```

**Description**

Property; a Boolean value that specifies whether Live Preview is enabled. If set to `true`, components appear on the Stage as they will appear in the published Flash content, including their approximate size. If set to `false`, components appear only as outlines. The default value is `true`.

**Example**

The following example sets Live Preview to `false`:

```
fl.getDocumentDOM().livePreview = false;
```

## document.loadCuepointXML() - dropped

**Availability**

Flash Professional CS5. Dropped in Flash Professional CC.

**Usage**

```
document.loadCuepointXML(String URI)
```

**Parameters**

**URI** String; the absolute path to the cue point XML file.

**Description**

Dropped in Flash Professional CC.

Method; loads a cue point XML file. The format and DTD of the XML file is the same as the one imported and exported by the Cue Points Property inspector. The return value is the same as the string serialized in the Cue Point property of the object containing the instance of an FLVPlayback Component.

**Example**

The following example loads the cue points XML file located at C:\\testCuePoints.xml:

```
var cuePoints = fl.getDocumentDOM().LoadCuepointXML("c:\\\\testCuePoints.xml");
```

## document.match()

**Availability**

Flash MX 2004.

**Usage**

```
document.match(bWidth, bHeight [, bUseDocumentBounds])
```

**Parameters**

**bWidth** A Boolean value that, when set to `true`, causes the method to make the widths of the selected items the same.

**bHeight** A Boolean value that, when set to `true`, causes the method to make the heights of the selected items the same.

**bUseDocumentBounds** A Boolean value that, when set to `true`, causes the method to match the size of the objects to the bounds of the document. Otherwise, the method uses the bounds of the largest object. The default is `false`. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; makes the size of the selected objects the same.

**Example**

The following example matches the width of the selected objects only:

```
fl.getDocumentDOM().match(true, false);
```

The following example matches the height only:

```
fl.getDocumentDOM().match(false, true);
```

The following example matches the width only to the bounds of the document:

```
fl.getDocumentDOM().match(true, false, true);
```

**See also**

[document.getAlignToDocument\(\)](#), [document.setAlignToDocument\(\)](#)

## document.mouseClick()

**Availability**

Flash MX 2004.

**Usage**

```
document.mouseClick(position, bToggleSel, bShiftSel)
```

**Parameters**

**position** A pair of floating-point values that specify the *x* and *y* coordinates of the click in pixels.

**bToggleSel** A Boolean value that specifies the state of the Shift key: `true` for pressed; `false` for not pressed.

**bShiftSel** A Boolean value that specifies the state of the application preference Shift select: `true` for on; `false` for off.

**Returns**

Nothing.

**Description**

Method; performs a mouse click from the Selection tool.

**Example**

The following example performs a mouse click at the specified location:

```
f1.getDocumentDOM().mouseClick({x:300, y:200}, false, false);
```

**See also**

[document.mouseDblClk\(\)](#)

## document.mouseDblClk()

**Availability**

Flash MX 2004.

**Usage**

```
document.mouseDblClk(position, bAltDown, bShiftDown, bShiftSelect)
```

**Parameters**

**position** A pair of floating-point values that specify the *x* and *y* coordinates of the click in pixels.

**bAltDown** A Boolean value that records whether the Alt key is down at the time of the event: `true` for pressed; `false` for not pressed.

**bShiftDown** A Boolean value that records whether the Shift key was down when the event occurred: `true` for pressed; `false` for not pressed.

**bShiftSelect** A Boolean value that indicates the state of the application preference Shift select: `true` for on; `false` for off.

### Returns

Nothing.

### Description

Method; performs a double mouse click from the Selection tool.

### Example

The following example performs a double mouse click at the specified location:

```
f1.getDocumentDOM().mouseDblClk({x:392.9, y:73}, false, false, true);
```

### See also

[document.mouseClick\(\)](#)

## document.moveSelectedBezierPointsBy()

### Availability

Flash MX 2004.

### Usage

```
document.moveSelectedBezierPointsBy(delta)
```

### Parameters

**delta** A pair of floating-point values that specify the *x* and *y* coordinates in pixels by which the selected Bézier points are moved. For example, passing `({x:1, y:2})` specifies a location that is to the right by one pixel and down by two pixels from the current location.

### Returns

Nothing.

### Description

Method; if the selection contains at least one path with at least one Bézier point selected, moves all selected Bézier points on all selected paths by the specified amount.

### Example

The following example moves the selected Bézier points 10 pixels to the right and 5 pixels down:

```
f1.getDocumentDOM().moveSelectedBezierPointsBy({x:10, y:5});
```

## document.moveSelectionBy()

### Availability

Flash MX 2004.

### Usage

```
document.moveSelectionBy(distanceToMove)
```

### Parameters

**distanceToMove** A pair of floating-point values that specify the *x* and *y* coordinate values by which the method moves the selection. For example, passing `{x:1,y:2}`) specifies a location one pixel to the right and two pixels down from the current location.

### Returns

Nothing.

### Description

Method; moves selected objects by a specified distance.

**Note:** When the user uses the arrow keys to move the item, the History panel combines all presses of the arrow key as one move step. When the user presses the arrow keys repeatedly, rather than taking multiple steps in the History panel, the method performs one step, and the arguments are updated to reflect the repeated arrow keys.

For information on making a selection, see [document.setSelectionRect\(\)](#), [document.mouseClick\(\)](#), [document.mouseDblClk\(\)](#), and the [Element object](#).

### Example

The following example moves the selected item 62 pixels to the right and 84 pixels down:

```
f1.getDocumentDOM().moveSelectionBy({x:62, y:84});
```

## document.name

### Availability

Flash MX 2004.

### Usage

```
document.name
```

### Description

Read-only property; a string that represents the name of a document (FLA file).

### Example

The following example sets the variable `fileName` to the filename of the first document in the documents array:

```
var fileName = flash.documents[0].name;
```

The following example displays the names of all the open documents in the Output panel:

```
var openDocs = fl.documents;
for(var i=0;i < openDocs.length; i++){
    fl.trace(i + " " + openDocs[i].name +"\n");
}
```

## document.optimizeCurves()

### Availability

Flash MX 2004.

### Usage

```
document.optimizeCurves(smoothing, bUseMultiplePasses)
```

### Parameters

**smoothing** An integer in the range from 0 to 100, with 0 specifying no smoothing and 100 specifying maximum smoothing.

**bUseMultiplePasses** A Boolean value that, when set to `true`, indicates that the method should use multiple passes, which is slower but produces a better result. This parameter has the same effect as clicking the Use Multiple Passes button in the Optimize Curves dialog box.

### Returns

Nothing.

### Description

Method; optimizes smoothing for the current selection, allowing multiple passes, if specified, for optimal smoothing. This method is equivalent to selecting Modify > Shape > Optimize.

### Example

The following example optimizes the curve of the current selection to 50° of smoothing with multiple passes:

```
fl.getDocumentDOM().optimizeCurves(50, true);
```

## document.path

### Availability

Flash MX 2004.

### Usage

```
document.path
```

### Description

Read-only property; a string that represents the path of the document in a platform-specific format. If the document has never been saved, this property is `undefined`.

**Example**

The following example displays the path of the first document in the documents array in the Output panel. You must save the document before running this script. In the example, the file is named test.fla and is saved in the My Documents folder on a Windows computer.

```
var filePath = flash.documents[0].path;
fl.trace(filePath);
// displays C:\Documents and Settings\<user name>\My Documents\test.fla
```

**See also**

[document.pathURI](#)

## document.pathURI

**Availability**

Flash CS4 Professional.

**Usage**

`document.pathURI`

**Description**

Read-only property; a string that represents the path of the document, expressed as a file:/// URI. If the document has never been saved, this property is `undefined`.

**Example**

The following example displays the path of the first document in the documents array as a file:/// URI string in the Output panel. You must save the document before running this script. In the example, the file is named test.fla and is saved in the My Documents folder on a Windows computer.

```
var filePathURI = flash.documents[0].pathURI;
fl.trace(filePathURI);
// displays file:///C:/Documents%20and%20Settings/<userName>/My%20Documents/test.fla
```

**See also**

[document.path](#)

## document.publish()

**Availability**

Flash MX 2004.

**Usage**

`document.publish()`

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; publishes the document according to the active publish settings (File > Publish Settings). This method is equivalent to selecting File > Publish.

**Example**

The following example publishes the current document:

```
f1.getDocumentDOM().publish();
```

## document.publishProfiles

**Availability**

Flash MX 2004.

**Usage**

```
document.publishProfiles
```

**Description**

Read-only property; an array of the publish profile names for the document.

**Example**

The following example displays the names of the publish profiles for the document:

```
var myPubProfiles = f1.getDocumentDOM().publishProfiles;
for (var i=0; i < myPubProfiles.length; i++) {
    f1.trace(myPubProfiles[i]);
}
```

## document.punch()

**Availability**

Flash 8.

**Usage**

```
document.punch()
```

**Parameters**

None.

**Returns**

None.

**Description**

Method; uses the top selected drawing object to punch through all selected drawing objects underneath it. If no objects are selected, calling this method results in an error and the script breaks at that point.

**Example**

The following example punches through drawing objects underneath the selected drawing object:

```
f1.getDocumentDOM().punch();
```

**See also**

[document.crop\(\)](#), [document.deleteEnvelope\(\)](#), [document.intersect\(\)](#), [document.union\(\)](#),  
[shape.isDrawingObject](#)

## document.removeAllFilters()

**Availability**

Flash 8.

**Usage**

```
document.removeAllFilters()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; removes all filters from the selected objects.

**Example**

The following example removes all filters from the selected objects:

```
f1.getDocumentDOM().removeAllFilters();
```

**See also**

[document.addFilter\(\)](#), [document.changeFilterOrder\(\)](#), [document.disableAllFilters\(\)](#),  
[document.getFilters\(\)](#), [document.removeFilter\(\)](#), [Filter object](#)

## document.removeDataFromDocument()

**Availability**

Flash MX 2004.

**Usage**

```
document.removeDataFromDocument(name)
```

**Parameters**

**name** A string that specifies the name of the data to remove.

**Returns**

Nothing.

**Description**

Method; removes persistent data with the specified name that has been attached to the document.

**Example**

The following example removes from the document the persistent data named "myData":

```
f1.getDocumentDOM().removeDataFromDocument("myData");
```

**See also**

[document.addDataToDocument\(\)](#), [document.documentHasData\(\)](#), [document.getDataFromDocument\(\)](#)

## document.removeDataFromSelection()

**Availability**

Flash MX 2004.

**Usage**

```
document.removeDataFromSelection(name)
```

**Parameters**

**name** A string that specifies the name of the persistent data to remove.

**Returns**

Nothing.

**Description**

Method; removes persistent data with the specified name that has been attached to the selection.

**Example**

The following example removes from the selection the persistent data named "myData":

```
f1.getDocumentDOM().removeDataFromSelection("myData");
```

**See also**

[document.addDataToSelection\(\)](#)

## document.removeFilter()

### Availability

Flash 8.

### Usage

```
document.removeFilter(filterIndex)
```

### Parameters

**filterIndex** An integer specifying the zero-based index of the filter to remove from the selected objects.

### Returns

Nothing.

### Description

Method; removes the specified filter from the Filters list of the selected objects.

### Example

The following example removes the first filter (index value 0) from the Filters list of the selected objects:

```
f1.getDocumentDOM().removeFilter(0);
```

### See also

[document.addFilter\(\)](#), [document.changeFilterOrder\(\)](#), [document.disableFilter\(\)](#),  
[document.getFilters\(\)](#), [document.removeAllFilters\(\)](#), [Filter object](#)

## document.renamePublishProfile()

### Availability

Flash MX 2004.

### Usage

```
document.renamePublishProfile([profileNewName])
```

### Parameters

**profileNewName** An optional parameter that specifies the new name for the profile. The new name must be unique. If the name is not specified, a default name is provided.

### Returns

A Boolean value: `true` if the name is changed successfully; `false` otherwise.

### Description

Method; renames the current profile.

### Example

The following example renames the current profile to a default name and displays it:

```
alert(f1.getDocumentDOM().renamePublishProfile());
```

## document.renameScene()

### Availability

Flash MX 2004.

### Usage

```
document.renameScene(name)
```

### Parameters

**name** A string that specifies the new name of the scene.

### Returns

A Boolean value: `true` if the name is changed successfully; `false` otherwise. If the new name is not unique, for example, the method returns `false`.

### Description

Method; renames the currently selected scene in the Scenes panel. The new name for the selected scene must be unique.

### Example

The following example renames the current scene to "new\_name":

```
var success = f1.getDocumentDOM().renameScene("new_name");
```

## document.reorderScene()

### Availability

Flash MX 2004.

### Usage

```
document.reorderScene(sceneToMove, sceneToPutItBefore)
```

### Parameters

**sceneToMove** An integer that specifies which scene to move, with 0 (zero) being the first scene.

**sceneToPutItBefore** An integer that specifies the scene before which you want to move the scene specified by `sceneToMove`. Specify 0 (zero) for the first scene. For example, if you specify 1 for `sceneToMove` and 0 for `sceneToPutItBefore`, the second scene is placed before the first scene. Specify -1 to move the scene to the end.

### Returns

Nothing.

### Description

Method; moves the specified scene before another specified scene.

**Example**

The following example moves the second scene to before the first scene:

```
f1.getDocumentDOM().reorderScene(1, 0);
```

## document.resetOvalObject()

**Availability**

Flash CS3 Professional.

**Usage**

```
document.resetOvalObject()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; sets all values in the Property inspector to default Oval object settings. If any Oval objects are selected, their properties are reset to default values as well.

**Example**

The following example resets Oval object properties in the current document to default values:

```
f1.getDocumentDOM().resetOvalObject();
```

**See also**

[document.resetRectangleObject\(\)](#)

## document.resetRectangleObject()

**Availability**

Flash CS3 Professional.

**Usage**

```
document.resetRectangleObject()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; sets all values in the Property inspector to default Rectangle object settings. If any Rectangle objects are selected, their properties are reset to default values as well.

**Example**

The following example resets Rectangle object properties in the current document to default values:

```
f1.getDocumentDOM().resetRectangleObject();
```

**See also**

[document.resetOvalObject\(\)](#)

## document.resetTransformation()

**Availability**

Flash MX 2004.

**Usage**

```
document.resetTransformation()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; resets the transformation matrix. This method is equivalent to selecting Modify > Transform > Remove Transform.

**Example**

The following example resets the transformation matrix for the current selection:

```
f1.getDocumentDOM().resetTransformation();
```

## document.revert()

**Availability**

Flash MX 2004.

**Usage**

```
document.revert()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; reverts the specified document to its previously saved version. This method is equivalent to selecting File > Revert.

**Example**

The following example reverts the current document to the previously saved version:

```
f1.getDocumentDOM().revert();
```

**See also**

[document.canRevert\(\)](#), [f1.revertDocument\(\)](#)

## document.rotate3DSelection()

**Availability**

Flash CS4 Professional.

**Usage**

```
document.rotate3DSelection(xyzCoordinate, bGlobalTransform)
```

**Parameters**

**xyzCoordinate** An XYZ coordinate point that specifies the axes for 3D rotation.

**bGlobalTransform** A Boolean value that specifies whether the transformation mode should be global (`true`) or local (`false`).

**Returns**

Nothing.

**Description**

Method: applies a 3D rotation to the selection. This method is available only for movie clips.

**Example**

In the following example, the selection is first rotated relative to the stage (globally) and then relative to itself (locally).

```
var myDocument = f1.getDocumentDOM();
myDocument.rotate3DSelection({x:52.0, y:0, z:0}, true);
myDocument.rotate3DSelection({x:52.0, y:0, z:-55.2}, false);
```

## document.rotateSelection()

**Availability**

Flash MX 2004.

**Usage**

```
document.rotateSelection(angle [, rotationPoint])
```

**Parameters**

**angle** A floating-point value that specifies the angle of the rotation.

**rotationPoint** A string that specifies which side of the bounding box to rotate. Acceptable values are "top right", "top left", "bottom right", "bottom left", "top center", "right center", "bottom center", and "left center". If unspecified, the method uses the transformation point. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; rotates the selection by a specified number of degrees. The effect is the same as using the Free Transform tool to rotate the object.

**Example**

The following example rotates the selection by 45° around the transformation point:

```
f1.getDocumentDOM().rotateSelection(45);
```

The following example rotates the selection by 45° around the lower-left corner:

```
f1.getDocumentDOM().rotateSelection(45, "bottom left");
```

## document.save()

**Availability**

Flash MX 2004.

**Usage**

```
document.save([bOkToSaveAs])
```

**Parameters**

**bOkToSaveAs** An optional parameter that, if `true` or omitted, and the file was never saved, opens the Save As dialog box. If `false` and the file was never saved, the file is not saved.

**Returns**

A Boolean value: `true` if the save operation completes successfully; `false` otherwise.

**Description**

Method; saves the document in its default location. This method is equivalent to selecting File > Save.

To specify a name for the file (instead of saving it with the same name), use [f1.saveDocument\(\)](#).

**Note:** If the file is new and has not been modified or saved, or if the file has not been modified since the last time it was saved, this method has no effect and `false` is returned. To allow an unsaved or unmodified file to be saved, use [f1.saveDocumentAs\(\)](#).

**Example**

The following example saves the current document in its default location:

```
f1.getDocumentDOM().save();
```

**See also**

[f1.saveAll\(\)](#), [f1.saveDocument\(\)](#), [f1.saveDocumentAs\(\)](#)

## document.saveAndCompact() - dropped

**Availability**

Flash MX 2004. *Dropped in Flash Professional CS6.*

**Usage**

```
document.saveAndCompact([bOkToSaveAs])
```

**Parameters**

**bOkToSaveAs** An optional parameter that, if `true` or omitted and the file was never saved, opens the Save As dialog box. If `false` and the file was never saved, the file is not saved. The default value is `true`.

**Returns**

A Boolean value: `true` if the save-and-compact operation completes successfully; `false` otherwise.

**Description**

*Dropped in Flash Professional CS6.*

**Example**

The following example saves and compacts the current document:

```
f1.getDocumentDOM().saveAndCompact();
```

**See also**

[document.save\(\)](#), [f1.saveDocumentAs\(\)](#), [f1.saveDocument\(\)](#), [f1.saveAll\(\)](#)

## document.saveAsCopy()

**Availability**

Flash CS6.

**Usage**

```
document.saveAsCopy(URI [, selectionOnly])
```

**Parameters**

**URI** String: The URI to export the new FLA file to. This URI must reference a local file. Example:  
file:///c|/tests/myTest.fla.

**selectionOnly** Optional. A boolean indicating whether only the current Stage selection should be saved to the new FLA file.

**Returns**

Boolean.

**Description**

Method; Saves a new FLA file based on the existing document object, with an option to save only the current selection on Stage.

**Example**

The following example exports the current selection to a new FLA file at the URI specified by the user:

```
var uri = fl.browseForFileURL("save");
var selectionOnly = true;
fl.getDocumentDOM().saveAsCopy(uri, selectionOnly);
```

## document.scaleSelection()

**Availability**

Flash MX 2004.

**Usage**

```
document.scaleSelection(xScale, yScale [, whichCorner])
```

**Parameters**

**xScale** A floating-point value that specifies the amount of *x* by which to scale.

**yScale** A floating-point value that specifies the amount of *y* by which to scale.

**whichCorner** A string value that specifies the edge about which the transformation occurs. If omitted, scaling occurs about the transformation point. Acceptable values are: "bottom left", "bottom right", "top right", "top left", "top center", "right center", "bottom center", and "left center". This parameter is optional.

**Returns**

Nothing.

**Description**

Method; scales the selection by a specified amount. This method is equivalent to using the Free Transform tool to scale the object.

**Example**

The following example expands the width of the current selection to double the original width and shrinks the height to half:

```
fl.getDocumentDOM().scaleSelection(2.0, 0.5);
```

The following example flips the selection vertically:

```
fl.getDocumentDOM().scaleSelection(1, -1);
```

The following example flips the selection horizontally:

```
fl.getDocumentDOM().scaleSelection(-1, 1);
```

The following example scales the selection vertically by 1.9 from the top center:

```
fl.getDocumentDOM().scaleSelection(1, 1.90, 'top center');
```

## document.screenOutline - dropped

### Availability

Flash MX 2004. *Dropped in Flash Professional CC.*

### Usage

```
document.screenOutline
```

### Description

*Dropped in Flash Professional CC.*

### Example

The following example displays the array of values in the screenOutline property:

```
var myArray = new Array();
for(var i in fl.getDocumentDOM().screenOutline) {
    myArray.push(" "+i+" : "+fl.getDocumentDOM().screenOutline[i]) ;
}
fl.trace("Here is the property dump for screenOutline: "+myArray);
```

### See also

[document.allowScreens\(\) - dropped](#)

## document.selectAll()

### Availability

Flash MX 2004.

### Usage

```
document.selectAll()
```

### Parameters

None.

### Returns

Nothing.

**Description**

Method; selects all items on the Stage. This method is equivalent to pressing Control+A (Windows) or Command+A (Macintosh) or selecting Edit > Select All.

**Example**

The following example selects everything that is currently visible to the user:

```
f1.getDocumentDOM().selectAll();
```

**See also**

[document.selection](#), [document.selectNone\(\)](#)

## document.selection

**Availability**

Flash MX 2004.

**Usage**

```
document.selection
```

**Description**

Property; an array of the selected objects in the document. If nothing is selected, returns an array of length zero. If no document is open, returns `null`.

To add objects to the array, you must first select them in one of the following ways:

- Manually select objects on the Stage.
- Use one of the selection methods, such as `document.setSelectionRect()`, `document.setSelectionBounds()`, `document.mouseClick()`, `document.mouseDblClk()`, or `document.selectAll()`.
- Manually select a frame or frames.
- Use one of the methods of the [Timeline object](#) to select a frame or frames, such as `timeline.getSelectedFrames()`, `timeline.setSelectedFrames()`, or `timeline.selectAllFrames()`.
- Specify all the elements in a particular frame (see [Element object](#)). See the first example below.
- Create an array of one or more elements and then assign that array to the `document.selection` array. See the third example below.

**Example**

The following example assigns all elements on Frame 11 to the current selection (remember that index values are different from frame number values):

```
f1.getDocumentDOM().getTimeline().currentFrame = 10;
f1.getDocumentDOM().selection =
f1.getDocumentDOM().getTimeline().layers[0].frames[10].elements;
```

The following example creates a rectangle in the upper left corner of the Stage and a text string underneath the rectangle. Then it selects both objects using `document.setSelectionRect()` and adds them to the `document.selection` array. Finally, it displays the contents of `document.selection` in the Output panel.

```
fl.getDocumentDOM().addNewRectangle({left:0, top:0, right:99, bottom:99}, 0);
fl.getDocumentDOM().addNewText({left:-1, top:117.3, right:9.2, bottom:134.6});
fl.getDocumentDOM().setTextString('Hello World');
fl.getDocumentDOM().setSelectionRect({left:-28, top:-22, right:156.0, bottom:163});

var theSelectionArray = fl.getDocumentDOM().selection;

for(var i=0;i<theSelectionArray.length;i++){
fl.trace("fl.getDocumentDOM().selection["+i+"] = " + theSelectionArray[i]);
}
```

The following example is an advanced example. It shows how to loop through the layer array and elements array to locate instances of a particular symbol and select them. You could extend this example to include loops for multiple frames or scenes. This example assigns all instances of the movie clip `myMovieClip` in the first frame to the current selection:

```
// Assigns the layers array to the variable "theLayers".
var theLayers = fl.getDocumentDOM().getTimeline().layers;
// Creates an array to hold all the elements
// that are instances of "myMovieClip".
var myArray = new Array();
// Counter variable
var x = 0;
// Begin loop through all the layers.
for (var i = 0; i < theLayers.length; i++) {
    // Gets the array of elements in Frame 1
    // and assigns it to the array "theElems".
    var theElems = theLayers[i].frames[0].elements;
    // Begin loop through the elements on a layer.
    for (var c = 0; c < theElems.length; c++) {
        // Checks to see if the element is of type "instance".
        if (theElems[c].elementType == "instance") {
            // If the element is an instance, it checks
            // if it is an instance of "myMovieClip".
            if (theElems[c].libraryItem.name == "myMovieClip") {
                // Assigns elements that are instances of "myMovieClip" to "myArray".
                myArray[x] = theElems[c];
                // Increments counter variable.
                x++;
            }
        }
    }
}
// Now that you have assigned all the instances of "myMovieClip"
// to "myArray", you then set the document.selection array
// equal to myArray. This selects the objects on the Stage.
fl.getDocumentDOM().selection = myArray;
```

## document.selectNone()

### Availability

Flash MX 2004.

**Usage**

```
document.selectNone()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; deselects any selected items.

**Example**

The following example deselects any items that are selected:

```
f1.getDocumentDOM().selectNone();
```

**See also**

[document.selectAll\(\)](#), [document.selection](#)

## document.setAlignToDocument()

**Availability**

Flash MX 2004.

**Usage**

```
document.setAlignToDocument(bToStage)
```

**Parameters**

**bToStage** A Boolean value that, if set to `true`, aligns objects to the Stage. If set to `false`, it does not.

**Returns**

Nothing.

**Description**

Method; sets the preferences for [document.align\(\)](#), [document.distribute\(\)](#), [document.match\(\)](#), and [document.space\(\)](#) to act on the document. This method is equivalent to enabling the To Stage button in the Align panel.

**Example**

The following example enables the To Stage button in the Align panel to align objects with the Stage:

```
f1.getDocumentDOM().setAlignToDocument(true);
```

**See also**

[document.getAlignToDocument\(\)](#)

## document.setBlendMode()

### Availability

Flash 8.

### Usage

```
document.setBlendMode(mode)
```

### Parameters

**mode** A string that represents the desired blending mode for the selected objects. Acceptable values are "normal", "layer", "multiply", "screen", "overlay", "hardlight", "lighten", "darker", "difference", "add", "subtract", "invert", "alpha", and "erase".

### Returns

Nothing.

### Description

Method; sets the blending mode for the selected objects.

### Example

The following example sets the blending mode for the selected object to "add".

```
f1.getDocumentDOM().setBlendMode("add");
```

### See also

[document.addFilter\(\)](#), [document.setFilterProperty\(\)](#), [symbolInstance.blendMode](#)

## document.setCustomFill()

### Availability

Flash MX 2004.

### Usage

```
document.setCustomFill(fill)
```

### Parameters

**fill** A Fill object that specifies the fill settings to be used. See [Fill object](#).

### Returns

Nothing.

### Description

Method; sets the fill settings for the Tools panel, Property inspector, and any selected shapes. This allows a script to set the fill settings before drawing the object, rather than drawing the object, selecting it, and changing the fill settings. It also lets a script change the Tools panel and Property inspector fill settings.

**Example**

The following example changes the color of the fill color swatch in the Tools panel, Property inspector, and any selected shapes to white:

```
var fill = fl.getDocumentDOM().getCustomFill();
fill.color = '#FFFFFF';
fill.style = "solid";
fl.getDocumentDOM().setCustomFill(fill);
```

**See also**

[document.getCustomFill\(\)](#)

## document.setCustomStroke()

**Availability**

Flash MX 2004.

**Usage**

```
document.setCustomStroke(stroke)
```

**Parameters**

**stroke** A [Stroke object](#).

**Returns**

Nothing.

**Description**

Method; sets the stroke settings for the Tools panel, Property inspector, and any selected shapes. This allows a script to set the stroke settings before drawing the object, rather than drawing the object, selecting it, and changing the stroke settings. It also lets a script change the Tools panel and Property inspector stroke settings.

**Example**

The following example changes the stroke thickness setting in the Tools panel, Property inspector, and any selected shapes:

```
var stroke = fl.getDocumentDOM().getCustomStroke();
stroke.thickness += 2;
fl.getDocumentDOM().setCustomStroke(stroke);
```

**See also**

[document.getCustomStroke\(\)](#)

## document.setProperty()

**Availability**

Flash MX 2004.

**Usage**

```
document.setProperty(property, value)
```

**Parameters**

**property** A string that specifies the name of the `Element` property to set. For a complete list of properties and values, see the Property summary table for the [Element object](#).

You can't use this method to set values for read-only properties, such as `element.elementType`, `element.top`, or `element.left`.

**value** An integer that specifies the value to set in the specified `Element` property.

**Returns**

Nothing.

**Description**

Method; sets the specified `Element` property on selected objects in the document. This method does nothing if there is no selection.

**Example**

The following example sets the width of all selected objects to 100 and the height to 50:

```
f1.getDocumentDOM().setProperty("width", 100);
f1.getDocumentDOM().setProperty("height", 50);
```

## document.setElementTextAttr()

**Availability**

Flash MX 2004.

**Usage**

```
document.setElementTextAttr(attrName, attrValue [, startIndex [, endIndex]])
```

**Parameters**

**attrName** A string that specifies the name of the `TextAttrs` property to change.

**attrValue** The value to which to set the `TextAttrs` property. For a list of property names and expected values, see the Property summary table for the [TextAttrs object](#).

**startIndex** An integer value that specifies the index of the first character that is affected. This parameter is optional.

**endIndex** An integer value that specifies the index of the last character that is affected. This parameter is optional.

**Returns**

A Boolean value: `true` if at least one text attribute property is changed; `false` otherwise.

### Description

Method; sets the specified `textAttrs` property of the selected text items to the specified value. For a list of property names and allowable values, see the Property summary table for the [TextAttrs object](#). If the optional parameters are not passed, the method sets the style of the currently selected text range, or the whole text field if no text is selected. If only `startIndex` is passed, the method sets that character's attributes. If `startIndex` and `endIndex` are passed, the method sets the attributes on the characters starting from `startIndex` up to, but not including, `endIndex`. If paragraph styles are specified, all the paragraphs that fall within the range are affected.

### Example

The following examples set the `fillColor`, `italic`, and `bold` text attributes for the selected text items:

```
var success = fl.getDocumentDOM().setElementTextAttr("fillColor", "#00ff00");
var pass = fl.getDocumentDOM().setElementTextAttr("italic", true, 10);
var ok = fl.getDocumentDOM().setElementTextAttr("bold", true, 5, 15);
```

## document.setFillColor()

### Availability

Flash MX 2004.

### Usage

```
document.setFillColor(color)
```

### Parameters

`color` The color of the fill, in one of the following formats:

- A string in the format "#RRGGBB" or "#RRGGBAA"
- A hexadecimal number in the format 0xRRGGBB
- An integer that represents the decimal equivalent of a hexadecimal number

If set to `null`, no fill color is set, which is the same as setting the Fill color swatch in the user interface to no fill.

### Returns

Nothing.

### Description

Method; changes the selection and the tools panel to the specified fill color. For additional information on changing the fill color in the Tools panel and Property inspector, see [document.setCustomFill\(\)](#).

### Example

The first three statements in the following example set the fill color using each of the different formats for specifying color. The fourth statement sets the fill to no fill.

```
fl.getDocumentDOM().setFillColor("#cc00cc");
fl.getDocumentDOM().setFillColor(0xcc00cc);
fl.getDocumentDOM().setFillColor(120000);
fl.getDocumentDOM().setFillColor(null);
```

## document.setFilterProperty()

### Availability

Flash 8.

### Usage

```
document.setFilterProperty(property, filterIndex, value)
```

### Parameters

**property** A string specifying the property to be set. Acceptable values are "blurX", "blurY", "quality", "angle", "distance", "strength", "knockout", "inner", "bevelType", "color", "shadowColor", and "highlightColor".

**filterIndex** An integer specifying the zero-based index of the filter in the Filters list.

**value** A number or string specifying the value to be set for the specified filter property. Acceptable values depend on the property and the filter being set.

### Returns

Nothing.

### Description

Method; sets a specified filter property for the currently selected objects (assuming that the object supports the specified filter).

### Example

The following example sets the `quality` property to 2 for the second filter (index value of 1) in the Filters list of the selected objects and then sets the `shadowColor` property of the first filter in the Filters list on the selected objects:

```
f1.getDocumentDOM().setFilterProperty("quality", 1, 2);
f1.getDocumentDOM().setFilterProperty("shadowColor", 0, "#FF00FF");
```

### See also

[document.addFilter\(\)](#), [document.getFilters\(\)](#), [document.setBlendMode\(\)](#), [document.setFilters\(\)](#), [Filter object](#)

## document.setFilters()

### Availability

Flash 8.

### Usage

```
document.setFilters(filterArray)
```

### Parameters

**filterArray** The array of filters currently specified.

**Returns**

Nothing.

**Description**

Method; applies filters to the selected objects. Use this method after calling `document.getFilters()` and making any desired changes to the filters.

**Example**

The following example gets the filters on the selected object and sets the `blurX` property for all Blur filters to 50:

```
var myFilters = fl.getDocumentDOM().getFilters();
for (i=0; i < myFilters.length; i++) {
    if (myFilters[i].name == "blurFilter"){
        myFilters[i].blurX = 50;
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

**See also**

[document.addFilter\(\)](#), [document.getFilters\(\)](#), [document.setFilterProperty\(\)](#), [Filter object](#)

## document.setInstanceAlpha()

**Availability**

Flash MX 2004.

**Usage**

```
document.setInstanceAlpha(opacity)
```

**Parameters**

**opacity** An integer between 0 (transparent) and 100 (completely saturated) that adjusts the transparency of the instance.

**Returns**

Nothing.

**Description**

Methods; sets the opacity of the instance.

**Example**

The following example sets the opacity of the tint to a value of 50:

```
fl.getDocumentDOM().setInstanceAlpha(50);
```

## document.setInstanceBrightness()

### Availability

Flash MX 2004.

### Usage

```
document.setInstanceBrightness(brightness)
```

### Parameters

**brightness** An integer that specifies brightness as a value from -100 (black) to 100 (white).

### Returns

Nothing.

### Description

Method; sets the brightness for the instance.

### Example

The following example sets the brightness for the instance to a value of 50:

```
f1.getDocumentDOM().setInstanceBrightness(50);
```

## document.setInstanceTint()

### Availability

Flash MX 2004.

### Usage

```
document.setInstanceTint( color, strength )
```

### Parameters

**color** The color of the tint, in one of the following formats:

- A string in the format "#RRGGBB" or "#RRGGBAA"
- A hexadecimal number in the format 0xRRGGBB
- An integer that represents the decimal equivalent of a hexadecimal number

**strength** An integer between 0 and 100 that specifies the opacity of the tint.

### Returns

Nothing.

### Description

Method; sets the tint for the instance.

**Example**

The following example sets the tint for the selected instance to red with an opacity value of 50:

```
f1.getDocumentDOM().setInstanceTint(0xff0000, 50);
```

## document.setMetadata()

**Availability**

Flash 8.

**Usage**

```
document.setMetadata(strMetadata)
```

**Parameters**

**strMetadata** A string containing the XML metadata to be associated with the document. For more information, see the following description.

**Returns**

A Boolean value: `true` if successful; `false` otherwise.

**Description**

Method; sets the XML metadata for the specified document, overwriting any existing metadata. The XML passed as `strMetadata` is validated and may be rewritten before being stored. If it cannot be validated as legal XML or violates specific rules, then the XML metadata is not set and `false` is returned. (If `false` is returned, there is no way to get more detailed error information.)

**Note:** Even if `true` is returned, the XML that is set may not be exactly the same string that you passed in. To get the exact value to which the XML was set, use [document.getMetadata\(\)](#).

The format of the metadata is RDF that is compliant with the XMP specification. For more information about RDF and XMP, see the following sources:

- The RDF Primer at [www.w3.org/TR/rdf-primer/](http://www.w3.org/TR/rdf-primer/)
- The RDF specification at [www.w3.org/TR/1999/REC-rdf-syntax-19990222/](http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/)
- The XMP home page at [www.adobe.com/products/xmp/](http://www.adobe.com/products/xmp/)

**Example**

The following examples show several different legal ways to represent the same data. In all of these cases but the second one, if the data were sent to `Document.setMetadata()`, it would not be rewritten (aside from removing line breaks).

In the first example, metadata is in tags, with different schemas placed in separate `rdf:Description` tags:

```
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<rdf:Description rdf:about='' xmlns:dc='http://purl.org/dc/1.1/'>
<dc:title>Simple title</dc:title>
<dc:description>Simple description</dc:description>
</rdf:Description>
<rdf:Description rdf:about='' xmlns:xmp='http://ns.adobe.com/xap/1.0/'>
<xmp:CreateDate>2004-10-12T10:29-07:00</xmp:CreateDate>
<xmp:CreatorTool>Flash Authoring WIN 8,0,0,215</xmp:CreatorTool>
</rdf:Description>
</rdf:RDF>
```

In the second example, metadata is in tags, but with different schemas all in one `rdf:Description` tag. This example also includes comments, which will be ignored and discarded by the `Document.setMetadata()`:

```
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
    <!-- This is before the first rdf:Description tag -->
<rdf:Description rdf:about='' xmlns:dc='http://purl.org/dc/1.1/'>
<dc:title>Simple title</dc:title>
<dc:description>Simple description</dc:description>
</rdf:Description>
    <!-- This is between the two rdf:Description tags -->
<rdf:Description rdf:about='' xmlns:xmp='http://ns.adobe.com/xap/1.0/'>
<xmp:CreateDate>2004-10-12T10:29-07:00</xmp:CreateDate>
<xmp:CreatorTool>Flash Authoring WIN 8,0,0,215</xmp:CreatorTool>
</rdf:Description>
    <!-- This is after the second rdf:Description tag -->
</rdf:RDF>
```

In the third example, metadata is in attributes, and different schemas are all in one `rdf:Description` tag:

```
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<rdf:Description rdf:about='' xmlns:dc='http://purl.org/dc/1.1/' dc:title='Simple title'
dc:description='Simple description' />
<rdf:Description rdf:about='' xmlns:xmp='http://ns.adobe.com/xap/1.0/'
xmp:CreateDate='2004-10-12T10:29-07:00' xmp:CreatorTool='Flash Authoring WIN 8,0,0,215' />
</rdf:RDF>
```

## See also

[document.getMetadata\(\)](#)

# document.setMobileSettings()

## Availability

Flash CS3 Professional.

## Usage

```
document.setMobileSettings(xmlString)
```

## Parameters

**xmlString** A string that describes the XML settings in a mobile FLA file.

## Returns

A value of `true` if the settings were successfully set; `false` otherwise.

**Description**

Method; sets the value of an XML settings string in a mobile FLA file. (Most mobile FLA files have an XML string that describes the settings within the document.)

**Example**

The following example sets the XML settings string for a mobile FLA file. Note that the example below represents a single line of code.

```
fl.getDocumentDOM().setMobileSettings("<? xml version='1.0' encoding='UTF-16' standalone='no'?> <mobileSettings> <contentType id='standalonePlayer' name='Standalone Player'/> <testDevices> <testDevice id='1170' name='Generic Phone' selected='yes'/> </testDevices> <outputMsgFiltering info='no' trace='yes' warning='yes'/> <testWindowState height='496' splitterClosed='No' splitterXPos='400' width='907'/> </mobileSettings>");
```

**See also**

[document.getMobileSettings\(\)](#)

## document.setOvalObjectProperty()

**Availability**

Flash CS3 Professional.

**Usage**

```
document.setOvalObjectProperty(propertyName, value)
```

**Parameters**

**propertyName** A string that specifies the property to be set. For acceptable values, see the Property summary table for the [Oval object](#).

**value** The value to be assigned to the property. Acceptable values vary depending on the property you specify in *propertyName*.

**Returns**

Nothing.

**Description**

Method; specifies a value for a specified property of primitive Oval objects.

**Example**

See individual properties in [Oval object](#) for examples.

**See also**

[Oval object](#), [shape.isOvalObject](#)

## document.setPlayerVersion()

### Availability

Flash CS3 Professional.

### Usage

```
document.setPlayerVersion(version)
```

### Parameters

**version** A string that represents the version of Flash Player targeted by the specified document. Acceptable values are "FlashLite", "FlashLite11", "FlashLite20", "FlashLite30", "1", "2", "3", "4", "5", "6", "7", "8", "9", "FlashPlayer10", "FlashPlayer10.3", "FlashPlayer11.1", "FlashPlayer11.2", "FlashPlayer11.3", "FlashPlayer11.4", "FlashPlayer11.5", "FlashPlayer11.6", "FlashPlayer11.7", "AdobeAIR1\_1", "AdobeAIR1\_1", "AdobeAIR2\_5", "AdobeAIR3\_6", "android3\_6", and "PF13\_6".

### Returns

A value of `true` if the player version was successfully set; `false` otherwise.

### Description

Method; sets the version of the Flash Player targeted by the specified document. This is the same value as that set in the Publish Settings dialog box.

### Example

The following example targets Flash Player 6 as the player version for the current document:

```
f1.getDocumentDOM().setPlayerVersion("6");
```

### See also

[document.getPlayerVersion\(\)](#)

## document.setPublishDocumentData()

### Availability

Flash Professional CC.

### Usage

```
document.setPublishDocumentData(format, publish)
```

### Parameters

**format** A string that specifies the publishing format.

**Note:** `_EMBED_SWF` is a special built-in publishing format for persistent data. If set, the persistent data will be embedded in the SWF file every time a document is published. The persistent data can then be accessed via ActionScript with the `.metaData` property. This requires SWF version 19 (Flash Player 11.6) and above and is only for symbol instances onstage. Other custom publishing formats may be specified for custom JSFL scripts if this method is called with the same format.

**publish** A boolean that indicates whether to enable or disable publishing of persistent data for the specified format.

None.

### Returns

None.

### Description

Method; Enables or disables publishing of persistent data for an entire document.

### Example

The following example illustrates the use of this method:

```
var doc = fl.getDocumentDOM();
// set the data
if (doc) {
    // get the first selected element
    var elem = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];
    if (elem) {
        // add persistent data "myAlign" of "left"
        elem.setPersistentData( "myAlign", "string", "left" );
        // enable publishing of persistent data to SWF for the element
        elem.setPublishPersistentData("myAlign", "_EMBED_SWF_", true);
        // enable publishing to SWF for entire document
        doc.setPublishDocumentData("_EMBED_SWF_", true);
    }
}
// example getting data
if (doc && doc.getPublishDocumentData("_EMBED_SWF_")) {
    // get the first selected element
    var elem = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];
    if (elem && elem.hasPersistentData("myAlign") && elem.getPersistentData("myAlign",
    "_EMBED_SWF_")) {
        alert("elem.metaData.myAlign = '" + elem.getPersistentData("myAlign") + "' will be
embedded in SWF when published.");
    }
}
```

### See also

[document.getPublishDocumentData\(\)](#)

## document.setRectangleObjectProperty()

### Availability

Flash CS3 Professional.

### Usage

`document.setRectangleObjectProperty(propertyName, value)`

**Parameters**

**propertyName** A string that specifies the property to be set. For acceptable values, see the Property summary table for the [Rectangle object](#).

**value** The value to be assigned to the property. Acceptable values vary depending on the property you specify in *propertyName*.

**Returns**

Nothing.

**Description**

Method; specifies a value for a specified property of primitive Rectangle objects.

**Example**

See individual properties in [Rectangle object](#) for examples.

**See also**

[Rectangle object](#), [shape.isRectangleObject](#)

## document.setSelectionBounds()

**Availability**

Flash MX 2004; *bContactSensitiveSelection* parameter added in Flash 8.

**Usage**

```
document.setSelectionBounds(boundingRectangle [, bContactSensitiveSelection])
```

**Parameters**

**boundingRectangle** A rectangle that specifies the new location and size of the selection. For information on the format of *boundingRectangle*, see [document.addNewRectangle\(\)](#).

**bContactSensitiveSelection** A Boolean value that specifies whether the Contact Sensitive selection mode is enabled (`true`) or disabled (`false`) during object selection. The default value is `false`.

**Returns**

Nothing.

**Description**

Method; moves and resizes the selection in a single operation.

If you pass a value for *bContactSensitiveSelection*, it is valid only for this method and doesn't affect the Contact Sensitive selection mode for the document (see [f1.contactSensitiveSelection](#)).

**Example**

The following example moves the current selection to 10, 20 and resizes it to 100, 200:

```
var l = 10;  
var t = 20;  
f1.getDocumentDOM().setSelectionBounds({left:l, top:t, right:(100+l), bottom:(200+t)});
```

**See also**

[document.selection](#), [document.setSelectionRect\(\)](#)

## document.setSelectionRect()

**Availability**

Flash MX 2004; *bContactSensitiveSelection* parameter added in Flash 8.

**Usage**

```
document.setSelectionRect(rect [, bReplaceCurrentSelection [, bContactSensitiveSelection]])
```

**Parameters**

**rect** A rectangle object to set as selected. For information on the format of *rect*, see [document.addNewRectangle\(\)](#).

**bReplaceCurrentSelection** A Boolean value that specifies whether the method replaces the current selection (`true`) or adds to the current selection (`false`). The default value is `true`.

**bContactSensitiveSelection** A Boolean value that specifies whether the Contact Sensitive selection mode is enabled (`true`) or disabled (`false`) during object selection. The default value is `false`.

**Returns**

Nothing.

**Description**

Method; draws a rectangular selection marquee relative to the Stage, using the specified coordinates. This is unlike `document.getSelectionRect()`, in which the rectangle is relative to the object being edited.

This method is equivalent to dragging a rectangle with the Selection tool. An instance must be fully enclosed by the rectangle to be selected.

If you pass a value for *bContactSensitiveSelection*, it is valid only for this method and doesn't affect the Contact Sensitive selection mode for the document (see [f1.contactSensitiveSelection](#)).

**Note:** Repeating `setSelectionRect()` using the History panel or menu item repeats the step previous to the `setSelectionRect()` operation.

**Example**

In the following example, the second selection replaces the first one:

```
f1.getDocumentDOM().setSelectionRect({left:1, top:1, right:200, bottom:200});  
f1.getDocumentDOM().setSelectionRect({left:364.0, top:203.0, right:508.0, bottom:434.0},  
true);
```

In the following example, the second selection is added to the first selection. This is the same as the manual operation of holding down Shift and selecting a second object.

```
f1.getDocumentDOM().setSelectionRect({left:1, top:1, right:200, bottom:200});  
f1.getDocumentDOM().setSelectionRect({left:364.0, top:203.0, right:508.0, bottom:434.0},  
false);
```

**See also**

[document.getSelectionRect\(\)](#), [document.selection](#), [document.setSelectionBounds\(\)](#)

## document.setStageVanishingPoint()

### Availability

Flash CS4 Professional.

### Usage

```
document.setStageVanishingPoint(point)
```

### Parameters

**point** A point that specifies the *x* and *y* coordinates of the location at which to set the vanishing point for viewing 3D objects.

### Returns

Nothing.

### Description

Specifies the vanishing point for viewing 3D objects.

### Example

The following example sets the Stage vanishing point:

```
f1.getDocumentDOM().setStageVanishingPoint({x:45, y:45});
```

## document.setStageViewAngle()

### Availability

Flash CS4 Professional.

### Usage

```
document.setStageViewAngle(angle)
```

### Parameters

**angle** A floating point value between 0.0 and 179.0.

### Returns

Nothing.

### Description

Specifies the perspective angle for viewing 3D objects.

### Example

The following example sets the Stage perspective angle to 70 degrees:

```
f1.getDocumentDOM().setStageViewAngle(70);
```

## document.setStroke()

### Availability

Flash MX 2004.

### Usage

```
document.setStroke(color, size, strokeType)
```

### Parameters

**color** The color of the stroke, in one of the following formats:

- A string in the format "#RRGGBB" or "#RRGGBBAA"
- A hexadecimal number in the format 0xRRGGBB
- An integer that represents the decimal equivalent of a hexadecimal number

**size** A floating-point value that specifies the new stroke size for the selection.

**strokeType** A string that specifies the new type of stroke for the selection. Acceptable values are "hairline", "solid", "dashed", "dotted", "ragged", "stipple", and "hatched".

### Returns

Nothing.

### Description

Method; sets the color, width, and style of the selected stroke. For information on changing the stroke in the Tools panel and Property inspector, see [document.setCustomStroke\(\)](#).

### Example

The following example sets the color of the stroke to red, the size to 3.25, and the type to dashed:

```
f1.getDocumentDOM().setStroke("#ff0000", 3.25, "dashed");
```

## document.setStrokeColor()

### Availability

Flash MX 2004.

### Usage

```
document.setStrokeColor(color)
```

### Parameters

**color** The color of the stroke, in one of the following formats:

- A string in the format "#RRGGBB" or "#RRGGBBAA"
- A hexadecimal number in the format 0xRRGGBB
- An integer that represents the decimal equivalent of a hexadecimal number

**Returns**

Nothing.

**Description**

Method; changes the stroke color of the selection to the specified color. For information on changing the stroke in the Tools panel and Property inspector, see [document.setCustomStroke\(\)](#).

**Example**

The three statements in the following example set the stroke color using each of the different formats for specifying color:

```
f1.getDocumentDOM().setStrokeColor("#cc00cc");
f1.getDocumentDOM().setStrokeColor(0xcc00cc);
f1.getDocumentDOM().setStrokeColor(120000);
```

## document.setStrokeSize()

**Availability**

Flash MX 2004.

**Usage**

```
document.setStrokeSize(size)
```

**Parameters**

**size** A floating-point value from 0.25 to 250 that specifies the stroke size. The method ignores precision greater than two decimal places.

**Returns**

Nothing.

**Description**

Method; changes the stroke size of the selection to the specified size. For information on changing the stroke in the Tools panel and Property inspector, see [document.setCustomStroke\(\)](#).

**Example**

The following example changes the stroke size for the selection to 5:

```
f1.getDocumentDOM().setStrokeSize(5);
```

## document.setStrokeStyle()

**Availability**

Flash MX 2004.

**Usage**

```
document.setStrokeStyle(strokeType)
```

**Parameters**

**strokeType** A string that specifies the stroke style for the current selection. Acceptable values are "hairline", "solid", "dashed", "dotted", "ragged", "stipple", and "hatched".

**Returns**

Nothing.

**Description**

Method; changes the stroke style of the selection to the specified style. For information on changing the stroke in the Tools panel and Property inspector, see [document.setCustomStroke\(\)](#).

**Example**

The following example changes the stroke style for the selection to "dashed":

```
f1.getDocumentDOM().setStrokeStyle("dashed");
```

## document.setTextRectangle()

**Availability**

Flash MX 2004.

**Usage**

```
document.setTextRectangle(boundingRectangle)
```

**Parameters**

**boundingRectangle** A rectangle that specifies the new size within which the text item should flow. For information on the format of *boundingRectangle*, see [document.addNewRectangle\(\)](#).

**Returns**

A Boolean value: `true` if the size of at least one text field is changed; `false` otherwise.

**Description**

Method; changes the bounding rectangle for the selected text item to the specified size. This method causes the text to reflow inside the new rectangle; the text item is not scaled or transformed. The values passed in *boundingRectangle* are used as follows:

- If the text is horizontal and static, the method takes into account only the width value passed in *boundingRectangle*; the height is automatically computed to fit all the text.
- If the text is vertical (and therefore static), the method takes into account only the height value passed in *boundingRectangle*; the width is automatically computed to fit all the text.
- If the text is dynamic or input, the method takes into account both the width and height values passed in *boundingRectangle*, and the resulting rectangle might be larger than needed to fit all the text. However, if the parameters specify a rectangle size that is too small to fit all the text, the method takes into account only the width value passed in *boundingRectangle* (the height is automatically computed to fit all the text).

**Example**

The following example changes the size of the bounding text rectangle to the specified dimensions:

```
f1.getDocumentDOM().setTextRectangle({left:0, top:0, right:50, bottom:200})
```

## document.setTextSelection()

### Availability

Flash MX 2004.

### Usage

```
document.setTextSelection(startIndex, endIndex)
```

### Parameters

**startIndex** An integer that specifies the position of the first character to select. The first character position is 0 (zero).

**endIndex** An integer that specifies the end position of the selection up to, but not including, *endIndex*. The first character position is 0 (zero).

### Returns

A Boolean value: `true` if the method can successfully set the text selection; `false` otherwise.

### Description

Method; sets the text selection of the currently selected text field to the values specified by the *startIndex* and *endIndex* values. Text editing is activated, if it isn't already.

### Example

The following example selects the text from the 6th character through the 25th character:

```
f1.document.setTextSelection(5, 25);
```

## document.setTextString()

### Availability

Flash MX 2004.

### Usage

```
document.setTextString(text [, startIndex [, endIndex]])
```

### Parameters

**text** A string of the characters to insert in the text field.

**startIndex** An integer that specifies the first character to replace. The first character position is 0 (zero). This parameter is optional.

**endIndex** An integer that specifies the last character to replace. This parameter is optional.

### Returns

A Boolean value: `true` if the text of at least one text string is set; `false` otherwise.

## Description

Method; inserts a string of text. If the optional parameters are not passed, the existing text selection is replaced; if the Text object isn't currently being edited, the whole text string is replaced. If only *startIndex* is passed, the string passed is inserted at this position. If *startIndex* and *endIndex* are passed, the string passed replaces the segment of text starting from *startIndex* up to, but not including, *endIndex*.

## Example

The following example replaces the current text selection with "Hello World":

```
var success = fl.getDocumentDOM().setTextString("Hello World!");
```

The following example inserts "hello" at position 6 of the current text selection:

```
var pass = fl.getDocumentDOM().setTextString("hello", 6);
```

The following example inserts "Howdy" starting at position 2 and up to, but not including, position 7 of the current text selection:

```
var ok = fl.getDocumentDOM().setTextString("Howdy", 2, 7);
```

## See also

[document.getTextString\(\)](#)

# document.setTransformationPoint()

## Availability

Flash MX 2004.

## Usage

```
document.setTransformationPoint( transformationPoint )
```

## Parameters

**transformationPoint** A point (for example, {x:10, y:20}, where x and y are floating-point numbers) that specifies values for the transformation point of each of the following elements:

- Shapes: *transformationPoint* is set relative to the document (0,0 is the upper left corner of the Stage).
- Symbols: *transformationPoint* is set relative to the symbol's registration point (0,0 is located at the registration point).
- Text: *transformationPoint* is set relative to the text field (0,0 is the upper left corner of the text field).
- Bitmaps/videos: *transformationPoint* is set relative to the bitmap/video (0,0 is the upper left corner of the bitmap or video).
- Drawing objects, primitive ovals and rectangles, and groups: *transformationPoint* is set relative to the document (0,0 is the upper left corner of the Stage). To set *transformationPoint* relative to the center point of the object, primitive, or group, use [element.setTransformationPoint\(\)](#).

## Returns

Nothing.

**Description**

Method; sets the position of the current selection's transformation point.

**Example**

The following example sets the transformation point of the current selection to 100, 200:

```
f1.getDocumentDOM().setTransformationPoint({x:100, y:200});
```

**See also**

[document.getTransformationPoint\(\)](#), [element.setTransformationPoint\(\)](#)

## document.silent

**Availability**

Flash MX 2004.

**Usage**

```
document.silent
```

**Description**

Property; a Boolean value that specifies whether the object is accessible. This is equivalent to the inverse logic of the Make Movie Accessible setting in the Accessibility panel. That is, if `document.silent` is `true`, it is the same as the Make Movie Accessible option being unchecked. If it is `false`, it is the same as the Make Movie Accessible option being checked.

**Example**

The following example sets the `isSilent` variable to the value of the `silent` property:

```
var isSilent = f1.getDocumentDOM().silent;
```

The following example sets the `silent` property to `false`, indicating that the document is accessible:

```
f1.getDocumentDOM().silent = false;
```

## document.skewSelection()

**Availability**

Flash MX 2004.

**Usage**

```
document.skewSelection(xSkew, ySkew [, whichEdge])
```

**Parameters**

**xSkew** A floating-point number that specifies the amount of *x* by which to skew, measured in degrees.

**ySkew** A floating-point number that specifies the amount of *y* by which to skew, measured in degrees.

**whichEdge** A string that specifies the edge where the transformation occurs; if omitted, skew occurs at the transformation point. Acceptable values are "top center", "right center", "bottom center", and "left center". This parameter is optional.

**Returns**

Nothing.

**Description**

Method; skews the selection by a specified amount. The effect is the same as using the Free Transform tool to skew the object.

**Example**

The following examples skew the selected object by 2.0 vertically and 1.5 horizontally. The second example transforms the object at the top center edge:

```
f1.getDocumentDOM().skewSelection(2.0, 1.5);  
f1.getDocumentDOM().skewSelection(2.0, 1.5, "top center");
```

## document.smoothSelection()

**Availability**

Flash MX 2004.

**Usage**

```
document.smoothSelection()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; smooths the curve of each selected fill outline or curved line. This method performs the same action as the Smooth button in the Tools panel.

**Example**

The following example smooths the curve of the current selection:

```
f1.getDocumentDOM().smoothSelection();
```

## document.sourcePath

**Availability**

Flash CS4 Professional.

**Usage**

```
document.sourcePath
```

**Description**

Property; a string that contains a list of items in the document's ActionScript 3.0 Source path, which specifies the location of ActionScript class files. Items in the string are delimited by semi-colons. In the authoring tool, the items are specified by choosing File > Publish Settings and then choosing ActionScript 3.0 Script Settings on the Flash tab.

**Example**

The following example adds the ./Class files folder to the document's Source path:

```
var myDoc = fl.getDocumentDOM();
fl.trace(myDoc.sourcePath);
myDoc.sourcePath = "./Class files;" + myDoc.sourcePath;
fl.trace(myDoc.sourcePath);
```

**See also**

[document.externalLibraryPath](#), [document.libraryPath](#), [fl.sourcePath](#)

## document.space()

**Availability**

Flash MX 2004.

**Usage**

```
document.space(direction [, bUseDocumentBounds])
```

**Parameters**

**direction** A string that specifies the direction in which to space the objects in the selection. Acceptable values are "horizontal" or "vertical".

**bUseDocumentBounds** A Boolean value that, when set to `true`, spaces the objects to the document bounds. Otherwise, the method uses the bounds of the selected objects. The default is `false`. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; spaces the objects in the selection evenly.

**Example**

The following example spaces the objects horizontally, relative to the Stage:

```
fl.getDocumentDOM().space("horizontal",true);
```

The following example spaces the objects horizontally, relative to each other:

```
fl.getDocumentDOM().space("horizontal");
```

The following example spaces the objects horizontally, relative to each other, with *bUseDocumentBounds* expressly set to `false`:

```
f1.getDocumentDOM().space("horizontal",false);
```

**See also**

[document.getAlignToDocument\(\)](#), [document.setAlignToDocument\(\)](#)

## document.straightenSelection()

**Availability**

Flash MX 2004.

**Usage**

```
document.straightenSelection()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; straightens the currently selected strokes. This method is equivalent to using the Straighten button in the Tools panel.

**Example**

The following example straightens the curve of the current selection:

```
f1.getDocumentDOM().straightenSelection();
```

## document.swapElement()

**Availability**

Flash MX 2004.

**Usage**

```
document.swapElement(name)
```

**Parameters**

**name** A string that specifies the name of the library item to use.

**Returns**

Nothing.

**Description**

Method; swaps the current selection with the specified one. The selection must contain a graphic, button, movie clip, video, or bitmap. This method displays an error message if no object is selected or the given object could not be found.

**Example**

The following example swaps the current selection with `Symbol 1` from the library:

```
f1.getDocumentDOM().swapElement('Symbol 1');
```

## document.swapStrokeAndFill()

**Availability**

Flash 8.

**Usage**

```
document.swapStrokeAndFill()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; swaps the Stroke and Fill colors.

**Example**

The following example swaps the Stroke and Fill colors in the current document:

```
f1.getDocumentDOM().swapStrokeAndFill();
```

## document.swfJPEGQuality

**Availability**

Flash Professional CS6.

**Usage**

```
document.swfJPEGQuality
```

**Description**

Property; an integer, returns the JPEG Quality setting from the current Publish Profile in the document.

**Example**

The following example display the current SWF JPEG quality:

```
f1.trace("current profile's JPEG Quality is: " + f1.getDocumentDOM().swfJPEGQuality);
```

## document.testMovie()

### Availability

Flash MX 2004.

### Usage

```
document.testMovie([Boolean abortIfExistsExist])
```

### Parameters

**abortIfExistsExist** Boolean; the default value is false. If set to true, the test movie session will not start and the .swf window will not open if there are compiler errors. Compiler warnings will not abort the command. This parameter was added in Flash Professional CS5.

### Returns

Nothing.

### Description

Method; executes a Test Movie operation on the document.

### Example

The following example tests the movie for the current document, but aborts the test movie if compiler errors exist:

```
f1.getDocumentDOM().testMovie(1);
```

### See also

[document.canTestMovie\(\)](#), [document.testScene\(\)](#)

## document.testScene()

### Availability

Flash MX 2004.

### Usage

```
document.testScene()
```

### Parameters

None.

### Returns

Nothing.

### Description

Method; executes a Test Scene operation on the current scene of the document.

### Example

The following example tests the current scene in the document:

```
f1.getDocumentDOM().testScene();
```

**See also**

[document.canTestScene\(\)](#), [document.testMovie\(\)](#)

## document.timelines

**Availability**

Flash MX 2004.

**Usage**

```
document.timelines
```

**Description**

Read-only property; an array of Timeline objects (see [Timeline object](#)).

**Example**

The following example gets the array of current timelines in the active document and displays their names in the Output panel:

```
var i = 0;
var curTimelines = f1.getDocumentDOM().timelines;
while(i < f1.getDocumentDOM().timelines.length){
    alert(curTimelines[i].name);
    ++i;
}
```

**See also**

[document.currentTimeline](#), [document.getTimeline\(\)](#)

## document.traceBitmap()

**Availability**

Flash MX 2004.

**Usage**

```
document.traceBitmap(threshold, minimumArea, curveFit, cornerThreshold)
```

**Parameters**

**threshold** An integer that controls the number of colors in your traced bitmap. Acceptable values are integers between 0 and 500.

**minimumArea** An integer that specifies the radius measured in pixels. Acceptable values are integers between 1 and 1000.

**curveFit** A string that specifies how smoothly outlines are drawn. Acceptable values are "pixels", "very tight", "tight", "normal", "smooth", and "very smooth".

**cornerThreshold** A string that is similar to *curveFit*, but it pertains to the corners of the bitmap image. Acceptable values are "many corners", "normal", and "few corners".

**Returns**

Nothing.

**Description**

Method; performs a trace bitmap on the current selection. This method is equivalent to selecting Modify > Bitmap > Trace Bitmap.

**Example**

The following example traces the selected bitmap, using the specified parameters:

```
f1.getDocumentDOM().traceBitmap(0, 500, 'normal', 'normal');
```

## document.translate3DCenter()

**Availability**

Flash CS4 Professional.

**Usage**

```
document.translate3DCenter(xyzCoordinate)
```

**Parameters**

**xyzCoordinate** An XYZ coordinate that specifies the center point for 3D rotation or translation.

**Returns**

Nothing.

**Description**

Method: sets the XYZ position around which the selection is translated or rotated. This method is available only for movie clips.

**Example**

The following example specifies the XYZ axes for 3D translation:

```
f1.getDocumentDOM().translate3DCenter({x:180, y:18,z:-30});
```

## document.translate3DSelection()

**Availability**

Flash CS4 Professional.

**Usage**

```
document.translate3DSelection(xyzCoordinate, bGlobalTransform)
```

**Parameters**

**xyzCoordinate** An XYZ coordinate that specifies the axes for 3D translation.

**bGlobalTransform** A Boolean value that specifies whether the transformation mode should be global (`true`) or local (`false`).

**Returns**

Nothing.

**Description**

Method: applies a 3D translation to the selection. This method is available only for movie clips.

**Example**

In the following example, the selection is first translated relative to the stage (globally) and then relative to itself (locally).

```
var myDocument = fl.getDocumentDOM();
myDocument.translate3DSelection({x:52.0, y:0, z:0}, true);
myDocument.translate3DSelection({x:52.0, y:0, z:-55.2}, false);
```

**See also**

[document.translate3DCenter\(\)](#)

## document.transformSelection()

**Availability**

Flash MX 2004.

**Usage**

```
document.transformSelection(a, b, c, d)
```

**Parameters**

**a** A floating-point number that specifies the (0,0) element of the transformation matrix.

**b** A floating-point number that specifies the (0,1) element of the transformation matrix.

**c** A floating-point number that specifies the (1,0) element of the transformation matrix.

**d** A floating-point number that specifies the (1,1) element of the transformation matrix.

**Returns**

Nothing.

**Description**

Method; performs a general transformation on the current selection by applying the matrix specified in the arguments. For more information, see the [element.matrix](#) property.

**Example**

The following example stretches the selection by a factor of 2 in the x direction:

```
f1.getDocumentDOM().transformSelection(2.0, 0.0, 0.0, 1.0);
```

## document.unGroup()

**Availability**

Flash MX 2004.

**Usage**

```
document.unGroup()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; ungroups the current selection.

**Example**

The following example ungroups the elements in the current selection:

```
f1.getDocumentDOM().unGroup();
```

**See also**

[document.group\(\)](#)

## document.union()

**Availability**

Flash 8.

**Usage**

```
document.union()
```

**Parameters**

None.

**Returns**

None.

**Description**

Method; combines all selected shapes into a drawing object. If no objects are selected, calling this method results in an error and the script breaks at that point.

**Example**

The following example combines all selected shapes into a drawing object:

```
f1.getDocumentDOM().union();
```

**See also**

[document.crop\(\)](#), [document.deleteEnvelope\(\)](#), [document.intersect\(\)](#), [document.punch\(\)](#),  
[shape.isDrawingObject](#)

## document.unlockAllElements()

**Availability**

Flash MX 2004.

**Usage**

```
document.unlockAllElements();
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; unlocks all locked elements on the currently selected frame.

**Example**

The following example unlocks all locked objects on the current frame:

```
f1.getDocumentDOM().unlockAllElements();
```

**See also**

[element.locked](#)

## document.viewMatrix

**Availability**

Flash MX 2004.

**Usage**

```
document.viewMatrix
```

**Description**

Read-only property; a Matrix object. The `viewMatrix` is used to transform from object space to document space when the document is in edit mode. The mouse location, as a tool receives it, is relative to the object that is currently being edited. See [Matrix object](#).

For example, if you create a symbol, double-click to edit it, and draw with the PolyStar tool, the point (0,0) will be at the registration point of the symbol. However, the `drawingLayer` object expects values in document space, so if you draw a line from (0,0) using the `drawingLayer`, it will start at the upper left corner of the Stage. The `viewMatrix` property provides a way to transform from the space of the object being edited to document space.

**Example**

The following example gets the value of the `viewMatrix` property:

```
var mat = fl.getDocumentDOM().viewMatrix;
```

## document.width

**Availability**

Flash MX 2004.

**Usage**

```
document.width
```

**Description**

Property; an integer that specifies the width of the document (Stage) in pixels.

**Example**

The following example sets the width of the Stage to 400 pixels.

```
fl.getDocumentDOM().width= 400;
```

**See also**

[document.height](#)

## document.xmlPanel()

**Availability**

Flash MX 2004.

**Usage**

```
document.xmlPanel(fileURI)
```

**Parameters**

**fileURI** A string, expressed as a file:/// URI, that specifies the path to the XML file defining the controls in the panel. The full path is required.

**Returns**

An object that has properties defined for all controls defined in the XML file. All properties are returned as strings. The returned object will have one predefined property named "dismiss" that will have the string value "accept" or "cancel".

**Description**

Method; posts an XMLUI dialog box. See [f1.xmlui](#).

**Example**

The following example loads the Test.xml file and displays each property contained within it:

```
var obj = f1.getDocumentDOM().xmlPanel(f1.configURI + "Commands/Test.xml");
for (var prop in obj) {
    f1.trace("property " + prop + " = " + obj[prop]);
}
```

## document.zoomFactor

**Availability**

Flash 8.

**Usage**

```
document.zoomFactor
```

**Description**

Property; specifies the zoom percent of the Stage at authoring time. A value of 1 equals 100 percent zoom, 8 equals 800 percent, .5 equals 50 percent, and so on.

**Example**

The following example sets the zoom factor of the Stage to 200 percent.

```
f1.getDocumentDOM().zoomFactor = 2;
```

# Chapter 12: drawingLayer object

## drawingLayersummary

### Availability

Flash MX 2004.

### Description

The drawingLayer object is accessible from JavaScript as a child of the flash object. The drawingLayer object is used for extensible tools when the user wants to temporarily draw while dragging—for example, when creating a selection marquee. You should call `drawingLayer.beginFrame()` before you call any other drawingLayer methods.

### Method summary

The following methods are available for the drawingLayer object:

Method	Description
<code>drawingLayer.beginDraw()</code>	Puts Flash in drawing mode.
<code>drawingLayer.beginFrame()</code>	Erases what was previously drawn using the drawingLayer and prepares for more drawing commands.
<code>drawingLayer.cubicCurveTo()</code>	Draws a cubic curve from the current pen location using the parameters as the coordinates of the cubic segment.
<code>drawingLayer.curveTo()</code>	Draws a quadratic curve segment starting at the current drawing position and ending at a specified point.
<code>drawingLayer.drawPath()</code>	Draws the specified path.
<code>drawingLayer.endDraw()</code>	Exits drawing mode.
<code>drawingLayer.endFrame()</code>	Signals the end of a group of drawing commands.
<code>drawingLayer.lineTo()</code>	Draws a line from the current drawing position to the point (x,y).
<code>drawingLayer.moveTo()</code>	Sets the current drawing position.
<code>drawingLayer newPath()</code>	Returns a new Path object.
<code>drawingLayer.setColor()</code>	Sets the color of subsequently drawn data.
<code>drawingLayer.setFill()</code>	This method is not available.
<code>drawingLayer.setStroke()</code>	This method is not available.

## drawingLayer.beginDraw()

### Availability

Flash MX 2004.

### Usage

```
drawingLayer.beginDraw([persistentDraw])
```

**Parameters**

**persistentDraw** A Boolean value (optional). If set to `true`, it indicates that the drawing in the last frame remains on the Stage until a new `beginDraw()` or `beginFrame()` call is made. (In this context, *frame* refers to where you start and end drawing; it does not refer to timeline frames.) For example, when users draw a rectangle, they can preview the outline of the shape while dragging the mouse. If you want that preview shape to remain after the user releases the mouse button, set `persistentDraw` to `true`.

**Returns**

Nothing.

**Description**

Method; puts Flash in drawing mode. Drawing mode is used for temporary drawing while the mouse button is pressed. You typically use this method only when creating extensible tools.

**Example**

The following example puts Flash in drawing mode:

```
f1.drawingLayer.beginDraw();
```

## **drawingLayer.beginFrame()**

**Availability**

Flash MX 2004.

**Usage**

```
drawingLayer.beginFrame()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; erases what was previously drawn using the `drawingLayer` and prepares for more drawing commands. Should be called after `drawingLayer.beginDraw()`. Everything drawn between `drawingLayer.beginFrame()` and an `drawingLayer.endFrame()` remains on the Stage until you call the next `beginFrame()` and `endFrame()`. (In this context, *frame* refers to where you start and end drawing; it does not refer to timeline frames.) You typically use this method only when creating extensible tools. See [drawingLayer.beginDraw\(\)](#).

## **drawingLayer.cubicCurveTo()**

**Availability**

Flash MX 2004.

**Usage**

```
drawingLayer.cubicCurveTo(x1Ctl, y1Ctl, x2Ctl, y2Ctl, xEnd, yEnd)
```

**Parameters**

- x1Ctl** A floating-point value that is the *x* location of the first control point.
- y1Ctl** A floating-point value that is the *y* location of the first control point.
- x2Ctl** A floating-point value that is the *x* position of the middle control point.
- y2Ctl** A floating-point value that is the *y* position of the middle control point.
- xEnd** A floating-point value that is the *x* position of the end control point.
- yEnd** A floating-point value that is the *y* position of the end control point.

**Returns**

Nothing.

**Description**

Method; draws a cubic curve from the current pen location using the parameters as the coordinates of the cubic segment. You typically use this method only when creating extensible tools.

**Example**

The following example draws a cubic curve using the specified control points:

```
f1.drawingLayer.cubicCurveTo(0, 0, 1, 1, 2, 0);
```

## **drawingLayer.curveTo()**

**Availability**

Flash MX 2004.

**Usage**

```
drawingLayer.curveTo(xCtl, yCtl, xEnd, yEnd)
```

**Parameters**

- xCtl** A floating-point value that is the *x* position of the control point.
- yCtl** A floating-point value that is the *y* position of the control point.
- xEnd** A floating-point value that is the *x* position of the end control point.
- yEnd** A floating-point value that is the *y* position of the end control point.

**Returns**

Nothing.

**Description**

Method; draws a quadratic curve segment starting at the current drawing position and ending at a specified point. You typically use this method only when creating extensible tools.

**Example**

The following example draws a quadratic curve using the specified control points:

```
f1.drawingLayer.curveTo(0, 0, 2, 0);
```

## **drawingLayer.drawPath()**

**Availability**

Flash MX 2004.

**Usage**

```
drawingLayer.drawPath(path)
```

**Parameters**

**path** A Path object to draw.

**Returns**

Nothing.

**Description**

Method; draws the path specified by the *path* parameter. You typically use this method only when creating extensible tools.

**Example**

The following example draws a path specified by the Path object named `gamePath`:

```
f1.drawingLayer.drawPath(gamePath);
```

## **drawingLayer.endDraw()**

**Availability**

Flash MX 2004.

**Usage**

```
drawingLayer.endDraw()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; exits drawing mode. Drawing mode is used when you want to temporarily draw while the mouse button is pressed. You typically use this method only when creating extensible tools.

**Example**

The following example exits drawing mode:

```
f1.drawingLayer.endDraw();
```

## **drawingLayer.endFrame()**

**Availability**

Flash MX 2004.

**Usage**

```
drawingLayer.endFrame()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; signals the end of a group of drawing commands. A group of drawing commands refers to everything drawn between `drawingLayer.beginFrame()` and `drawingLayer.endFrame()`. The next call to `drawingLayer.beginFrame()` will erase whatever was drawn in this group of drawing commands. You typically use this method only when creating extensible tools.

## **drawingLayer.lineTo()**

**Availability**

Flash MX 2004.

**Usage**

```
drawingLayer.lineTo(x, y)
```

**Parameters**

**x** A floating-point value that is the *x* coordinate of the end point of the line to draw.

**y** A floating-point value that is the *y* coordinate of the end point of the line to draw.

**Returns**

Nothing.

**Description**

Method; draws a line from the current drawing position to the point (*x,y*). You typically use this method only when creating extensible tools.

**Example**

The following example draws a line from the current drawing position to the point (20,30):

```
f1.drawingLayer.lineTo(20, 30);
```

## **drawingLayer.moveTo()**

**Availability**

Flash MX 2004.

**Usage**

```
drawingLayer.moveTo(x, y)
```

**Parameters**

**x** A floating-point value that specifies the *x* coordinate of the position at which to start drawing.

**y** A floating-point value that specifies the *y* coordinate of the position at which to start drawing.

**Returns**

Nothing.

**Description**

Method; sets the current drawing position. You typically use this method only when creating extensible tools.

**Example**

The following example sets the current drawing position at the point (10,15):

```
f1.drawingLayer.moveTo(10, 15);
```

## **drawingLayer newPath()**

**Availability**

Flash MX 2004.

**Usage**

```
drawingLayer.newPath()
```

**Parameters**

None.

**Returns**

A Path object.

**Description**

Method; returns a new Path object. You typically use this method only when creating extensible tools. See [Path object](#).

**Example**

The following example returns a new Path object:

```
f1.drawingLayer newPath();
```

## **drawingLayer.setColor()**

**Availability**

Flash MX 2004.

**Usage**

```
drawingLayer.setColor(color)
```

**Parameters**

**color** The color of subsequently drawn data, in one of the following formats:

- A string in the format "#RRGGBB" or "#RRGGBAA"
- A hexadecimal number in the format 0xRRGGBB
- An integer that represents the decimal equivalent of a hexadecimal number

**Returns**

Nothing.

**Description**

Method; sets the color of subsequently drawn data. Applies only to persistent data. To use this method, the parameter passed to [drawingLayer.beginDraw\(\)](#) must be set to `true`. You typically use this method only when creating extensible tools. See [drawingLayer.beginDraw\(\)](#).

**Example**

The following example draws a red line on the Stage:

```
f1.drawingLayer.beginDraw( true );
f1.drawingLayer.beginFrame();
f1.drawingLayer.setColor( "#ff0000" );
f1.drawingLayer.moveTo(0,0);
f1.drawingLayer.lineTo(100,100);
f1.drawingLayer.endFrame();
f1.drawingLayer.endDraw();
```

## **drawingLayer.setFill()**

This method is not available.

## **drawingLayer.setStroke()**

This method is not available.

# Chapter 13: Edge object

## edge summary

### Availability

Flash MX 2004.

### Description

The Edge object represents an edge of a shape on the Stage.

### Method summary

The following methods are available for the Edge object:

Method	Description
<code>edge.getControl()</code>	Gets a point object set to the location of the specified control point of the edge.
<code>edge.getHalfEdge()</code>	Returns a <a href="#">HalfEdge object</a> .
<code>edge.setControl()</code>	Sets the position of the control point of the edge.
<code>edge.splitEdge()</code>	Splits the edge into two pieces.

### Property summary

The following properties are available for the Edge object:

Property	Description
<code>edge.cubicSegmentIndex</code>	An integer that specifies the index value of a cubic segment of the edge.
<code>edge.id</code>	Read-only; an integer that represents a unique identifier for the edge.
<code>edge.isLine</code>	Read-only; an integer with a value of 0 or 1.
<code>edge.stroke</code>	A <a href="#">Stroke object</a> .

## edge.cubicSegmentIndex

### Availability

Flash CS4 Professional.

### Usage

```
edge.cubicSegmentIndex
```

### Description

Read-only property; an integer that specifies the index value of a cubic segment of the edge (see [shape.getCubicSegmentPoints\(\)](#)).

**Example**

The following code displays the index values of all the cubic segments of the specified edge:

```
var theShape = fl.getDocumentDOM().selection[0];
var edgesArray = theShape.edges;
for(var i=0;i<edgesArray.length; i++) {
    fl.trace(edgesArray[i].cubicSegmentIndex);
}
```

## edge.getControl()

**Availability**

Flash MX 2004.

**Usage**

```
edge.getControl(i)
```

**Parameters**

**i** An integer that specifies which control point of the edge to return. Specify 0 for the first control point, 1 for the middle control point, or 2 for the end control point. If the `edge.isLine` property is `true`, the middle control point is set to the midpoint of the segment joining the beginning and ending control points.

**Returns**

The specified control point.

**Description**

Method; gets a point object set to the location of the specified control point of the edge.

**Example**

The following example stores the first control point of the specified shape in the `pt` variable:

```
var shape = fl.getDocumentDOM().selection[0];
var pt = shape.edges[0].getControl(0);
```

## edge.getHalfEdge()

**Availability**

Flash MX 2004.

**Usage**

```
edge.getHalfEdge(index)
```

**Parameters**

**index** An integer that specifies which half edge to return. The value of `index` must be either 0 for the first half edge or 1 for the second half edge.

**Returns**

A HalfEdge object.

**Description**

Method; returns a [HalfEdge object](#).

**Example**

The following example stores the half edges of the specified edge in the `hEdge0` and `hEdge1` variables:

```
var shape = fl.getDocumentDOM().selection[0];
var edge = shape.edges[0];
var hEdge0 = edge.getHalfEdge(0);
var hEdge1 = edge.getHalfEdge(1);
```

## edge.id

**Availability**

Flash MX 2004.

**Usage**

```
edge.id
```

**Description**

Read-only property; an integer that represents a unique identifier for the edge.

**Example**

The following example stores a unique identifier for the specified edge in the `my_shape_id` variable:

```
var shape = fl.getDocumentDOM().selection[0];
var my_shape_id = shape.edges[0].id;
```

## edge.isLine

**Availability**

Flash MX 2004.

**Usage**

```
edge.isLine
```

**Description**

Read-only property; an integer with a value of 0 or 1. A value of 1 indicates that the edge is a straight line. In that case, the middle control point bisects the line joining the two end points.

**Example**

The following example determines whether the specified edge is a straight line and shows a value of 1 (it is a straight line) or 0 (it isn't a straight line) in the Output panel:

```
var shape = fl.getDocumentDOM().selection[0];
fl.trace(shape.edges[0].isLine);
```

## edge.setControl()

### Availability

Flash MX 2004.

### Usage

```
edge.setControl(index, x, y)
```

### Parameters

**index** An integer that specifies which control point to set. Use values 0, 1, or 2 to specify the beginning, middle, and end control points, respectively.

**x** A floating-point value that specifies the horizontal location of the control point. If the Stage is in edit or edit-in-place mode, the point coordinate is relative to the edited object. Otherwise, the point coordinate is relative to the Stage.

**y** A floating-point value that specifies the vertical location of the control point. If the Stage is in edit or edit-in-place mode, the point coordinate is relative to the edited object. Otherwise, the point coordinate is relative to the Stage.

### Returns

Nothing.

### Description

Method; sets the position of the control point of the edge. You must call `shape.beginEdit()` before using this method. See [shape.beginEdit\(\)](#).

### Example

The following example sets the beginning control point of the specified edge to the (0, 1) coordinates:

```
x = 0; y = 1;
var shape = fl.getDocumentDOM().selection[0];
shape.beginEdit();
shape.edges[0].setControl(0, x, y);
shape.endEdit();
```

## edge.splitEdge()

### Availability

Flash MX 2004.

### Usage

```
edge.splitEdge(t)
```

**Parameters**

**t** A floating-point value between 0 and 1 that specifies where to split the edge. A value of 0 represents one end point and a value of 1 represents the other. For example, passing a value of 0.5 splits the edge in the middle, which, for a line is exactly in the center. If the edge represents a curve, 0.5 represents the parametric middle of the curve.

**Returns**

Nothing.

**Description**

Method; splits the edge into two pieces. You must call [shape.beginEdit\(\)](#) before using this method.

**Example**

The following example splits the specified edge in half:

```
var shape = fl.getDocumentDOM().selection[0];
shape.beginEdit()
shape.edges[0].splitEdge( 0.5 );
shape.endEdit()
```

## edge.stroke

**Availability**

Flash CS4 Professional.

**Usage**

`edge.stroke`

**Description**

Property; a [Stroke object](#).

**Example**

The following example displays the stroke color of the first edge of the selected object:

```
var shape = fl.getDocumentDOM().selection[0];
fl.trace(shape.edges[0].stroke.color);
```

# Chapter 14: Element object

## element summary

### Availability

Flash MX 2004.

### Description

Everything that appears on the Stage is of the type Element. The following code example lets you select an element:

```
var el = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];
```

### Method summary

The following methods are available for the Element object:

Method	Description
<code>element.getPersistentData()</code>	Retrieves the value of the data specified by the <code>name</code> parameter.
<code>element.getPublishPersistentData()</code>	True if the specified persistent data is enable for the specified format; otherwise False.
<code>element.getTransformationPoint()</code>	Gets the value of the specified element's transformation point.
<code>element.hasPersistentData()</code>	Determines whether the specified data has been attached to the specified element.
<code>element.removePersistentData()</code>	Removes any persistent data with the specified name that has been attached to the object.
<code>element.setPersistentData()</code>	Stores data with an element.
<code>element.setPublishPersistentData()</code>	Enables or disables publishing of persistent data for a specified format.
<code>element.setTransformationPoint()</code>	Sets the position of the element's transformation point.

### Property summary

The following properties are available for the Element object:

Property	Description
<code>element.depth</code>	Read-only; an integer that has a value greater than 0 for the depth of the object in the view.
<code>element.elementType</code>	Read-only; a string that represents the type of the specified element.
<code>element.height</code>	A float value that specifies the height of the element in pixels.
<code>element.layer</code>	Read-only; represents the <a href="#">Layer object</a> on which the element is located.
<code>element.left</code>	Read-only; a float value that represents the left side of the element.
<code>element.locked</code>	A Boolean value: <code>true</code> if the element is locked; <code>false</code> otherwise.
<code>element.matrix</code>	A <a href="#">Matrix object</a> . The matrix has properties <code>a</code> , <code>b</code> , <code>c</code> , <code>d</code> , <code>tx</code> , and <code>ty</code> . <code>a</code> , <code>b</code> , <code>c</code> , and <code>d</code> are floating-point values; <code>tx</code> and <code>ty</code> are coordinates.
<code>element.name</code>	A string that specifies the name of the element, normally referred to as the Instance name.

Property	Description
<code>element.rotation</code>	An integer or float value between -180 and 180 that specifies the object's clockwise rotation, in degrees.
<code>element.scaleX</code>	A float value that specifies the x scale value of symbols, drawing objects, and primitive rectangles and ovals.
<code>element.scaleY</code>	A float value that specifies the y scale value of symbols, drawing objects, and primitive rectangles and ovals.
<code>element.selected</code>	A Boolean value that specifies whether the element is selected or not.
<code>element.skewX</code>	A float value between -180 and 180 that specifies the x skew value of symbols, drawing objects, and primitive rectangles and ovals.
<code>element.skewY</code>	A float value between -180 and 180 that specifies the y skew value of symbols, drawing objects, and primitive rectangles and ovals.
<code>element.top</code>	Read-only; top side of the element.
<code>element.transformX</code>	A floating-point number that specifies the x value of the selected element's transformation point, within the coordinate system of the element's parent.
<code>element.transformY</code>	A floating-point number that specifies the y value of the selected element's transformation point, within the coordinate system of the element's parent.
<code>element.width</code>	A float value that specifies the width of the element in pixels.
<code>element.x</code>	A float value that specifies the x value of the selected element's registration point.
<code>element.y</code>	A float value that specifies the y value of the selected element's registration point.

## element.depth

### Availability

Flash MX 2004.

### Usage

`element.depth`

### Description

Read-only property; an integer that has a value greater than 0 for the depth of the object in the view. The drawing order of objects on the Stage specifies which one is on top of the others. Object order can also be managed with the Modify > Arrange menu item.

### Example

The following example displays the depth of the specified element in the Output panel:

```
// Select an object and run this script.  
fl.trace("Depth of selected object: " + fl.getDocumentDOM().selection[0].depth);
```

See the example for `element.elementType`.

## element.elementType

### Availability

Flash MX 2004.

### Usage

```
element.elementType
```

### Description

Read-only property; a string that represents the type of the specified element. The value is one of the following:

- "shape"
- "text"
- "tlfText" (Flash Pro CS5 and later)
- "instance"
- "shapeObj"

### Example

The following example stores the type of the first element in the eType variable:

```
// In a new file, place a movie clip on first frame top layer, and
// then run this line of script.
var eType = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].elementType; // eType = instance
```

The following example displays several properties for all the elements in the current layer or frame:

```
var tl = fl.getDocumentDOM().getTimeline()
var elts = tl.layers[tl.currentLayer].frames[tl.currentFrame].elements;
for (var x = 0; x < elts.length; x++) {
    var elt = elts[x];
    fl.trace("Element " + x + " Name = " + elt.name + " Type = " + elt.elementType + " location
= " + elt.left + "," + elt.top + " Depth = " + elt.depth);
}
```

## element.getPersistentData()

### Availability

Flash MX 2004.

### Usage

```
element.getPersistentData(name)
```

### Parameters

**name** A string that identifies the data to be returned.

### Returns

The data specified by the *name* parameter, or 0 if the data doesn't exist.

**Description**

Method; retrieves the value of the data specified by the *name* parameter. The type of data depends on the type of the data that was stored (see [element.setPersistentData\(\)](#)). Only symbols and bitmaps support persistent data.

**Example**

The following example sets and gets data for the specified element, shows its value in the Output panel, and then removes the data:

```
// At least one symbol or bitmap is selected in the first layer, first frame.  
var elt = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];  
elt.setPersistentData("myData", "integer", 12);  
if (elt.hasPersistentData("myData")){  
    fl.trace("myData = "+ elt.getPersistentData("myData"));  
    elt.removePersistentData( "myData" );  
    fl.trace("myData = "+ elt.getPersistentData("myData"));  
}
```

## element.getPublishPersistentData()

**Availability**

Flash Professional CC.

**Usage**

```
element.getPublishPersistentData(name, format)
```

**Parameters**

**name** A string that specifies the name of the persistent data item (set with [element.setPersistentData\(\)](#)).

**format** A string that specifies the publishing format.

**Note:** *\_EMBED\_SWF* is a special built-in publishing format for persistent data. If set, the persistent data will be embedded in the SWF file every time a document is published. The persistent data can then be accessed via ActionScript with the *.metaData* property. This requires SWF version 19 (Flash Player 11.6) and above and is only for symbol instances onstage. Other custom publishing formats may be specified for custom JSFL scripts if this method is called with the same format.

**Returns**

Boolean; True if the specified persistent data is enabled for the specified format. Otherwise False.

**Description**

Method; Indicates whether publishing of a specified persistent data item is enabled for the specified format on an element.

**Example**

The following example illustrates the use of this method:

```
var doc = fl.getDocumentDOM();
// set the data
if (doc) {
    // get the first selected element
    var elem = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];
    if (elem) {
        // add persistent data "myAlign" of "left"
        elem.setPersistentData( "myAlign", "string", "left" );
        // enable publishing of persistent data to SWF for the element
        elem.setPublishPersistentData("myAlign", "_EMBED_SWF_", true);
        // enable publishing to SWF for entire document
        doc.setPublishDocumentData("_EMBED_SWF_", true);
    }
}
// example getting data
if (doc && doc.getPublishDocumentData("_EMBED_SWF_")) {
    // get the first selected element
    var elem = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];
    if (elem && elem.hasPersistentData("myAlign") && elem.getPublishPersistentData("myAlign",
"_EMBED_SWF_")) {
        alert("elem.metaData.myAlign = '" + elem.getPersistentData("myAlign") + "' will be
embedded in SWF when published.");
    }
}
```

**See also**

[element.setPublishPersistentData\(\)](#)

## element.getTransformationPoint()

**Availability**

Flash CS3 Professional.

**Usage**

`element.getTransformationPoint()`

**Parameters**

None.

**Returns**

A point (for example, {`x:10, y:20`}, where `x` and `y` are floating-point numbers) that specifies the position of the transformation point (also *origin point* or *zero point*) within the element's coordinate system.

**Description**

Method; gets the value of the specified element's transformation point.

Transformation points are relative to different locations, depending on the type of item selected. For more information, see [element.setTransformationPoint\(\)](#).

**Example**

The following example gets the transformation point for the third element in the ninth frame on the first layer in the document. The `transPoint.x` property gives the `x` coordinate of the transformation point. The `transPoint.y` property gives the `y` coordinate of the transformation point.

```
var transPoint =  
fl.getDocumentDOM().getTimeline().layers[0].frames[8].elements[2].getTransformationPoint();
```

**See also**

[document.getTransformationPoint\(\)](#), [element.setTransformationPoint\(\)](#), [element.transformX](#),  
[element.transformY](#)

## element.hasPersistentData()

**Availability**

Flash MX 2004.

**Usage**

```
element.hasPersistentData(name)
```

**Parameters**

**name** A string that specifies the name of the data item to test.

**Returns**

A Boolean value: `true` if the specified data is attached to the object; `false` otherwise.

**Description**

Method; determines whether the specified data has been attached to the specified element. Only symbols and bitmaps support persistent data.

**Example**

See [element.getPersistentData\(\)](#).

## element.height

**Availability**

Flash MX 2004.

**Usage**

```
element.height
```

**Description**

Property; a float value that specifies the height of the element in pixels.

Do not use this property to resize a text field. Instead, select the text field and use [document.setTextRectangle\(\)](#). Using this property with a text field scales the text.

**Example**

The following example sets the height of the specified element to 100:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].height = 100;
```

## element.layer

**Availability**

Flash 8.

**Usage**

```
element.layer
```

**Description**

Read-only property; represents the [Layer object](#) on which the element is located.

**Example**

The following example stores the Layer object that contains the element in the `theLayer` variable:

```
var theLayer = element.layer;
```

## element.left

**Availability**

Flash MX 2004.

**Usage**

```
element.left
```

**Description**

Read-only property; a float value that represents the left side of the element. The value of `element.left` is relative to the upper left of the Stage for elements that are in a scene and is relative to the symbol's registration point (also *origin point* or *zero point*) if the element is stored within a symbol. Use [document.setSelectionBounds\(\)](#) or [document.moveSelectionBy\(\)](#) to set this property.

**Example**

The following example illustrates how the value of this property changes when an element is moved:

```
// Select an element on the Stage and then run this script.  
var sel = f1.getDocumentDOM().selection[0];  
f1.trace("Left (before) = " + sel.left);  
f1.getDocumentDOM().moveSelectionBy({x:100, y:0});  
f1.trace("Left (after) = " + sel.left);
```

See the [element.elementType](#) example.

## element.locked

### Availability

Flash MX 2004.

### Usage

```
element.locked
```

### Description

Property; a Boolean value: `true` if the element is locked; `false` otherwise. If the value of `element.elementType` is "shape", this property is ignored.

### Example

The following example locks the first element in the first frame, top layer:

```
// Similar to Modify > Arrange > Lock:  
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].locked = true;
```

## element.matrix

### Availability

Flash MX 2004.

### Usage

```
element.matrix
```

### Description

Property; a Matrix object. A matrix has properties `a`, `b`, `c`, `d`, `tx`, and `ty`. The `a`, `b`, `c`, and `d` properties are floating-point values; the `tx` and `ty` properties are coordinates. See [Matrix object](#).

### Example

The following example moves the specified element by 10 pixels in `x` and 20 pixels in `y`:

```
var mat = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].matrix;  
mat.tx += 10;  
mat.ty += 20;  
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].matrix = mat;
```

## element.name

### Availability

Flash MX 2004.

### Usage

```
element.name
```

**Description**

Property; a string that specifies the name of the element, normally referred to as the Instance name. If the value of `element.elementType` is "shape", this property is ignored. See [element.elementType](#).

**Example**

The following example sets the Instance name of the first element in Frame 1, top layer to "clip\_mc":

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].name = "clip_mc";
```

See the [element.elementType](#) example.

## element.removePersistentData()

**Availability**

Flash MX 2004.

**Usage**

```
element.removePersistentData(name)
```

**Parameters**

**name** A string that specifies the name of the data to remove.

**Returns**

A Boolean value: `true` if data was removed; `false` otherwise.

**Description**

Method; removes any persistent data with the specified name that has been attached to the object. Only symbols and bitmaps support persistent data.

**Example**

See [element.getPersistentData\(\)](#).

## element.rotation

**Availability**

Flash CS3 Professional.

**Usage**

```
element.rotation
```

**Description**

Property; an integer or float value between -180 and 180 that specifies the object's clockwise rotation, in degrees.

**Example**

The following example sets the currently selected element's rotation to 45 degrees:

```
var element = fl.getDocumentDOM().selection[0];
fl.trace("Element rotation = " + element.rotation);
element.rotation = 45;
fl.trace("After setting rotation to 45: rotation = " + element.rotation);
```

## element.scaleX

### Availability

Flash CS3 Professional.

### Usage

```
element.scaleX
```

### Description

Property; a float value that specifies the *x* scale value of symbols, drawing objects, and primitive rectangles and ovals. A value of 1 indicates 100 percent scale.

### Example

The following example sets the *x* scale value of the current selection to 2 (doubles its value):

```
var element = fl.getDocumentDOM().selection[0];
element.scaleX = 2;
```

### See also

[element.scaleY](#)

## element.scaleY

### Availability

Flash CS3 Professional.

### Usage

```
element.scaleY
```

### Description

Property; a float value that specifies the *y* scale value of symbols, drawing objects, and primitive rectangles and ovals. A value of 1 indicates 100 percent scale.

### Example

The following example sets the *y* scale value of the current selection to 2 (doubles its value):

```
var element = fl.getDocumentDOM().selection[0];
element.scaleY = 2;
```

### See also

[element.scaleX](#)

## element.selected

### Availability

Flash 8.

### Usage

```
element.selected
```

### Description

Property; a Boolean value that specifies whether the element is selected (`true`) or not (`false`).

### Example

The following example selects the element:

```
element.selected = true;
```

## element.setPersistentData()

### Availability

Flash MX 2004.

### Usage

```
element.setPersistentData(name, type, value)
```

### Parameters

**name** A string that specifies the name to associate with the data. This name is used to retrieve the data.

**type** A string that defines the type of the data. The allowable values are "integer", "integerArray", "double", "doubleArray", "string", and "byteArray".

**value** Specifies the value to associate with the object. The data type of *value* depends on the value of the *type* parameter. The specified value should be appropriate to the data type specified by the *type* parameter.

### Returns

Nothing.

### Description

Method; stores data with an element. The data is available when the FLA file containing the element is reopened. Only symbols and bitmaps support persistent data.

### Example

See [element.getPersistentData\(\)](#).

## element.setPublishPersistentData()

### Availability

Flash Professional CC.

### Usage

```
element.setPublishPersistentData(name, format, publish)
```

### Parameters

**name** A string that specifies the name of the persistent data item (set with `element.setPersistentData()`).

**format** A string that specifies the publishing format.

**Note:** `_EMBED_SWF_` is a special built-in publishing format for persistent data. If set, the persistent data will be embedded in the SWF file every time a document is published. The persistent data can then be accessed via ActionScript with the `.metaData` property. This requires SWF version 19 (Flash Player 11.6) and above and is only for symbol instances onstage. Other custom publishing formats may be specified for custom JSFL scripts if this method is called with the same format.

**publish** A boolean that indicates whether to enable or disable publishing of persistent data for the specified format.

### Returns

None.

### Description

Method; Enables or disables publishing of persistent data for a specified format.

### Example

The following example illustrates the use of this method:

```
var doc = fl.getDocumentDOM();
// set the data if (doc) {
    // get the first selected element
    var elem = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];
    if (elem) {
        // add persistent data "myAlign" of "left"
        elem.setPersistentData("myAlign", "string", "left");
        // enable publishing of persistent data to SWF for the element
        elem.setPublishPersistentData("myAlign", "_EMBED_SWF_", true);
        // enable publishing to SWF for entire document
        doc.setPublishDocumentData("_EMBED_SWF_", true);
    }
}
// example getting data
if (doc && doc.getPublishDocumentData("_EMBED_SWF_")) {
    // get the first selected element
    var elem = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];
    if (elem && elem.hasPersistentData("myAlign") && elem.getPublishPersistentData("myAlign", "_EMBED_SWF_")) {
        alert("elem.metaData.myAlign = '" + elem.getPersistentData("myAlign") + "' will be
embedded in SWF when published.");
    }
}
```

**See also**`element.getPublishPersistentData()`

## element.setTransformationPoint()

**Availability**

Flash CS3 Professional.

**Usage**`element.setTransformationPoint(transformationPoint)`**Parameters**

**transformationPoint** A point (for example, `{x:10, y:20}`, where x and y are floating-point numbers) that specifies values for an element's or group's transformation point.

- Shapes: *transformationPoint* is set relative to the document (0,0 is the upper-left corner of the Stage).
- Symbols: *transformationPoint* is set relative to the symbol's registration point (0,0 is located at the registration point).
- Text: *transformationPoint* is set relative to the text field (0,0 is the upper-left corner of the text field).
- Bitmaps/videos: *transformationPoint* is set relative to the bitmap/video (0,0 is the upper-left corner of the bitmap or video).
- Drawing objects, primitive objects, and groups: *transformationPoint* is set relative to the stage.

**Returns**

Nothing.

**Description**

Method; sets the position of the element's transformation point.

This method is almost identical to `document.setTransformationPoint()`. It is different in the following way:

- You can set transformation points for elements without first selecting them.

This method moves the transformation point but does not move the element. By contrast, the `element.transformX` and `element.transformY` properties move the element.

**Example**

The following example sets the transformation point of the third element on the Stage to 100, 200:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[2].setTransformationPoint({x: 100, y:200});
```

**See also**`document.setTransformationPoint()`, `element.getTransformationPoint()`, `element.transformX`, `element.transformY`

## element.skewX

### Availability

Flash CS3 Professional.

### Usage

```
element.skewX
```

### Description

Property; a float value between -180 and 180 that specifies the *x* skew value of symbols, drawing objects, and primitive rectangles and ovals.

### Example

The following example sets the *x* skew value of the current selection to 10:

```
var element = fl.getDocumentDOM().selection[0];
element.skewX = 10;
```

### See also

[document.setTransformationPoint\(\)](#), [element.skewY](#)

## element.skewY

### Availability

Flash CS3 Professional.

### Usage

```
element.skewY
```

### Description

Property; a float value between -180 and 180 that specifies the *y* skew value of symbols, drawing objects, and primitive rectangles and ovals.

### Example

The following example sets the *y* skew value of the current selection to 10:

```
var element = fl.getDocumentDOM().selection[0];
element.skewY = 10;
```

### See also

[document.setTransformationPoint\(\)](#), [element.skewX](#)

## element.top

### Availability

Flash MX 2004.

### Usage

```
element.top
```

### Description

Read-only property; top side of the element. The value of `element.top` is relative to the upper left of the Stage for elements that are in a scene and is relative to the symbol's registration point if the element is stored within a symbol. Use `document.setSelectionBounds()` or `document.moveSelectionBy()` to set this property.

### Example

The following example shows how the value of this property changes when an element is moved:

```
// Select an element on the Stage and then run this script.  
var sel = fl.getDocumentDOM().selection[0];  
fl.trace("Top (before) = " + sel.top);  
fl.getDocumentDOM().moveSelectionBy({x:0, y:100});  
fl.trace("Top (after) = " + sel.top);
```

See the [element.elementType](#) example.

## element.transformX

### Availability

Flash CS3 Professional.

### Usage

```
element.transformX
```

### Description

Property; a floating-point number that specifies the *x* value of the selected element's transformation point, within the coordinate system of the element's parent. Setting this property to a new value moves the element. By contrast, the `element.setTransformationPoint()` method moves the transformation point but does not move the element.

### Example

### See also

[element.getTransformationPoint\(\)](#), [element.setTransformationPoint\(\)](#), [element.transformY](#)

## element.transformY

### Availability

Flash CS3 Professional.

**Usage**

```
element.transformY
```

**Description**

Property; a floating-point number that specifies the *y* value of the selected element's transformation point, within the coordinate system of the element's parent. Setting this property to a new value moves the element. By contrast, the [element.setTransformationPoint\(\)](#) method moves the transformation point but does not move the element.

**See also**

[element.getTransformationPoint\(\)](#), [element.setTransformationPoint\(\)](#), [element.transformX](#)

## element.width

**Availability**

Flash MX 2004.

**Usage**

```
element.width
```

**Description**

Property; a float value that specifies the width of the element in pixels.

Do not use this property to resize a text field. Instead, select the text field and use [document.setTextRectangle\(\)](#). Using this property with a text field scales the text.

**Example**

The following example sets the width of the specified element to 100:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].width= 100;
```

## element.x

**Availability**

Flash CS3 Professional.

**Usage**

```
element.x
```

**Description**

Property; a float value that specifies the *x* value of the selected element's registration point.

**Example**

The following example sets the value of the specified element's registration point to 100, 200:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].x= 100;  
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].y= 200;
```

**See also**

[element.y](#)

## **element.y**

**Availability**

Flash CS3 Professional.

**Usage**

`element.y`

**Description**

Property; a float value that specifies the *y* value of the selected element's registration point.

**Example**

See [element.x](#)

# Chapter 15: Fill object

## fill summary

### Availability

Flash MX 2004.

### Description

This object contains all the properties of the Fill color setting of the Tools panel or of a selected shape. To retrieve a Fill object, use `document.getCustomFill()`.

### Property summary

The following properties are available for the Fill object:

Property	Description
<code>fill.bitmapIsClipped</code>	A Boolean value that specifies whether the bitmap fill for a shape that is larger than the bitmap is clipped or repeated.
<code>fill.bitmapPath</code>	A string that specifies the path and name of the bitmap fill in the Library.
<code>fill.color</code>	A string, hexadecimal value, or integer that represents the fill color.
<code>fill.colorArray</code>	An array of colors in gradient.
<code>fill.focalPoint</code>	An integer that specifies the gradient focal point horizontal offset from the transformation point.
<code>fill.linearRGB</code>	A Boolean value that specifies whether to render the fill as a linear or radial RGB gradient.
<code>fill.matrix</code>	A <a href="#">Matrix object</a> that defines the placement, orientation, and scales for gradient fills.
<code>fill.overflow</code>	A string that specifies the behavior of a gradient's overflow.
<code>fill.posArray</code>	An array of integers, each in the range of zero to 255, indicating the position of the corresponding color.
<code>fill.style</code>	A string that specifies the fill style.

## fill.bitmapIsClipped

### Availability

Flash CS4 Professional.

### Usage

```
fill.bitmapIsClipped
```

### Description

Property; a Boolean value that specifies whether the bitmap fill for a shape that is larger than the bitmap is clipped (`true`) or repeated (`false`). This property is available only if the value of the `fill.style` property is "bitmap". If the shape is smaller than the bitmap, this value is `false`.

**Example**

The following example displays information on whether the bitmap fill is clipped, if appropriate, in the Output panel:

```
var fill = fl.getDocumentDOM().getCustomFill();
if (fill.style == "bitmap")
    fl.trace("Fill image is clipped: " + fill.bitmapIsClipped);
```

**See also**

[fill.bitmapPath](#)

## fill.bitmapPath

**Availability**

Flash CS4 Professional.

**Usage**

`fill.bitmapPath`

**Description**

Property; a string that specifies the path and name of the bitmap fill in the Library. This property is available only if the value of the `fill.style` property is "bitmap".

**Example**

The following example sets the fill style of the specified item to a bitmap image in the Library:

```
var fill = fl.getDocumentDOM().getCustomFill();
fill.style = "bitmap";
fill.bitmapPath = "myBitmap.jpg";
fl.getDocumentDOM().setCustomFill(fill);
```

**See also**

[fill.bitmapIsClipped](#)

## fill.color

**Availability**

Flash MX 2004.

**Usage**

`fill.color`

**Description**

Property; the color of the fill, in one of the following formats:

- A string in the format "#RRGGBB" or "#RRGGBAA"
- A hexadecimal number in the format 0xRRGGBB

- An integer that represents the decimal equivalent of a hexadecimal number

**Example**

The following example sets the fill color of the current selection:

```
var fill = fl.getDocumentDOM().getCustomFill();
fill.color = "#FFFFFF";
fl.getDocumentDOM().setCustomFill( fill );
```

## fill.colorArray

**Availability**

Flash MX 2004.

**Usage**

```
fill.colorArray
```

**Description**

Property; an array of colors in the gradient, expressed as integers. This property is available only if the value of the `fill.style` property is either "radialGradient" or "linearGradient". See [fill.style](#)

**Example**

The following example displays the color array of the current selection, if appropriate, in the Output panel:

```
var fill = fl.getDocumentDOM().getCustomFill();
if(fill.style == "linearGradient" || fill.style == "radialGradient")
    alert(fill.colorArray);
```

The following example sets the fill to the specified linear gradient:

```
var fill = fl.getDocumentDOM().getCustomFill();
fill.style = "linearGradient";
fill.colorArray = ["#00ff00","#ff00ff"];
fill.posArray = [0, 255];
fl.getDocumentDOM().setCustomFill(fill);
```

## fill.focalPoint

**Availability**

Flash 8.

**Usage**

```
fill.focalPoint
```

**Description**

Property; an integer that specifies the gradient focal point horizontal offset from the transformation point. A value of 10, for example, would place the focal point at 10/255 of the distance from the transformation point to the edge of the gradient. A value of -255 would place the focal point at the left boundary of the gradient. The default value is 0.

This property is available only if the value of the `fill.style` property is "radialGradient".

### Example

The following example sets the focal point of a radial gradient for the current selection to 100 pixels to the right of the shape's center:

```
var fill = fl.getDocumentDOM().getCustomFill();
fill.style = "radialGradient";
fill.colorArray = ["#00ff00","#ff00ff"];
fill.posArray = [0, 255];
fill.focalPoint = 10100;
fl.getDocumentDOM().setCustomFill(fill);
```

## fill.linearRGB

### Availability

Flash 8.

### Usage

```
fill.linearRGB
```

### Description

Property; a Boolean value that specifies whether to render the fill as a linear or radial RGB gradient. Set this property to `true` to specify a linear interpolation of a gradient; set it to `false` to specify a radial interpolation of a gradient. The default value is `false`.

### Example

The following example specifies that the gradient of the current selection should be rendered with a linear RGB:

```
var fill = fl.getDocumentDOM().getCustomFill();
fill.linearRGB style = true"radialGradient";
fill.colorArray = ["#00ff00","#ff00ff"];
fill.posArray = [0, 255];
fill.focalPoint = 100;
fill.linearRGB = true;
fl.getDocumentDOM().setCustomFill(fill);
```

## fill.matrix

### Availability

Flash MX 2004.

### Usage

```
fill.matrix
```

### Description

Property; a [Matrix object](#) that defines the placement, orientation, and scales for gradient fills.

### Example

The following example uses the `fill.matrix` property to specify a gradient fill for the current selection:

```
var fill = fl.getDocumentDOM().getCustomFill();
fill.style = 'radialGradient';
fill.colorArray = ['#00ff00', '#ff00ff'];
fill.posArray = [0, 255];
fill.focalPoint = 100;
fill.linearRGB = false;
fill.overflow = 'repeat';
var mat = fill.matrix;
mat.a = 0.0167083740234375;
mat.b = -0.0096435546875;
mat.c = 0.0312957763671875;
mat.d = 0.05419921875;
mat.tx = 288.65;
mat.ty = 193.05;
for (i in mat) {
    fl.trace(i+' : '+mat[i]);
}
fl.getDocumentDOM().setCustomFill(fill);
```

## fill.overflow

### Availability

Flash 8.

### Usage

```
fill.overflow
```

### Description

Property; a string that specifies the behavior of a gradient's overflow. Acceptable values are "extend", "repeat", and "reflect"; the strings are not case-sensitive. The default value is "extend".

### Example

The following example specifies that the behavior of the overflow for the current selection should be "extend":

```
var fill = fl.getDocumentDOM().getCustomFill();
fill.overflow = "extend";
fl.getDocumentDOM().setCustomFill(fill);
```

## fill.posArray

### Availability

Flash MX 2004.

### Usage

```
fill.posArray
```

### Description

Property; an array of integers, each in the range of zero to 255, indicating the position of the corresponding color. This property is available only if the value of the `fill.style` property is either "radialGradient" or "linearGradient".

### Example

The following example specifies the colors to use in a linear gradient for the current selection:

```
var fill = fl.getDocumentDOM().getCustomFill();
fill.style = "linearGradient";
fill.colorArray = [ 0x00ff00, 0xff0000, 0x0000ff ];
fill.posArray= [0,100, 200];
fl.getDocumentDOM().setCustomFill( fill );
```

## fill.style

### Availability

Flash MX 2004. Value "bitmap" added in Flash CS4 Professional.

### Usage

```
fill.style
```

### Description

Property; a string that specifies the fill style. Acceptable values are "bitmap", "solid", "linearGradient", "radialGradient", and "noFill".

If this value is "linearGradient" or "radialGradient", the `fill.colorArray` and `fill.posArray` properties are also available. If this value is "bitmap", the `fill.bitmapIsClipped` and `fill.bitmapPath` properties are also available.

### Example

The following example specifies the colors to use in a linear gradient for the current selection:

```
var fill = fl.getDocumentDOM().getCustomFill();
fill.style= "linearGradient";
fill.colorArray = [ 0x00ff00, 0xff0000, 0x0000ff ];
fill.posArray= [0,100, 200];
fl.getDocumentDOM().setCustomFill( fill );
```

# Chapter 16: Filter object

## filter summary

### Availability

Flash 8.

### Description

This object contains all the properties for all filters. The `filter.name` property specifies the type of filter, and determines which properties are applicable to each filter. See [filter.name](#).

To return the filter list for an object or objects, use `document.getFilters()`. To apply filters to an object or objects, use `document.setFilters()`. See [document.getFilters\(\)](#) and [document.setFilters\(\)](#).

### Property summary

The following properties can be used with the Filter object:

Property	Description
<code>filter.angle</code>	A float value that specifies the angle of the shadow or highlight color, in degrees.
<code>filter.blurX</code>	A float value that specifies the amount to blur in the x direction, in pixels.
<code>filter.blurY</code>	A float value that specifies the amount to blur in the y direction.
<code>filter.brightness</code>	A float value that specifies the brightness of the filter.
<code>filter.color</code>	A string, hexadecimal value, or integer that represents the filter color.
<code>filter.contrast</code>	A float value that specifies the contrast value of the filter.
<code>filter.distance</code>	A float value that specifies the distance between the filter's effect and an object, in pixels.
<code>filter.enabled</code>	A Boolean value that specifies whether the specified filter is enabled.
<code>filter.hideObject</code>	A Boolean value that specifies whether the source image is hidden.
<code>filter.highlightColor</code>	A string, hexadecimal value, or integer that represents the highlight color.
<code>filter.hue</code>	A float value that specifies the hue of the filter.
<code>filter.inner</code>	A Boolean value that specifies whether the shadow is an inner shadow.
<code>filter.knockout</code>	A Boolean value that specifies whether the filter is a knockout filter.
<code>filter.name</code>	Read-only; a string that specifies the type of filter.
<code>filter.quality</code>	A string that specifies the blur quality.
<code>filter.saturation</code>	A float value that specifies the saturation value of the filter.
<code>filter.shadowColor</code>	A string, hexadecimal value, or integer that represents the shadow color.
<code>filter.strength</code>	An integer that specifies the percentage strength of the filter.
<code>filter.type</code>	A string that specifies the type of bevel or glow.

## filter.angle

### Availability

Flash 8.

### Usage

```
filter.angle
```

### Description

Property; a float value that specifies the angle of the shadow or highlight color, in degrees. Acceptable values are between 0 and 360. This property is defined for Filter objects with a value of "bevelFilter", "dropShadowFilter", "gradientBevelFilter", or "gradientGlowFilter" for the [filter.name](#) property.

### Example

The following example sets the angle to 120 for the Bevel filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++) {
    if(myFilters[i].name == 'bevelFilter'){
        myFilters[i].angle = 120;
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

### See also

[document.setFilterProperty\(\)](#)

## filter.blurX

### Availability

Flash 8.

### Usage

```
filter.blurX
```

### Description

Property; a float value that specifies the amount to blur in the x direction, in pixels. Acceptable values are between 0 and 255. This property is defined for Filter objects with a value of "bevelFilter", "blurFilter", "dropShadowFilter", "glowFilter", "gradientBevelFilter", or "gradientGlowFilter" for the [filter.name](#) property.

### Example

The following example sets the blurX value to 30 and the blurY value to 20 for the Blur filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++){
    if(myFilters[i].name == 'blurFilter'){
        myFilters[i].blurX = 30;
        myFilters[i].blurY = 20;
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

**See also**

[document.setFilterProperty\(\)](#), [filter.blurY](#)

## filter.blurY

**Availability**

Flash 8.

**Usage**

`filter.blurY`

**Description**

Property; a float value that specifies the amount to blur in the *y* direction, in pixels. Acceptable values are between 0 and 255. This property is defined for Filter objects with a value of "bevelFilter", "blurFilter", "dropShadowFilter", "glowFilter", "gradientBevelFilter", or "gradientGlowFilter" for the `filter.name` property.

**Example**

See [filter.blurX](#).

**See also**

[document.setFilterProperty\(\)](#), [filter.blurX](#)

## filter.brightness

**Availability**

Flash 8.

**Usage**

`filter.brightness`

**Description**

Property; a float value that specifies the brightness of the filter. Acceptable values are between -100 and 100. This property is defined for Filter objects with a value of "adjustColorFilter" for the `filter.name` property.

**Example**

The following example sets the brightness to 30.5 for the Adjust Color filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++){
    if(myFilters[i].name == 'adjustColorFilter'){
        myFilters[i].brightness = 30.5;
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

## filter.color

**Availability**

Flash 8.

**Usage**

```
filter.color
```

**Description**

Property; the color of the filter, in one of the following formats:

- A string in the format "#RRGGBB" or "#RRGGBBAA"
- A hexadecimal number in the format 0xRRGGBB
- An integer that represents the decimal equivalent of a hexadecimal number

This property is defined for Filter objects with a value of "dropShadowFilter" or "glowFilter" for the [filter.name](#) property.

**Example**

The following example sets the color to "#ff00003e" for the Drop Shadow filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++){
    if(myFilters[i].name == 'dropShadowFilter'){
        myFilters[i].color = '#ff00003e';
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

**See also**

[document.setFilterProperty\(\)](#)

## filter.contrast

**Availability**

Flash 8.

**Usage**

```
filter.contrast
```

**Description**

Property; a float value that specifies the contrast value of the filter. Acceptable values are between -100 and 100. This property is defined for Filter objects with a value of "adjustColorFilter" for the [filter.name](#) property.

**Example**

The following example sets the contrast value to -15.5 for the Adjust Color filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++){
    if(myFilters[i].name == 'adjustColorFilter'){
        myFilters[i].contrast = -15.5;
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

## filter.distance

**Availability**

Flash 8.

**Usage**

```
filter.distance
```

**Description**

Property; a float value that specifies the distance between the filter's effect and an object, in pixels. Acceptable values are from -255 to 255. This property is defined for Filter objects with a value of "bevelFilter", "dropShadowFilter", "gradientBevelFilter", or "gradientGlowFilter" for the [filter.name](#) property.

**Example**

The following example sets the distance to 10 pixels for the Drop Shadow filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++){
    if(myFilters[i].name == 'dropShadowFilter'){
        myFilters[i].distance = 10;
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

**See also**

[document.setFilterProperty\(\)](#)

## filter.enabled

### Availability

Flash CS3 Professional.

### Usage

```
filter.enabled
```

### Description

Property; a Boolean value that specifies whether the specified filter is enabled (`true`) or disabled (`false`).

### Example

The following example disables the Color filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++) {
    if(myFilters[i].name == 'adjustColorFilter'){
        myFilters[i].enabled = false;
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

## filter.hideObject

### Availability

Flash 8.

### Usage

```
filter.hideObject
```

### Description

Property; a Boolean value that specifies whether the source image is hidden (`true`) or displayed (`false`). This property is defined for Filter objects with a value of "dropShadowFilter" for the `filter.name` property.

### Example

The following example sets the `hideObject` value to `true` for the Drop Shadow filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++) {
    if(myFilters[i].name == 'dropShadowFilter'){
        myFilters[i].hideObject = true;
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

## filter.highlightColor

### Availability

Flash 8.

### Usage

```
filter.highlightColor
```

### Description

Property; the color of the highlight, in one of the following formats:

- A string in the format "#RRGGBB" or "#RRGGBBAA"
- A hexadecimal number in the format 0xRRGGBB
- An integer that represents the decimal equivalent of a hexadecimal number

This property is defined for Filter objects with a value of "bevelFilter" for the `filter.name` property.

### Example

The following example sets the highlight color to "#ff00003e" for the Bevel filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++){
    if(myFilters[i].name == 'bevelFilter'){
        myFilters[i].highlightColor = '#ff00003e';
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

## filter.hue

### Availability

Flash 8.

### Usage

```
filter.hue
```

### Description

Property; a float value that specifies the hue of the filter. Acceptable values are between -180 and 180. This property is defined for Filter objects with a value of "adjustColorFilter" for the `filter.name` property.

### Example

The following example sets the hue to 120 for the Adjust Color filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++){
    if(myFilters[i].name == 'adjustColorFilter'){
        myFilters[i].hue = 120;
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

## filter.inner

### Availability

Flash 8.

### Usage

```
filter.inner
```

### Description

Property; a Boolean value that specifies whether the shadow is an inner shadow (`true`) or not (`false`). This property is defined for Filter objects with a value of "dropShadowFilter" or "glowFilter" for the [filter.name](#) property.

### Example

The following example sets the value of the `inner` property to `true` for the Glow filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++){
    if(myFilters[i].name == 'glowFilter'){
        myFilters[i].inner = true;
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

### See also

[document.setFilterProperty\(\)](#)

## filter.knockout

### Availability

Flash 8.

### Usage

```
filter.knockout
```

### Description

Property; a Boolean value that specifies whether the filter is a knockout filter (`true`) or not (`false`). This property is defined for Filter objects with a value of "bevelFilter", "dropShadowFilter", "glowFilter", "gradientBevelFilter", or "gradientGlowFilter" for the [filter.name](#) property.

**Example**

The following example sets the knockout property to true for the Glow filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++){
    if(myFilters[i].name == 'glowFilter'){
        myFilters[i].knockout = true;
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

**See also**

[document.setFilterProperty\(\)](#)

## filter.name

**Availability**

Flash 8.

**Usage**

`filter.name`

**Description**

Read-only property; a string that specifies the type of filter. The value of this property determines which other properties of the Filter object are available. The value is one of the following: "adjustColorFilter", "bevelFilter", "blurFilter", "dropShadowFilter", "glowFilter", "gradientBevelFilter", or "gradientGlowFilter".

**Example**

The following example displays the filter names and index positions in the Output panel:

```
var myFilters = fl.getDocumentDOM().getFilters();
var traceStr = "";
for(i=0; i < myFilters.length; i++){
    traceStr = traceStr + " At index " + i + ": " + myFilters[i].name;
}
fl.trace(traceStr);
```

**See also**

[document.getFilters\(\)](#), [document.setFilterProperty\(\)](#)

## filter.quality

**Availability**

Flash 8.

**Usage**

```
filter.quality
```

**Description**

Property; a string that specifies the blur quality. Acceptable values are "low", "medium", and "high" ("high" is similar to a Gaussian blur). This property is defined for Filter objects with a value of "bevelFilter", "blurFilter", "dropShadowFilter", "glowFilter", "gradientGlowFilter", or "gradientBevelFilter" for the [filter.name](#) property.

**Example**

The following example sets the blur quality to "medium" for the Glow filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++){
    if(myFilters[i].name == 'glowFilter'){
        myFilters[i].quality = 'medium';
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

**See also**

[document.setFilterProperty\(\)](#)

## filter.saturation

**Availability**

Flash 8.

**Usage**

```
filter.saturation
```

**Description**

Property; a float value that specifies the saturation value of the filter. Acceptable values are from -100 to 100. This property is defined for Filter objects with a value of "adjustColorFilter" for the [filter.name](#) property.

**Example**

The following example sets the saturation value to -100 (grayscale) for the Adjust Color filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++){
    if(myFilters[i].name == 'adjustColorFilter'){
        myFilters[i].saturation = 0-100;
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

**See also**

[document.setFilterProperty\(\)](#)

## filter.shadowColor

### Availability

Flash 8.

### Usage

```
filter.shadowColor
```

### Description

Property; the color of the shadow, in one of the following formats:

- A string in the format "#RRGGBB" or "#RRGGBBAA"
- A hexadecimal number in the format 0xRRGGBB
- An integer that represents the decimal equivalent of a hexadecimal number

This property is defined for Filter objects with a value of "bevelFilter" for the [filter.name](#) property.

### Example

The following example sets the shadow color to "#ff00003e" for the Bevel filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++){
    if(myFilters[i].name == 'bevelFilter'){
        myFilters[i].shadowColor = '#ff00003e';
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

### See also

[document.setFilterProperty\(\)](#)

## filter.strength

### Availability

Flash 8.

### Usage

```
filter.strength
```

### Description

Property; an integer that specifies the percentage strength of the filter. Acceptable values are between 0 and 25,500.

This property is defined for Filter objects with a value of "bevelFilter", "dropShadowFilter", "glowFilter", "gradientGlowFilter", or "gradientBevelFilter" for the [filter.name](#) property.

### Example

The following example sets the strength to 50 for the Glow filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++){
    if(myFilters[i].name == 'glowFilter'){
        myFilters[i].strength = 50;
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

**See also**

[document.setFilterProperty\(\)](#)

## filter.type

**Availability**

Flash 8.

**Usage**

`filter.type`

**Description**

Property; a string that specifies the type of bevel or glow. Acceptable values are "inner", "outer", and "full". This property is defined for Filter objects with a value of "bevelFilter", "gradientGlowFilter", or "gradientBevelFilter" for the [filter.name](#) property.

**Example**

The following example sets the type to "full" for all Bevel filters on the selected object(s):

```
var myFilters = fl.getDocumentDOM().getFilters();
for(i=0; i < myFilters.length; i++){
    if(myFilters[i].name == 'bevelFilter'){
        myFilters[i].type = 'full';
    }
}
fl.getDocumentDOM().setFilters(myFilters);
```

**See also**

[document.setFilterProperty\(\)](#)

# Chapter 17: flash object (fl)

## fl summary

### Availability

Flash MX 2004.

### Description

The flash object represents the Flash application. You can use `flash` or `f1` to refer to this object. This documentation uses `f1` in code samples throughout.

### Method summary

The following methods can be used with the flash object:

Method	Description
<code>f1.addEventListener()</code>	Registers a function to be called when a specific event is received.
<code>f1.browseForFileURL()</code>	Opens a File Open or File Save system dialog box and lets the user specify a file to be opened or saved.
<code>f1.browseForFolderURL()</code>	Displays a Browse for Folder dialog box and lets the user select a folder.
<code>f1.clearPublishCache()</code>	Clears the publish cache.
<code>f1.clipCopyString()</code>	Copies the specified string to the Clipboard.
<code>f1.closeAll()</code>	Closes all open documents, displaying the Save As dialog box for any documents that were not previously saved.
<code>f1.closeAllPlayerDocuments()</code>	Closes all the SWF files that were opened with Control > Test Movie.
<code>f1.closeDocument()</code>	Closes the specified document.
<code>f1.copyLibraryItem()</code>	Silently copies a library item from a document without exposing the item in the Flash Pro user interface.
<code>f1.createDocument()</code>	Opens a new document and selects it.
<code>f1.exportPublishProfileString()</code>	Uniform Resource Identifier (URI) from which to export publish settings.
<code>f1.fileExists()</code>	Checks whether a file already exists on disk.
<code>f1.findDocumentDOM()</code>	Lets you target a specific file by using its unique identifier.
<code>f1.findDocumentIndex()</code>	Returns an array of integers that represent the position of a document in the <code>f1.documents</code> array.
<code>f1.findObjectInDocByName()</code>	Exposes elements with instance names that match specified text.
<code>f1.findObjectInDocByType()</code>	Exposes elements of a specified element type in a document.
<code>f1.getAppMemoryInfo()</code>	Returns an integer that represents the number of bytes being used in a specified area of Flash.exe memory.
<code>f1.getDocumentDOM()</code>	Retrieves the DOM ( <a href="#">Document object</a> ) of the currently active document.

Method	Description
<code>fl.getSwfPanel()</code>	Returns the SWFPanel object based on the panel's localized name or its SWF filename.
<code>fl.getThemeColor()</code>	Retrieves a theme color that matches the parameter specified theme parameter name.
<code>fl.getThemeColorParameters()</code>	Retrieves an array of theme parameter names.
<code>fl.getThemeFontInfo()</code>	Return either the font style or the font size that is used to draw the UI of the specified size.
<code>fl.isFontInstalled()</code>	Determines whether a specified font is installed.
<code>fl.mapPlayerURL()</code>	Maps an escaped Unicode URL to a UTF-8 or MBCS URL.
<code>fl.openDocument()</code>	Opens a Flash (FLA) document for editing in a new Flash Document window and gives it focus.
<code>fl.openScript()</code>	Opens a script (JSFL, AS, ASC) or other file (XML, TXT) in the Flash text editor.
<code>fl.quit()</code>	Quits Flash, prompting the user to save any changed documents.
<code>fl.reloadEffects() - dropped</code>	Dropped in Flash Professional CC.
<code>fl.reloadTools()</code>	Rebuilds the Tools panel from the toolconfig.xml file.
<code>fl.removeEvent Listener()</code>	Unregisters a function that was registered using <code>fl.addEvent Listener()</code> .
<code>fl.resetAS3PackagePaths()</code>	Resets the global Classpath setting in the ActionScript 3.0 Settings dialog box to the default value.
<code>fl.resetPackagePaths() - dropped</code>	Dropped in Flash Professional CC.
<code>fl.runScript()</code>	Executes a JavaScript file.
<code>fl.saveAll()</code>	Saves all open documents, displaying the Save As dialog box for any documents that were not previously saved.
<code>fl.saveDocument()</code>	Saves the specified document as a FLA document.
<code>fl.saveDocumentAs()</code>	Displays the Save As dialog box for the specified document.
<code>fl.selectElement()</code>	Enables selection or editing of an element.
<code>fl.selectTool()</code>	Selects the specified tool in the Tools panel.
<code>fl.setActiveWindow()</code>	Sets the active window to be the specified document.
<code>fl.setPrefBoolean()</code>	Sets a boolean preference value.
<code>fl.showIdleMessage()</code>	Lets you disable the warning about a script running too long.
<code>fl.toggleBreakpoint()</code>	Toggles a breakpoint for the given .as file at the given line.
<code>fl.trace()</code>	Sends a text string to the Output panel.
<code>fl.xmlPanel()</code>	Launches the XML To UI dialog from a URI that points to an XML-format file.
<code>fl.xmlPanelFromString()</code>	Launches the XML To UI dialog from an XML-format string.

### Property summary

The following properties can be used with the flash object.

Property	Description
<code>fl.actionsPanel</code>	Read-only; an <a href="#">actionsPanel object</a> .
<code>fl.as3PackagePaths</code>	A string that corresponds to the global Classpath setting in the ActionScript 3.0 Settings dialog box.
<code>fl.compilerErrors</code>	Read-only; a <a href="#">compilerErrors object</a> .
<code>fl.componentsPanel</code>	Read-only; a <a href="#">componentsPanel object</a> , which represents the Components panel.
<code>fl.configDirectory</code>	Read-only; a string that specifies the full path for the local user's Configuration folder as a platform-specific path.
<code>fl.configURI</code>	Read-only; a string that specifies the full path for the local user's Configuration directory as a file:/// URI.
<code>fl.contactSensitiveSelection</code>	A Boolean value that specifies whether Contact Sensitive selection mode is enabled.
<code>fl.createNewDocList</code>	Read-only; an array of strings that represent the various types of documents that can be created.
<code>fl.createNewDocListType</code>	Read-only; an array of strings that represent the file extensions of the types of documents that can be created.
<code>fl.createNewTemplateList</code>	Read-only; an array of strings that represent the various types of templates that can be created.
<code>fl.documents</code>	Read-only; an array of Document objects (see <a href="#">Document object</a> ) that represent the documents (FLA files) that are currently open for editing.
<code>fl.drawingLayer</code>	The drawingLayer object that an extensible tool should use when the user wants to temporarily draw while dragging.
<code>fl.externalLibraryPath</code>	A string that contains a list of items in the global ActionScript 3.0 External library path, which specifies the location of SWC files used as runtime shared libraries.
<code>fl.flexSDKPath</code>	A string that specifies the path to the Flex SDK folder, which contains bin, frameworks, lib, and other folders.
<code>fl.installedPlayers</code>	Returns an array of generic objects corresponding to the list of installed Flash Players in the document Property inspector.
<code>fl.languageCode</code>	Returns the five character code identifying the locale of the application's user interface.
<code>fl.libraryPath</code>	A string that contains a list of items in the global ActionScript 3.0 Library path, which specifies the location of SWC files or folders containing SWC files.
<code>fl.Math</code>	Read-only; the <a href="#">Math object</a> , which provides methods for matrix and point operations.
<code>fl.mruRecentFileList</code>	Read-only; an array of the complete filenames in the Most Recently Used (MRU) list that the Flash authoring tool manages.
<code>fl.mruRecentFileType</code>	Read-only; an array of the file types in the MRU list that the Flash authoring tool manages.
<code>fl.packagePaths - dropped</code>	Dropped in Flash Professional CC.
<code>fl.publishCacheDiskSizeMax</code>	An integer that sets the disk cache size limit preference.
<code>fl.publishCacheEnabled</code>	A boolean value that sets whether publish cache is enabled.
<code>fl.publishCacheMemoryEntrySizeLimit</code>	An integer property that sets the maximum size for the memory cache entry preference.

Property	Description
<code>fl.publishCacheMemorySizeMax</code>	An integer that sets the memory cache size limit preference.
<code>fl.objectDrawingMode</code>	An integer that represents the object drawing mode that is enabled.
<code>fl.outputPanel</code>	Read-only; reference to the <a href="#">outputPanel object</a> .
<code>fl.presetPanel</code>	Read-only; a <a href="#">presetPanel object</a> .
<code>fl.scriptURI</code>	Read-only; a string that represents the path of the currently running JSFL script, expressed as a file:/// URI.
<code>fl.sourcePath</code>	A string that contains a list of items in the global ActionScript 3.0 Source path, which specifies the location of ActionScript class files.
<code>fl.spriteSheetExporter</code>	Returns an instance of <a href="#">SpriteSheetExporter object</a> .
<code>fl.swfPanels</code>	An array of registered <a href="#">swfPanel objects</a> (see <a href="#">swfPanel object</a> ).
<code>fl.tools</code>	Read-only; an array of <a href="#">Tools objects</a> .
<code>fl.version</code>	Read-only; the long string version of the Flash authoring tool, including platform.
<code>fl.xmlui</code>	Read-only; an <a href="#">XMLUI object</a> .

## fl.actionsPanel

### Availability

Flash CS3 Professional.

### Usage

```
fl.actionsPanel
```

### Description

Read-only property; an actionsPanel object, which represents the currently displayed Actions panel. For information on using this property, see [actionsPanel object](#).

## fl.addEventListener()

### Availability

Flash CS3 Professional. The `prePublish`, `postPublish`, `selectionChanged`, and `dpiChanged` events are new in Flash Professional CC.

### Usage

```
fl.addEventListener(eventType, callbackFunction)
```

### Parameters

**eventType** A string that specifies the event type to pass to this callback function. Acceptable values are "documentNew", "documentOpened", "documentClosed", "mouseMove", "documentChanged", "layerChanged", "timelineChanged", "frameChanged", "prePublish", "postPublish", "selectionChanged", and `dpiChanged`.

The `documentChanged` value doesn't mean that the content of a document has changed; it means that a different document is now in the foreground. That is, `f1.getDocumentDOM()` will return a different value than it did before this event occurred.

**callbackFunction** The name of the function you want to execute every time the event occurs.

### Returns

An integer that identifies the event listener. Use this identifier when calling `f1.removeEventListener()`.

### Description

Method; registers a function to be called when a specific event occurs. Note that you can define multiple listeners for the same event.

When using this method, be aware that if the event occurs frequently (as might be the case with `mouseMove`) and the function takes a long time to run, your application might hang or otherwise enter an error state. Additionally, the `prePublish` and `postPublish` events should have minimal code and execute quickly.

### Example

The following example displays a message in the Output panel when a document is closed:

```
myFunction = function () {  
    f1.trace('document was closed');  
}  
var eventID = f1.addEventListener("documentClosed", myFunction);
```

### See also

[f1.removeEventListener\(\)](#)

## f1.as3PackagePaths

### Availability

Flash CS3 Professional.

### Usage

`f1.as3PackagePaths`

### Description

Property; a string that corresponds to the global Classpath setting in the ActionScript 3.0 Settings dialog box. Items in the string are delimited by semi-colons. To view or change ActionScript 2.0 Classpath settings, use `f1.packagePaths - dropped`.

### Example

The following example illustrates changing the ActionScript 3.0 Classpath settings.

```
f1.trace(f1.as3PackagePaths);  
// Output (assuming started with default value)  
// .;$ (AppConfig)/ActionScript 3.0/Classes  
f1.as3PackagePaths="buying;selling";  
f1.trace(f1.as3PackagePaths);  
// Output  
// buying; selling
```

**See also**[fl.resetAS3PackagePaths\(\)](#)

## fl/browseForFileURL()

**Availability**

Flash MX 2004.

**Usage**

```
fl.browseForFileURL([browseType [, title [, fileDescription [, fileFilter]]]])
```

**Parameters**

**browseType** A string that specifies the type of file browse operation. Valid values are "open", "select" or "save". The values "open" and "select" open the system File Open dialog box. Each value is provided for compatibility with Dreamweaver. The value "save" opens a system File Save dialog box.

**title** An optional string that specifies the title for the File Open or File Save dialog box. If this parameter is omitted, a default value is used. This parameter is optional.

**fileDescription** An optional string that specifies a file description, for example:

```
FLA Document (*.fla)  
ActionScript File (*.as)
```

**fileFilter** An optional string that specifies a filter, such that only files that match the filters are displayed in the dialog, for example:

```
"fla"  
"fla;as"  
"jsfl;fla;as"
```

**Returns**

The URL of the file, expressed as a file:/// URI; returns `null` if the user cancels out of the dialog box.

**Description**

Method; opens a File Open or File Save system dialog box and lets the user specify a file to be opened or saved.

**Example**

The following examples illustrate various options of the `fl.browseForFileURL()` method:

```
//CC
var uri = fl.browseForFileURL("open", "Select a FLA", "FLA Document (*.fla)", "fla");
or
var fileDescription = "FLA document (*.fla);Actionscript File (*.as)";
var fileFilter = "fla;as";
var uri = fl.browseForFileURL("open", "Select a FLA or AS file", fileDescription, fileFilter);

//The following are for CS4 through CS6. They do not work in CC.
var fileURL = fl.browseForFileURL("open", "Select file");
var doc = fl.openDocument(fileURL);

// The macFormat and winFormat parameters are supported in Flash CS4 through CS6.
// To enable only FLA files in the open file dialog
var macFormat = "Flash Document|SPA||";
var winFormat = "Flash Document|*.fla||";
var previewArea = {};
var uri = fl.browseForFileURL("open", "Select a FLA file", previewArea, macFormat, winFormat);

// To enable only AS files in the open file dialog
var macFormat = "ActionScript File|TEXT[*.as||";
var winFormat = "ActionScript File|*.as||";
var previewArea = {};
var uri = fl.browseForFileURL("open", "Select a FLA file", previewArea, macFormat, winFormat);

// To enable only FLA and AS files in the open file dialog
var macFormat = "Flash Document|SPA[* .fla|ActionScript File|TEXT[* .as||";
var winFormat = "Flash Document|*.fla|ActionScript File|*.as||";
var previewArea = {};
var uri = fl.browseForFileURL("open", "Select a FLA or AS file", previewArea, macFormat,
winFormat);
```

#### See also

[fl.browseForFolderURL\(\)](#)

## fl.browseForFolderURL()

#### Availability

Flash 8.

#### Usage

`fl.browseForFolderURL([description])`

#### Parameters

**description** An optional string that specifies the description of the Browse For Folder dialog box. If this parameter is omitted, the dialog box title is “Select Folder.”

#### Returns

The URL of the folder, expressed as a file:/// URI; returns `null` if the user cancels out of the dialog box.

**Description**

Method; displays a Browse for Folder dialog box and lets the user select a folder.

**Example**

The following example lets the user select a folder and then displays a list of files in that folder:

```
var folderURI = fl.browseForFolderURL("Select a folder.");
var folderContents = FLfile.listFolder(folderURI);
```

**See also**

[fl.browseForFileURL\(\)](#), [FLfile object](#)

## fl.clearPublishCache()

**Availability**

Flash CS5.5 Professional.

**Usage**

```
fl.clearPublishCache()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; empties the publish cache.

**Example**

The following code empties the publish cache:

```
fl.clearPublishCache()
```

**See also**

[fl.publishCacheDiskSizeMax](#), [fl.publishCacheEnabled](#), [fl.publishCacheMemoryEntrySizeLimit](#),  
[fl.publishCacheMemorySizeMax](#)

## fl.clipCopyString()

**Availability**

Flash CS3 Professional.

**Usage**

```
fl.clipCopyString(string)
```

**Parameters**

**string** A string to be copied to the Clipboard.

**Returns**

Nothing.

**Description**

Method; copies the specified string to the Clipboard.

To copy the current selection to the Clipboard, use `document.clipCopy()`.

**Example**

The following example copies the path of the current document to the Clipboard:

```
var documentPath = fl.getDocumentDOM().path;
fl.clipCopyString(documentPath);
```

## fl.closeAll()

**Availability**

Flash MX 2004.

**Usage**

```
fl.closeAll([bPromptToSave])
```

**Parameters**

**bPromptToSave** An optional Boolean value that specifies whether to display the Save dialog box for any files that have been changed since they were previously saved, or the Save As dialog box for files that have never been saved. The default value is `true`.

**Returns**

Nothing.

**Description**

Method; closes all open files (FLA files, SWF files, JSFL files, and so on). If you want to close all open files without saving changes to any of them, pass `false` for `bPromptToSave`. This method does not terminate the application.

**Example**

The following code closes all open files, prompting the user to save any new or changed files.

```
fl.closeAll();
```

**See also**

`fl.closeAllPlayerDocuments()`, `fl.closeDocument()`

## fl.closeAllPlayerDocuments()

### Availability

Flash CS3 Professional.

### Usage

```
fl.closeAllPlayerDocuments()
```

### Parameters

None.

### Returns

A Boolean value: `true` if one or more movie windows were open; `false` otherwise.

### Description

Method; closes all the SWF files that were opened with Control > Test Movie.

### Example

The following example closes all the SWF files that were opened with Control > Test Movie.

```
fl.closeAllPlayerDocuments();
```

### See also

[fl.closeAll\(\)](#), [fl.closeDocument\(\)](#)

## fl.closeDocument()

### Availability

Flash MX 2004.

### Usage

```
fl.closeDocument(documentObject [, bPromptToSaveChanges])
```

### Parameters

**documentObject** A [Document object](#). If *documentObject* refers to the active document, the Document window might not close until the script that calls this method finishes executing.

**bPromptToSaveChanges** A Boolean value. When *bPromptToSaveChanges* is `false`, the user is not prompted if the document contains unsaved changes; that is, the file is closed and the changes are discarded. If *bPromptToSaveChanges* is `true`, and if the document contains unsaved changes, the user is prompted with the standard yes-or-no dialog box. The default value is `true`. This parameter is optional.

### Returns

Nothing.

**Description**

Method; closes the specified document.

**Example**

The following example illustrates two ways of closing a document.

```
// Closes the specified document and prompts to save changes.  
fl.closeDocument(fl.documents[0]);  
fl.closeDocument(fl.documents[0], true); // Use of true is optional.  
// Closes the specified document without prompting to save changes.  
fl.closeDocument(fl.documents[0], false);
```

**See also**

[fl.closeAll\(\)](#)

## fl.compilerErrors

**Availability**

Flash CS3 Professional.

**Usage**

`fl.compilerErrors`

**Description**

Read-only property; a compilerErrors object, which represents the Errors panel. For information on using this property, see [compilerErrors object](#).

## fl.componentsPanel

**Availability**

Flash MX 2004.

**Usage**

`fl.componentsPanel`

**Description**

Read-only property; a [componentsPanel object](#), which represents the Components panel.

**Example**

The following example stores a componentsPanel object in the `comPanel` variable:

```
var comPanel = fl.componentsPanel;
```

## fl.configDirectory

### Availability

Flash MX 2004.

### Usage

```
fl.configDirectory
```

### Description

Read-only property; a string that specifies the full path for the local user's Configuration directory in a platform-specific format. To specify this path as a file:/// URI, which is not platform-specific, use [fl.configURI](#).

### Example

The following example displays the Configuration directory in the Output panel:

```
fl.trace("My local configuration directory is " + fl.configDirectory);
```

## fl.configURI

### Availability

Flash MX 2004.

### Usage

```
fl.configURI
```

### Description

Read-only property; a string that specifies the full path for the local user's Configuration directory as a file:/// URI. See also [fl.configDirectory](#).

### Example

The following example runs a specified script. Using `fl.configURI` lets you specify the location of the script without knowing which platform the script is running on.

```
// To run a command in your commands menu, change "Test.jsfl"
// to the command you want to run in the line below.
fl.runScript( fl.configURI + "Commands/Test.jsfl" );
```

## fl.contactSensitiveSelection

### Availability

Flash 8.

### Usage

```
fl.contactSensitiveSelection
```

**Description**

A Boolean value that specifies whether Contact Sensitive selection mode is enabled (`true`) or not (`false`).

**Example**

The following example shows how to disable Contact Sensitive selection mode before making a selection and then how to reset it to its original value after making the selection:

```
var contact = fl.contactSensitiveSelection;
fl.contactSensitiveSelection = false;
// Insert selection code here.
fl.contactSensitiveSelection = contact;
```

## **fl.copyLibraryItem()**

**Availability**

Flash Professional CS5.

**Usage**

```
fl.copyLibraryItem(fileURI, libraryItemPath)
```

**Parameters**

**fileURI** A string, expressed as a file:/// URI, that contains the path to the FLA or XFL file.

**libraryItemPath** A string, that specifies the path to the library item to be copied.

**Returns**

A Boolean value: `true` if the copy succeeds; `false` otherwise.

**Description**

Method; silently copies a library item from a document without exposing the item in the Flash Pro user interface. Call the `document.clipPaste()` method to paste the item into the new document.

**Example**

The following example illustrates use of the `fl.copyLibraryItem()` method to copy the `armjoint-11` library item.:

```
fl.copyLibraryItem("file:///c:/users/userid/Desktop/Robot.fla", "armjoint-11");
fl.getDocumentDOM().clipPaste(true);
```

## **fl.createDocument()**

**Availability**

Flash MX 2004.

**Usage**

```
fl.createDocument([docType])
```

**Parameters**

**docType** A string that specifies the type of document to create. The only acceptable value is "timeline". The default value is "timeline", which has the same effect as choosing File > New > Flash File (ActionScript 3.0). This parameter is optional.

**Returns**

The Document object for the newly created document, if the method is successful. If an error occurs, the value is undefined.

**Description**

Method; opens a new document and selects it. Values for size, resolution, and color are the same as the current defaults.

**Example**

The following example creates two timeline-based documents:

```
// Create two Timeline-based Flash documents.  
fl.createDocument();  
fl.createDocument("timeline");
```

## fl.createNewDocList

**Availability**

Flash MX 2004.

**Usage**

```
fl.createNewDocList
```

**Description**

Read-only property; an array of strings that represent the various types of documents that can be created.

**Example**

The following example displays the types of documents that can be created, in the Output panel:

```
fl.trace("Number of choices " + fl.createNewDocList.length);  
for (i = 0; i < fl.createNewDocList.length; i++)  
    fl.trace("choice: " + fl.createNewDocList[i]);
```

## fl.createNewDocListType

**Availability**

Flash MX 2004.

**Usage**

```
fl.createNewDocListType
```

**Description**

Read-only property; an array of strings that represent the file extensions of the types of documents that can be created. The entries in the array correspond directly (by index) to the entries in the `fl.createNewDocList` array.

**Example**

The following example displays the extensions of the types of documents that can be created, in the Output panel:

```
fl.trace("Number of types " + fl.createNewDocListType.length);
for (i = 0; i < fl.createNewDocListType.length; i++) fl.trace("type: " +
fl.createNewDocListType[i]);
```

## fl.createNewTemplateList

**Availability**

Flash MX 2004.

**Usage**

```
fl.createNewTemplateList
```

**Description**

Read-only property; an array of strings that represent the various types of templates that can be created.

**Example**

The following example displays the types of templates that can be created, in the Output panel:

```
fl.trace("Number of template types: " + fl.createNewTemplateList.length); for (i = 0; i <
fl.createNewTemplateList.length; i++) fl.trace("type: " + fl.createNewTemplateList[i]);
```

## fl.documents

**Availability**

Flash MX 2004.

**Usage**

```
fl.documents
```

**Description**

Read-only property; an array of Document objects (see [Document object](#)) that represent the documents (FLA files) that are currently open for editing.

**Example**

The following example stores an array of open documents in the `docs` variable:

```
var docs = fl.documents;
```

The following example displays the names of currently open documents, in the Output panel:

```
for (doc in fl.documents) {  
    fl.trace(fl.documents[doc].name);  
}
```

## fl.drawingLayer

### Availability

Flash MX 2004.

### Usage

```
fl.drawingLayer
```

### Description

Read-only property; the [drawingLayer object](#) that an extensible tool should use when the user wants to temporarily draw while dragging (for example, when creating a selection marquee).

### Example

See [drawingLayer.setColor\(\)](#).

## fl.exportPublishProfileString()

### Availability

Flash Professional CS5.

### Usage

```
fl.exportPublishProfileString( ucfURI [, profileName] )
```

### Parameters

**ucfURI** A string that specifies the file Uniform Resource Identifier (URI) from which to export the publish settings.

**profileName** A string that specifies the profile name to export. This parameter is optional.

### Returns

String.

### Description

Returns a specific document's publishing profile without having to open the file. The publish profile can also be specified, but this is optional.

### Example

The following example reads the publishing profile string:

```
var ppXML = "";
var ucfURI = fl.browseForFileURL("open", "select a FLA");
if (ucfURI && ucfURI.length > 0)
ppXML = fl.exportPublishProfileString(ucfURI);
fl.trace(ppXML);
```

## fl.externalLibraryPath

### Availability

Flash CS4 Professional.

### Usage

```
fl.externalLibraryPath
```

### Description

Property; a string that contains a list of items in the global ActionScript 3.0 External library path, which specifies the location of SWC files used as runtime shared libraries. Items in the string are delimited by semi-colons. In the authoring tool, the items are specified by choosing Edit > Preferences > ActionScript > ActionScript 3.0 Settings.

### Example

The following example adds the /SWC\_runtime folder to the global ActionScript 3.0 External library path:

```
fl.trace(fl.externalLibraryPath);
fl.externalLibraryPath = "/SWC_runtime;" + fl.externalLibraryPath;
fl.trace(fl.externalLibraryPath);
```

### See also

[fl.flexSDKPath](#), [fl.libraryPath](#), [fl.sourcePath](#), [document.externalLibraryPath](#)

## fl.fileExists()

### Availability

Flash MX 2004.

### Usage

```
fl.fileExists(fileURI)
```

### Parameters

**fileURI** A string, expressed as a file:/// URI, that contains the path to the file.

### Returns

A Boolean value: `true` if the file exists on disk; `false` otherwise.

### Description

Method; checks whether a file already exists on disk.

**Example**

The following example displays `true` or `false` in the Output panel for each specified file, depending on whether the file exists.

```
alert(f1.fileExists("file:///C|/example.fla"));
alert(f1.fileExists("file:///C|/example.jsfl"));
alert(f1.fileExists "");
```

## **fl.findDocumentDOM()**

**Availability**

Flash CS3 Professional.

**Usage**

```
fl.findDocumentDOM(id)
```

**Parameters**

`id` An integer that represents a unique identifier for a document.

**Returns**

A Document object, or `null` if no document exists with the specified `id`.

**Description**

Method; lets you target a specific file by using its unique identifier (instead of its index value, for example). Use this method in conjunction with `document.id`.

**Example**

The following example illustrates reading a document's ID and then using it to target that document:

```
var originalDocID = fl.getDocumentDOM().id;
// other code here, maybe working in different files
var targetDoc = fl.findDocumentDOM(originalDocID);
// Set the height of the Stage in the original document to 400 pixels.
targetDoc.height = 400;
```

**See also**

[fl.findDocumentIndex\(\)](#)

## **fl.findDocumentIndex()**

**Availability**

Flash MX 2004.

**Usage**

```
fl.findDocumentIndex(name)
```

**Parameters**

**name** The document name for which you want to find the index. The document must be open.

**Returns**

An array of integers that represent the position of the document *name* in the `fl.documents` array.

**Description**

Method; returns an array of integers that represent the position of the document *name* in the `fl.documents` array. More than one document with the same name can be open (if the documents are located in different folders).

**Example**

The following example displays information about the index position of any open files named test.fla in the Output panel:

```
var filename = "test.fla"
var docIndex = fl.findDocumentIndex(filename);
for (var index in docIndex)
    fl.trace(filename + " is open at index " + docIndex[index]);
```

**See also**

[fl.documents](#), [fl.findDocumentDOM\(\)](#)

## fl.findObjectInDocByName()

**Availability**

Flash CS3 Professional.

**Usage**

```
fl.findObjectInDocByName(instanceName, document)
```

**Parameters**

**instanceName** A string that specifies the instance name of an item in the specified document.

**document** The [Document object](#) in which to search for the specified item.

**Returns**

An array of generic objects. Use the `.obj` property of each item in the array to get the object. The object has the following properties: `keyframe`, `layer`, `timeline`, and `parent`. You can use these properties to access the hierarchy of the object. For more information on these properties and how to access them, see [fl.findObjectInDocByType\(\)](#).

You can also access methods and properties for the `layer` and `timeline` values; they are equivalent to the [Layer object](#) and the [Timeline object](#), respectively.

**Description**

Method; exposes elements in a document with instance names that match the specified text.

**Note:** In some cases, this method works only when run as a command from within a FLA file, not when you are currently viewing or editing the JSFL file.

**Example**

The following example searches the current document for elements named "instance01".

```
var nameToSearchFor = "instance01";
var doc = fl.getDocumentDOM();
var results = fl.findObjectInDocByName(nameToSearchFor, doc);
if (results.length > 0) {
    alert("success, found " + results.length + " objects");
}
else {
    alert("failed, no objects named " + nameToSearchFor + " found");
}
```

**See also**

[fl.findObjectInDocByType\(\)](#)

## fl.findObjectInDocByType()

**Availability**

Flash CS3 Professional.

**Usage**

```
fl.findObjectInDocByType(elementType, document)
```

**Parameters**

**elementType** A string that represents the type of element to search for. For acceptable values, see [element.elementType](#).

**document** The [Document object](#) in which to search for the specified item.

**Returns**

An array of generic objects. Use the `.obj` property of each item in the array to get the element object. Each object has the following properties: `keyframe`, `layer`, `timeline`, and `parent`. You can use these properties to access the hierarchy of the object.

You can also access methods and properties for the `layer` and `timeline` values; they are equivalent to the [Layer object](#) and the [Timeline object](#), respectively.

The second and third examples in the Examples section show how to access these properties.

**Description**

Method; exposes elements of a specified element type in a document.

**Note:** In some cases, this method works only when run as a command from within a FLA file, not when you are currently viewing or editing the JSFL file.

**Example**

The following example searches the current document for text fields and then changes their contents:

```
var doc = fl.getDocumentDOM();
var typeToSearchFor = "text";
var results = fl.findObjectInDocByType(typeToSearchFor, doc);
if (results.length > 0) {
    for (var i = 0; i < results.length; i++) {
        results[i].obj.setTextString("new text");
    }
    alert("success, found " + results.length + " objects");
}
else {
    alert("failed, no objects of type " + typeToSearchFor + " found");
}
```

The following example shows how to access the special properties of the object returned by this method:

```
var doc = fl.getDocumentDOM();
var resultsArray = findObjectInDocByType("text", doc);
if (resultsArray.length > 0)
{
    var firstItem = resultsArray[0];

    // firstItem.obj- This is the element object that was found.

    // You can access the following properties of this object:
    // firstItem.keyframe- The keyframe that the element is on.
    // firstItem.layer- The layer that the keyframe is on.
    // firstItem.timeline- The timeline that the layer is on.
    // firstItem.parent- The parent of the timeline. For example,
    //       the timeline might be in a symbol instance.
}
```

The following example shows how to back up the DOM to find the name of a layer on which a text field was found, using the resultArray.obj object:

```
var doc = fl.getDocumentDOM();
var typeToSearchFor = "text";
var resultsArray = fl.findObjectInDocByType(typeToSearchFor, doc);
if (resultsArray.length > 0) {
    for (var i = 0; i < resultsArray.length; i++) {
        resultsArray[i].obj.setTextString("new text");
        var firstItem = resultsArray[0];
        firstItemObj = firstItem.obj;
        fl.trace(firstItemObj.layer.name+"layerName");
    }
} else {
    alert("failed, no objects of type " + typeToSearchFor + " found");
}
```

## See also

[fl.findObjectInDocByName\(\)](#)

## fl.flexSDKPath

### Availability

Flash CS4 Professional.

### Usage

```
fl.flexSDKPath
```

### Description

Property; a string that specifies the path to the Flex SDK folder, which contains bin, frameworks, lib, and other folders. In the authoring tool, the items are specified by choosing Edit > Preferences > ActionScript > ActionScript 3.0 Settings.

### Example

The following code displays the Flex SDK path in the Output panel:

```
fl.trace(fl.flexSDKPath);
```

### See also

[fl.externalLibraryPath](#), [fl.libraryPath](#), [fl.sourcePath](#)

## fl.getAppMemoryInfo()

### Availability

Flash 8 (Windows only).

### Usage

```
fl.getAppMemoryInfo(memType)
```

### Parameters

**memType** An integer that specifies the memory utilization area to be queried. For a list of acceptable values, see the following description.

### Returns

An integer that represents the number of bytes being used in a specified area of Flash.exe memory.

### Description

Method (Windows only); returns an integer that represents the number of bytes being used in a specified area of Flash.exe memory. Use the following table to determine which value you want to pass as *memType*:

memType	Resource data
0	PAGEFAULTCOUNT
1	PEAKWORKINGSETSIZE
2	WORKINGSETSIZE

memType	Resource data
3	QUOTAPEAKPAGEDPOOLUSAGE
4	QUOTAPAGEDPOOLUSAGE
5	QUOTAPEAKNONPAGEDPOOLUSAGE
6	QUOTANONPAGEDPOOLUSAGE
7	PAGEFILEUSAGE
8	PEAKPAGEFILEUSAGE

**Example**

The following example displays the current working memory consumption:

```
var memsize = fl.getAppMemoryInfo(2);
fl.trace("Flash current memory consumption is " + memsize + " bytes or " + memsize/1024 + " KB");
```

## **fl.getDocumentDOM()**

**Availability**

Flash MX 2004.

**Usage**

```
fl.getDocumentDOM()
```

**Parameters**

None.

**Returns**

A Document object, or null if no documents are open.

**Description**

Method; retrieves the DOM ([Document object](#)) of the currently active document (FLA file). If one or more documents are open but a document does not currently have focus (for example, if a JSFL file has focus), retrieves the DOM of the most recently active document.

**Example**

The following example displays the name of the current or most recently active document in the Output panel:

```
var currentDoc = fl.getDocumentDOM();
fl.trace(currentDoc.name);
```

## **fl.getThemeColor()**

**Availability**

Flash Professional CC.

**Usage**

```
fl.getThemeColor(themeParamName)
```

**Parameters**

**themeParamName** A string that contains a theme parameter from the list returned by the `fl.getThemeColorParameters()` method. If the theme parameter is `themeUseGradients`, this method returns either "true" or "false".

**Returns**

A String containing a theme color (in #rrggbba or #rrggbbaa format) that matches the passed parameter. If the theme parameter is `themeUseGradients`, this method returns either "true" or "false".

**Description**

Method; returns the theme color that matches the passed theme parameter. Flash Professional CC introduced 2 UI themes: Dark and Light UI, and this method retrieves the current theme color to help you render your custom content.

**Example**

The following example returns the theme colors that corresponds to `themeAppBackgroundColor` and `themeStaticTextNormalColor`:

```
var colorValue = fl.getThemeColor("themeAppBackgroundColor");
fl.trace("app background color " + colorValue);
var staticColor = fl.getThemeColor("themeStaticTextNormalColor");
fl.trace("staticColor " + staticColor);
```

## **fl.getThemeColorParameters()**

**Availability**

Flash Professional CC.

**Usage**

```
fl.getThemeColorParameters()
```

**Parameters**

None.

**Returns**

An Array of strings that contain the theme color parameters.

**Description**

Method; returns an Array of strings that contain the theme color parameters. The available theme color parameters are as follows:

- `themeAppBackgroundColor`
- `themeItemSelectedColor`
- `themeItemHighlightedColor`
- `themeHotTextNormalColor`

- themeHotTextRolloverColor
- themeHotTextDisableColor
- themeStaticTextNormalColor
- themeStaticTextDisableColor
- themeTextEditNormalBackgroundColor
- themeTextEditDisableBackgroundColor
- themeUseGradients
- themeEnableShading
- themeDividerLine
- themeDividerLineBevel
- themeControlFocus
- themeControlBorderNormal
- themeControlBorderDisabled
- themeControlFillNormal
- themeControlFillTopNormal
- themeControlFillBottomNormal
- themeControlFillOver
- themeControlFillTopOver
- themeControlFillBottomOver
- themeControlFillDown
- themeControlFillTopDown
- themeControlFillBottomDown
- themeControlFillDisabled
- themeControlFillTopDisabled
- themeControlFillBottomDisabled
- themeControlFillSelectedOver
- themeControlFillTopSelectedOver
- themeControlFillBottomSelectedOver
- themeGenericIconNormal
- themeGenericIconShadowNormal
- themeGenericIconDisabled
- themeGenericIconShadowDisabled

Flash Professional CC introduced 2 UI themes: Dark and Light UI, and this method retrieves the current theme color parameters to help you render your custom content.

**Note:** The "top" and "bottom" colors are for drawing controls when shading is enabled. When shading is disabled, use `themeControlFillNormal`, `themeControlFillOver` without the "top" and "bottom" in the names

**Example**

The following example lists the theme color parameters:

```
var params = fl.getThemeColorParameters();
for(x = 0; x < params.length; x++) {
    fl.trace("params: " + params[x]);
}
```

## **fl.getThemeFontInfo()**

**Availability**

Flash Professional CC.

**Usage**

```
fl.getThemeFontInfo(infoType, size)
```

**Parameters**

**infoType** A string that contains one of the following:

- **fontStyle** - Return the font style for the size specified by the *size* parameter.
- **fontSize** - Return the font size for the size specified by the *size* parameter.

**size** A string that specifies either "large" or "small".

**Returns**

A String containing either the font style or the font size for the specific size.

**Description**

Method; returns either the font Style or the font Size that is used to draw the UI of the specified size.

**Example**

The following example illustrates the use of this method:

```
fl.getThemeFontInfo('fontStyle', 'large'); // Return the fontStyle for large size
fl.getThemeFontInfo('fontStyle', 'small'); // Return the fontStyle for small size
fl.getThemeFontInfo('fontSize', 'large'); // Return the fontSize for large size
fl.getThemeFontInfo('fontSize', 'small'); // Return the fontSize for small size
```

## **fl.getSwfPanel()**

**Availability**

Flash CS5.5 Professional.

**Usage**

```
fl.getSwfPanel(panelName, [useLocalizedPanelName])
```

**Parameters**

**panelName** The localized panel name or the root filename of the panel's SWF file. Pass in false as the second parameter if using the latter.

**useLocalizedPanelName** Optional. Defaults to true. If false, the panelName parameter is assumed to be the English (unlocalized) name of the panel, which corresponds to the SWF filename without the file extension.

**Returns**

SWFPanel object.

**Description**

Method; returns the SWFPanel object based on the panel's localized name or its SWF filename (without the filename extension).

**Example**

The following example displays the name of the panel referenced as 'Project' in the Output panel:

```
fl.trace('name of panel is: ' + fl.getSwfPanel('Project').name);
```

## fl.installedPlayers

**Availability**

Flash CS5.5 Professional.

**Usage**

```
fl.installedPlayers()
```

**Parameters**

None.

**Returns**

An array of generic objects corresponding to the list of installed Flash Players in the document PI.

**Description**

Read-only property; The array of generic objects corresponding to the list of installed Flash Players in the document PI.

Each object in the array contains the following properties:

**name** The string name of the document.

**version** Can be used to set the current player for a document, using the `Document.setPlayerVersion()` function.

**minASVersion** The minimum ActionScript version required by the document. An integer between the minASVersion and the maxASVersion (inclusive) can be used to set the ActionScript version of the document, using the `Document.asVersion` property.

**maxASVersion** The maximum ActionScript version supported by the document.

**stageWidth** The default Stage width in pixels for the given target. For example, for iPhone the default size is 320 x 480 pixels. For Android, the default size is 480 x 800.

**stageHeight** The default Stage height in pixels for the given target. For example, for iPhone the default size is 320 x 480 pixels. For Android, the default size is 480 x 800.

### Example

The following example traces the properties of all objects in the `installedPlayers` array to the output window:

```
var arr = fl.installedPlayers;
for (var i in arr) fl.trace("name: " + arr[i].name + " version: " + arr[i].version + "
minASVersion: " + arr[i].minASVersion + " maxASVersion: " + arr[i].maxASVersion + " stageWidth:
" + arr[i].stageWidth + " stageHeight: " + arr[i].stageHeight + " ");
```

## fl.isFontInstalled()

### Availability

Flash CS4 Professional.

### Usage

```
fl.isFontInstalled(fontName)
```

### Parameters

**fontName** A string that specifies the name of a device font.

### Returns

A Boolean value of `true` if the specified font is installed; `false` otherwise.

### Description

Method; determines whether a specified font is installed.

### Example

The following code displays “true” in the Output panel if the Times font is installed.

```
fl.trace(fl.isFontInstalled("Times"));
```

## fl.languageCode

### Availability

Flash CS5 Professional.

### Usage

```
fl.languageCode
```

### Description

Property; a string that returns the five character code identifying the locale of the application’s user interface.

**Example**

The following example returns the five character language code indicated by the Flash application's localized user interface:

```
locConfigURI = fl.applicationURI + fl.languageCode + "/Configuration";
```

## fl.libraryPath

**Availability**

Flash CS4 Professional.

**Usage**

```
fl.libraryPath
```

**Description**

Property; a string that contains a list of items in the global ActionScript 3.0 Library path, which specifies the location of SWC files or folders containing SWC files. Items in the string are delimited by semi-colons. In the authoring tool, the items are specified by choosing Edit > Preferences > ActionScript > ActionScript 3.0 Settings.

**Example**

The following example adds the /SWC folder to the global ActionScript 3.0 Library path:

```
fl.trace(fl.libraryPath);
fl.libraryPath = "/SWC;" + fl.libraryPath;
fl.trace(fl.libraryPath);
```

**See also**

[fl.externalLibraryPath](#), [fl.flexSDKPath](#), [fl.sourcePath](#), [document.libraryPath](#)

## fl.mapPlayerURL()

**Availability**

Flash MX 2004.

**Usage**

```
fl.mapPlayerURL(URI [, returnMBCS])
```

**Parameters**

**URI** A string that contains the escaped Unicode URL to map.

**returnMBCS** A Boolean value that you must set to `true` if you want an escaped MBCS path returned. Otherwise, the method returns UTF-8. The default value is `false`. This parameter is optional.

**Returns**

A string that is the converted URL.

**Description**

Method; maps an escaped Unicode URL to a UTF-8 or MBCS URL. Use this method when the string will be used in ActionScript to access an external resource. You must use this method if you need to handle multibyte characters.

**Example**

The following example converts a URL to UTF-8 so the player can load it:

```
var url = MMExecute( "fl.mapPlayerURL(" + myURL + ", false);" );
mc.loadMovie( url );
```

## fl.Math

**Availability**

Flash MX 2004.

**Usage**

```
fl.Math
```

**Description**

Read-only property; the [Math object](#) provides methods for matrix and point operations.

**Example**

The following example shows the transformation matrix of the selected object and its inverse:

```
// Select an element on the Stage and then run this script.
var mat = fl.getDocumentDOM().selection[0].matrix;
for(var prop in mat) {
    fl.trace("mat."+prop+" = " + mat[prop]);
}
var invMat = fl.Math.inverseMatrix( mat );
for(var prop in invMat) {
    fl.trace("invMat."+prop+" = " + invMat[prop]);
}
```

## fl.mruRecent fileList

**Availability**

Flash MX 2004.

**Usage**

```
fl.mruRecent fileList
```

**Description**

Read-only property; an array of the complete filenames in the Most Recently Used (MRU) list that the Flash authoring tool manages.

**Example**

The following example displays the number of recently opened files and the name of each file, in the Output panel:

```
fl.trace("Number of recently opened files: " + fl.mruRecentFileList.length);
for (i = 0; i < fl.mruRecentFileList.length; i++) fl.trace("file: " + fl.mruRecentFileList[i]);
```

## **fl.mruRecentFileType**

**Availability**

Flash MX 2004.

**Usage**

```
fl.mruRecentFileType
```

**Description**

Read-only property; an array of the file types in the MRU list that the Flash authoring tool manages. This array corresponds to the array in the [fl.mruRecentFileList](#) property.

**Example**

The following example displays the number of recently opened files and the type of each file, in the Output panel:

```
fl.trace("Number of recently opened files: " + fl.mruRecentFileType.length);
for (i = 0; i < fl.mruRecentFileType.length; i++) fl.trace("type: " +
fl.mruRecentFileType[i]);
```

## **fl.objectDrawingMode**

**Availability**

Flash 8.

**Usage**

```
fl.objectDrawingMode
```

**Description**

Property; a Boolean value that specifies whether the object drawing mode is enabled (`true`) or the merge drawing mode is enabled (`false`).

**Example**

The following example toggles the state of the object drawing mode:

```
var toggleMode = fl.objectDrawingMode;
if (toggleMode) {
    fl.objectDrawingMode = false;
} else {
    fl.objectDrawingMode = true;
}
```

## fl.openDocument()

### Availability

Flash MX 2004.

### Usage

```
fl.openDocument(fileURI)
```

### Parameters

**fileURI** A string, expressed as a file:/// URI, that specifies the name of the file to be opened.

### Returns

The [Document object](#) for the newly opened document, if the method is successful. If the file is not found or is not a valid FLA file, an error is reported and the script is cancelled.

### Description

Method; opens a Flash document (FLA file) for editing in a new Flash Document window and gives it focus. For a user, the effect is the same as selecting File > Open and then selecting a file. If the specified file is already open, the window that contains the document comes to the front. The window that contains the specified file becomes the currently selected document.

### Example

The following example opens a file named Document.fla that is stored in the root directory on the C drive. The code stores a Document object representing that document in the doc variable and sets the document to be the currently selected document. That is, until focus is changed, fl.getDocumentDOM() refers to this document.

```
var doc = fl.openDocument("file:///c|/Document.fla");
```

## fl.openScript()

### Availability

Flash MX 2004.

### Usage

```
fl.openScript(fileURI )
```

### Parameters

**fileURI** A string, expressed as a file:/// URI, that specifies the path of the JSFL, AS, ASC, XML, TXT, or other file that should be loaded into Flash.

### Returns

Nothing.

### Description

Method; opens an existing file or creates a new script (JSFL, AS, ASC) or other file (XML, TXT) in Flash.

**Example**

The following example opens a file named my\_test.jsfl that is stored in the /temp directory on the C drive:

```
fl.openScript("file:///c|/temp/my_test.jsfl");
```

## fl.outputPanel

**Availability**

Flash MX 2004.

**Usage**

```
fl.outputPanel
```

**Description**

Read-only property; reference to the [outputPanel object](#).

**Example**

See [outputPanel object](#).

## fl.packagePaths - dropped

**Availability**

Flash CS3 Professional. *Dropped in Flash Professional CC.*

**Usage**

```
fl.packagePaths
```

**Description**

*Dropped in Flash Professional CC.*

Property; a string that corresponds to the global Classpath setting in the ActionScript 2.0 Settings dialog box. Class paths within the string are delimited with semi-colons (;). To view or change ActionScript 3.0 Classpath settings, use [fl.as3PackagePaths](#).

**Example**

The following example illustrates changing the ActionScript 2.0 Classpath settings:

```
fl.trace(fl.packagePaths);
// Output (assuming started with default value)
// .;$(LocalData)/Classes
fl.packagePaths="buying;selling";
fl.trace(fl.packagePaths);
// Output
// buying; selling
```

**See also**

[fl.resetPackagePaths\(\) - dropped](#)

## fl.presetPanel

**Availability**

Flash CS4 Professional.

**Usage**

```
fl.presetPanel
```

**Description**

Read-only property: a [presetPanel](#) object.

## fl.publishCacheDiskSizeMax

**Availability**

Flash CS5.5 Professional.

**Usage**

```
fl.publishCacheDiskSizeMax
```

**Description**

Property: an integer that sets the maximum size, in megabytes, of the publish cache on disk.

**Example**

The following code sets the maximum publish cache size on disk to 1 megabyte:

```
fl.publishCacheDiskSizeMax = 1
```

**See also**

[fl.clearPublishCache\(\)](#), [fl.publishCacheEnabled](#), [fl.publishCacheMemoryEntrySizeLimit](#),  
[fl.publishCacheMemorySizeMax](#)

## fl.publishCacheEnabled

**Availability**

Flash CS5.5 Professional.

**Usage**

```
fl.publishCacheEnabled
```

**Description**

Property: a boolean value that sets whether the publish cache is enabled.

**Example**

The following code displays whether the published cache is enabled in the Output window.

```
fl.trace(f1.publishCacheEnabled);
```

**See also**

[fl.publishCacheDiskSizeMax](#), [fl.clearPublishCache\(\)](#), [fl.publishCacheMemoryEntrySizeLimit](#),  
[fl.publishCacheMemorySizeMax](#)

## fl.publishCacheMemoryEntrySizeLimit

**Availability**

Flash CS5.5 Professional.

**Usage**

```
fl.publishCacheMemoryEntrySizeLimit
```

**Description**

Property: an integer that sets the maximum size, in kilobytes, of entries that can be added to the publish cache in memory. Anything at or below this size will be kept in memory; anything larger will be written to disk.

Users with a lot of memory might want to raise this value to increase performance, while a user with very little memory might want to make it lower to keep the publish cache from consuming too much memory.

**Example**

The following code sets the maximum publish cache entry size that can be stored in memory to 100 kilobytes:

```
fl.publishCacheMemoryEntrySizeLimit = 100
```

**See also**

[fl.publishCacheDiskSizeMax](#), [fl.publishCacheEnabled](#), [fl.clearPublishCache\(\)](#),  
[fl.publishCacheMemorySizeMax](#)

## fl.publishCacheMemorySizeMax

**Availability**

Flash CS5.5 Professional.

**Usage**

```
fl.publishCacheMemorySizeMax
```

**Description**

Property: an integer that sets the maximum size, in megabytes, of the publish cache in memory.

**Example**

The following code sets the maximum publish cache size in memory to 1 megabyte:

```
fl.publishCacheMemorySizeMax = 1
```

**See also**

`fl.publishCacheDiskSizeMax`, `fl.publishCacheEnabled`, `fl.publishCacheMemoryEntrySizeLimit`,  
`fl.clearPublishCache()`

## fl.publishDocument()

**Availability**

Flash CS5 Professional.

**Usage**

```
fl.publishDocument( flaURI [, publishProfile] )
```

**Parameters**

**flaURI** A string, expressed as a file:/// URI, that specifies the path of the FLA file that should be silently published.

**publishProfile** A string that specifies the publish profile to use when publishing. If this parameter is omitted, the default publish profile is used.

**Returns**

Boolean

**Description**

Method; publishes a FLA file without opening it. This API opens the FLA in a headless mode and publishes the SWF (or whatever the profile is set to). The second parameter (publishProfile) is optional. The return value is a boolean indicating if the profile was found or not. In the case where the second parameter is not supplied, the return value is always true.

**Example**

The following example prompts the user to select a FLA file and silently publishes it using the “Default” publish profile:

```
var uri = fl.browseForFileURL("select", "select a FLA file to publish");
var publishProfileName = "Default";
fl.publishDocument(uri, publishProfileName);
```

## fl.quit()

**Availability**

Flash MX 2004.

**Usage**

```
fl.quit( [bPromptIfNeeded] )
```

**Parameters**

**bPromptIfNeeded** A Boolean value that is `true` (default) if you want the user to be prompted to save any modified documents. Set this parameter to `false` if you do not want the user to be prompted to save modified documents. In

the latter case, any modifications in open documents will be discarded and the application will exit immediately. Although it is useful for batch processing, use this method with caution. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; quits Flash, prompting the user to save any changed documents.

**Example**

The following example illustrates quitting with and without asking to save modified documents:

```
// Quit with prompt to save any modified documents.  
fl.quit();  
fl.quit(true); // True is optional.  
// Quit without saving any files.  
fl.quit(false);
```

## fl.reloadEffects() - dropped

**Availability**

Flash MX 2004. *Dropped in Flash Professional CC.*

**Usage**

```
fl.reloadEffects()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

*Dropped in Flash Professional CC.*

Method; reloads all effects descriptors defined in the user's Configuration Effects folder. This permits you to rapidly change the scripts during development, and it provides a mechanism to improve the effects without relaunching the application. This method works best if used in a command placed in the Commands folder.

**Example**

The following example is a one-line script that you can place in the Commands folder. When you need to reload effects, go to the Commands menu and execute the script.

```
fl.reloadEffects();
```

## fl.reloadTools()

### Availability

Flash MX 2004.

### Usage

```
fl.reloadTools()
```

### Parameters

None.

### Returns

Nothing.

### Description

Method; rebuilds the Tools panel from the toolconfig.xml file. This method is used only when creating extensible tools. Use this method when you need to reload the Tools panel, for example, after modifying the JSFL file that defines a tool that is already present in the panel.

### Example

The following example is a one-line script that you can place in the Commands folder. When you need to reload the Tools panel, run the script from the Commands menu.

```
fl.reloadTools();
```

## fl.removeEventListener()

### Availability

Flash CS3 Professional. The `id` parameter is new in CS4.

### Usage

```
fl.removeEventListener(eventType, id)
```

### Parameters

**eventType** A string that specifies the event type to remove from this callback function. Acceptable values are "documentNew", "documentOpened", "documentClosed", "mouseMove", "documentChanged", "layerChanged", "timelineChanged", and "frameChanged".

**id** An integer that specifies the listener ID returned from the corresponding `fl.addEventListener()` call.

### Returns

A Boolean value of `true` if the event listener was successfully removed; `false` if the function was never added to the list with the `fl.addEventListener()` method.

### Description

Unregisters a function that was registered using `fl.addEventListener()`.

**Example**

The following example removes the event listener associated with the `documentClosed` event:

```
fl.removeEventListener("documentClosed", eventID);
```

**See also**

[fl.addEventListener\(\)](#)

## fl.resetAS3PackagePaths()

**Availability**

Flash CS3 Professional.

**Usage**

```
fl.resetAS3PackagePaths();
```

**Parameters**

None.

**Description**

Method; resets the global Classpath setting in the ActionScript 3.0 Settings dialog box to the default value. To reset the ActionScript 2.0 global Classpath, use [fl.resetPackagePaths\(\) - dropped](#).

**Example**

The following example resets the ActionScript 3.0 Classpath setting to its default value.

```
fl.resetAS3PackagePaths();
```

**See also**

[fl.as3PackagePaths](#)

## fl.resetPackagePaths() - dropped

**Availability**

Flash CS3 Professional. *Dropped in Flash Professional CC.*

**Usage**

```
fl.resetPackagePaths();
```

**Parameters**

None.

**Description**

*Dropped in Flash Professional CC.*

Method; resets the global Classpath setting in the ActionScript 2.0 Settings dialog box to the default value. To reset the ActionScript 3.0 global Classpath, use [fl.resetAS3PackagePaths\(\)](#).

**Example**

The following example resets the ActionScript 2.0 Classpath setting to its default value.

```
fl.resetPackagePaths();
```

**See also**

[fl.packagePaths - dropped](#)

## fl.revertDocument()

**Availability**

Flash MX 2004.

**Usage**

```
fl.revertDocument(documentObject)
```

**Parameters**

**documentObject** A [Document object](#). If *documentObject* refers to the active document, the Document window might not revert until the script that calls this method finishes executing.

**Returns**

A Boolean value: `true` if the Revert operation completes successfully; `false` otherwise.

**Description**

Method; reverts the specified FLA document to its last saved version. Unlike the File > Revert menu option, this method does not display a warning window that asks the user to confirm the operation. See also [document.revert\(\)](#) and [document.canRevert\(\)](#).

**Example**

The following example reverts the current FLA document to its last saved version; any changes made since the last save are lost.

```
fl.revertDocument(f1.getDocumentDOM());
```

## fl.runScript()

**Availability**

Flash MX 2004.

**Usage**

```
fl.runScript(fileURI [, funcName [, arg1, arg2, ...]])
```

**Parameters**

**fileURI** A string, expressed as a file:/// URI, that specifies the name of the script file to execute.

**funcName** A string that identifies a function to execute in the JSFL file that is specified in *fileURI*. This parameter is optional.

**arg** An optional parameter that specifies one or more arguments to be passed to *funcname*.

**Returns**

The function's result as a string, if *funcName* is specified; otherwise, nothing.

**Description**

Method; executes a JavaScript file. If a function is specified as one of the arguments, it runs the function and also any code in the script that is not within the function. The rest of the code in the script runs before the function is run.

**Example**

Suppose there is a script file named testScript.jsfl in the root directory on the C drive and its contents are as follows:

```
function testFunct(num, minNum) {  
    fl.trace("in testFunct: 1st arg: " + num + " 2nd arg: " + minNum);  
}  
for (i=0; i<2; i++) {  
    fl.trace("in for loop i=" + i);  
}  
fl.trace("end of for loop");  
// End of testScript.jsfl
```

If you issue the following command,

```
fl.runScript("file:///C|/testScript.jsfl", "testFunct", 10, 1);
```

the following information appears in the Output panel:

```
in for loop i=0  
in for loop i=1  
end of for loop  
in testFunct: 1st arg: 10 2nd arg: 1
```

You can also just call testScript.jsfl without executing a function, as follows:

```
fl.runScript("file:///C|/testScript.jsfl");
```

This produces the following in the Output panel:

```
in for loop i=0  
in for loop i=1  
end of for loop
```

## **fl.saveAll()**

**Availability**

Flash MX 2004.

**Usage**

```
fl.saveAll()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; saves all open documents.

If a file has never been saved, the Save As dialog box displays. If a file has not been modified since the last time it was saved, the file isn't saved. To allow an unsaved or unmodified file to be saved, use `fl.saveDocumentAs()`.

**Example**

The following example saves all open documents:

```
fl.saveAll();
```

**See also**

[document.save\(\)](#), [document.saveAndCompact\(\)](#) - dropped, [fl.saveDocument\(\)](#), [fl.saveDocumentAs\(\)](#)

## fl.saveDocument()

**Availability**

Flash MX 2004.

**Usage**

```
fl.saveDocument(document [, fileURI])
```

**Parameters**

**document** A [Document](#) object that specifies the document to be saved. If *document* is `null`, the active document is saved.

**fileURI** A string, expressed as a file:/// URI, that specifies the name of the saved document. If the *fileURI* parameter is `null` or omitted, the document is saved with its current name. This parameter is optional.

**Returns**

A Boolean value: `true` if the save operation completes successfully; `false` otherwise.

This method save the file regardless of whether it is new, modified, or unmodified.

**Description**

Method; saves the specified document as a FLA document.

**Example**

The following example saves the current document and two specified documents:

```
// Save the current document.  
alert(f1.saveDocument(f1.getDocumentDOM()));  
// Save the specified documents.  
alert(f1.saveDocument(f1.documents[0], "file:///C|/example1.fla"));  
alert(f1.saveDocument(f1.documents[1], "file:///C|/example2.fla"));
```

**See also**

[document.save\(\)](#), [document.saveAndCompact\(\)](#) - dropped, [f1.saveAll\(\)](#), [f1.saveDocumentAs\(\)](#)

## fl.saveDocumentAs()

**Availability**

Flash MX 2004.

**Usage**

```
f1.saveDocumentAs(document)
```

**Parameters**

**document** A [Document](#) object that specifies the document to save. If *document* is null, the active document is saved.

**Returns**

A Boolean value: true if the Save As operation completes successfully; false otherwise.

**Description**

Method; displays the Save As dialog box for the specified document.

**Example**

The following example prompts the user to save the specified document and then displays an alert message that indicates whether the document was saved:

```
alert(f1.saveDocumentAs(f1.documents[1]));
```

**See also**

[document.save\(\)](#), [document.saveAndCompact\(\)](#) - dropped, [f1.saveAll\(\)](#), [f1.saveDocument\(\)](#)

## fl.scriptURI

**Availability**

Flash CS3 Professional.

**Usage**

```
f1.scriptURI
```

**Description**

Read-only property; a string that represents the path of the currently running JSFL script, expressed as a file:/// URI. If the script was called from [fl.runScript\(\)](#), this property represents the path of the immediate parent script. That is, it doesn't traverse multiple calls to [fl.runScript\(\)](#) to find the path of the original calling script.

**Example**

The following example displays the path of the currently running JSFL script in the Output panel:

```
fl.trace(f1.scriptURI);
```

**See also**

[fl.runScript\(\)](#)

## fl.selectElement()

**Availability**

Flash CS3 Professional.

**Usage**

```
fl.selectElement(elementObject, editMode)
```

**Parameters**

**elementObject** The [Element object](#) you want to select.

**editMode** A Boolean value that specifies whether you want to edit the element (`true`) or want only to select it (`false`).

**Returns**

A Boolean value of `true` if the element was successfully selected; `false` otherwise.

**Description**

Method; enables selection or editing of an element. Generally, you will use this method on objects returned by [fl.findObjectInDocByName\(\)](#) or [fl.findObjectInDocByType\(\)](#).

**Example**

The following example selects an element named "second text field" if one is found in the document:

```
var nameToSearchFor = "second text field";
var doc = fl.getDocumentDOM();

// Start by viewing Scene 1 (index value of 0).
document.editScene(0);

// Search for element by name.
var results = fl.findObjectInDocByName(nameToSearchFor, doc);
if (results.length > 0) {
    // Select the first element found.
    // Pass false, so the symbolInstance you are searching for is selected.
    // If you pass true, the symbol instance will switch to edit mode.
    fl.selectElement(results[0], false);
    alert("success, found " + results.length + " objects")
}
else {
    alert("failed, no objects with name '" + nameToSearchFor + "' found");
}
```

**See also**

[fl.findObjectInDocByName\(\)](#), [fl.findObjectInDocByType\(\)](#)

## fl.selectTool()

**Availability**

Flash CS3 Professional.

**Usage**

`fl.selectTool(toolName)`

**Parameters**

**toolName** A string that specifies the name of the tool to select. See “Description” below for information on acceptable values for this parameter.

**Description**

Method; selects the specified tool in the Tools panel. The acceptable default values for *toolName* are "arrow", "bezierSelect", "freeXform", "fillXform", "lasso", "pen", "penplus", "penminus", "penmodify", "text", "line", "rect", "oval", "rectPrimitive", "ovalPrimitive", "polystar", "pencil", "brush", "inkBottle", "bucket", "eyeDropper", "eraser", "hand", and "magnifier".

If you or a user creates custom tools, the names of those tools can also be passed as the *toolName* parameter. The list of tool names is located in the following file:

- Windows 7:

`boot drive\Users\username\AppData\Local\Adobe\Flash CC\language\Configuration\Tools\toolConfig.xml`

- Mac OS X:

`Macintosh HD/Users/username/Library/Application Support/Adobe/Flash CS3/language/Configuration/Tools/toolConfig.xml`

**Example**

The following example selects the Pen tool.

```
fl.selectTool("pen");
```

**See also**

[Tools object](#), [ToolObj object](#)

## fl.setActiveWindow()

**Availability**

Flash MX 2004.

**Usage**

```
fl.setActiveWindow(document [, bActivateFrame])
```

**Parameters**

**document** A [Document object](#) that specifies the document to select as the active window.

**bActivateFrame** An optional parameter that is ignored by Flash and Fireworks and is present only for compatibility with Dreamweaver.

**Returns**

Nothing.

**Description**

Method; sets the active window to be the specified document. This method is also supported by Dreamweaver and Fireworks. If the document has multiple views (created by Window > Duplicate Window), the most recently active view is selected.

**Example**

The following example shows two ways to activate a specified document:

```
fl.setActiveWindow(fl.documents[0]);  
  
var theIndex = fl.findDocumentIndex("myFile.fla");  
fl.setActiveWindow(fl.documents[theIndex]);
```

## fl.showIdleMessage()

**Availability**

Flash 8.

**Usage**

```
fl.showIdleMessage(show)
```

**Parameters**

**show** A Boolean value specifying whether to enable or disable the warning about a script running too long.

**Returns**

Nothing.

**Description**

Method; lets you disable the warning about a script running too long (pass `false` for `show`). You might want to do this when processing batch operations that take a long time to complete. To re-enable the alert, issue the command again, this time passing `true` for `show`.

**Example**

The following example illustrates how to disable and re-enable the warning about a script running too long:

```
fl.showIdleMessage(false);  
var result = timeConsumingFunction();  
fl.showIdleMessage(true); ;  
var result = timeConsumingFunction();
```

## fl.setPrefBoolean()

**Availability**

Flash Professional CC.

**Usage**

```
fl.setPrefBoolean(keySection, keyName, keyValue)
```

**Parameters**

**keySection** A string that contains the preferences section that contains `keyName`. (usually this is "Settings").

**keyName** A string that contains the name of the boolean preference setting to be set.

**keyValue** A string that contains the value to be set (true or false).

**Returns**

None.

**Description**

Method; sets a boolean preference value.

**Example**

The following example calls the `fl.setPrefBoolean()` method:

```
fl.setPrefBoolean("BridgeTalk", "LogIncoming", true);
```

## fl.sourcePath

### Availability

Flash CS4 Professional.

### Usage

```
fl.sourcePath
```

### Description

Property; a string that contains a list of items in the global ActionScript 3.0 Source path, which specifies the location of ActionScript class files. Items in the string are delimited by semi-colons. In the authoring tool, the items are specified by choosing Edit > Preferences > ActionScript > ActionScript 3.0 Settings.

### Example

The following example adds the /Classes folder to the global ActionScript 3.0 Source path:

```
fl.trace(fl.sourcePath);
fl.sourcePath = "/Classes;" + fl.sourcePath;
fl.trace(fl.sourcePath);
```

### See also

[fl.flexSDKPath](#), [fl.externalLibraryPath](#), [fl.libraryPath](#), [document.sourcePath](#)

## fl.spriteSheetExporter

### Availability

Flash Pro CS6.

### Usage

```
fl.spriteSheetExporter
```

### Description

Property; returns an instance of the SpriteSheetExporter object.

### Example

The following example returns a reference to the SpriteSheetExporter object:

```
fl.trace(fl.spriteSheetExporter);
```

## fl.swfPanels

### Availability

Flash CS4 Professional.

**Usage**

```
fl.swfPanels
```

**Description**

Read-only property; an array of registered swfPanel objects (see [swfPanel object](#)). A swfPanel object is registered if it has been opened at least once.

A panel's position in the array represents the order in which it was opened. If the first panel opened is named TraceBitmap and the second panel opened is named AnotherFunction, then `fl.swfPanels[0]` is the TraceBitmap swfPanel object, `fl.swfPanels[1]` is the AnotherFunction swfPanel object, and so on.

This property is `undefined` if none of panels have been opened (this behavior is new in Flash Professional CC).

**Example**

The following code displays the name and path of any registered Window SWF panels in the Output panel:

```
if(fl.swfPanels.length > 0) {
    for(x = 0; x < fl.swfPanels.length; x++) {
        fl.trace("Panel: " + fl.swfPanels[x].name + " -- Path: " + fl.swfPanels[x].path);
    }
}
```

## fl.toggleBreakpoint()

**Availability**

Flash Professional CC.

**Usage**

```
fl.toggleBreakpoint(String fileURI, int line, Boolean enable)
```

**Parameters**

**fileURI** A string; the URI of the AS file in which to toggle the breakpoint.

**line** An integer; the line number at which to toggle the breakpoint.

**enable** Boolean; if set to true, the breakpoint is enabled. If set to false, the breakpoint is disabled.

**Description**

Toggles a breakpoint for the given .as file at the given line. If `enable` is false, the breakpoint currently stored at that line will be erased.

**Example**

The following example enables a breakpoint at line 10 of the AS file located at C:\AS\breakpointTest.as:

```
fl.toggleBreakpoint("file:///C|/AS/breakpointTest.as", 10, 1);
```

## fl.tools

### Availability

Flash MX 2004.

### Usage

```
fl.tools
```

### Description

Read-only property; an array of Tools objects (see [Tools object](#)). This property is used only when creating extensible tools.

## fl.trace()

### Availability

Flash MX 2004.

### Usage

```
fl.trace (message)
```

### Parameters

**message** A string that appears in the Output panel.

### Returns

Nothing.

### Description

Method; sends a text string to the Output panel, terminated by a new line, and displays the Output panel if it is not already visible. This method is identical to [outputPanel.trace \(\)](#) and works in the same way as the `trace ()` statement in ActionScript.

To send a blank line, use `fl.trace ("")` or `fl.trace ("\n")`. You can use the latter command inline, making `\n` a part of the *message* string.

### Example

The following example displays several lines of text in the Output panel:

```
fl.outputPanel.clear();
fl.trace("Hello World!!!");
var myPet = "cat";
fl.trace("\nI have a " + myPet);
fl.trace("");
fl.trace("I love my " + myPet);
fl.trace("Do you have a " + myPet +"?");
```

## fl.version

### Availability

Flash MX 2004.

### Usage

```
fl.version
```

### Description

Read-only property; the long string version of the Flash authoring tool, including platform.

### Example

The following example displays the version of the Flash authoring tool in the Output panel:

```
alert(fl.version); // For example, WIN 10,0,0,540
```

## fl.xmlPanel()

### Availability

Flash Professional CC.

### Usage

```
fl.xmlPanel(xmlURI)
```

### Parameters

**xmlURI** A URI specifying the XML file that defines the controls in the panel. You must specify the full path name.

### Returns

XMLUI. The object returned contains properties for all controls defined in the XML file. All properties are returned as strings. The returned object will have one predefined property named "dismiss," which will have a string value that is either "accept" or "cancel".

### Description

Method; Launches the XML To UI dialog from a URI that points to an XML-format file.

### Example

The following example calls the `xmlPanel` method:

```
var obj = fl.xmlPanel( fl.configURI + "Commands/Test.xml" );
for (var prop in obj) {
    fl.trace( "property " + prop + " = " + obj[prop] );
}
```

## fl.xmlPanelFromString()

### Availability

Flash Professional CC.

### Usage

```
fl.xmlPanelFromString(xmlString)
```

### Parameters

**xmlString** A string containing XML that defines a dialog.

### Returns

XMLUI.

### Description

Method; Launches the XML To UI dialog from an XML-format string.

### Example

The following example calls the `xmlPanelFromString` method:

```
var str = "<?xml version=\"1.0\"?><dialog id=\"scale-dialog\" title=\"Scale Selection\" buttons=\"accept, cancel\"><grid><columns><column/><column/></columns><rows><row align=\"center\"><label value=\"Scale y:\" control=\"yScale\"/><textbox id=\"yScale\"/></row></rows></grid></dialog>";
var theDialog = fl.xmlPanelFromString(str);
```

## fl.xmlui

### Availability

Flash MX 2004.

### Usage

```
fl.xmlui
```

### Description

Read-only property; an [XMLUI object](#). This property lets you get and set XMLUI properties in a XMLUI dialog box and lets you accept or cancel the dialog box programmatically.

### Example

See [XMLUI object](#).

# Chapter 18: FLfile object

## FLfile summary

### Availability

Flash MX 2004 7.2.

### Description

The FLfile object lets you write Flash extensions that can access, modify, and remove files and folders on the local file system. The FLfile API is provided in the form of an extension to the JavaScript API. This extension is called a *shared library* and is located in the following folder:

- Windows 7 and 8:

*boot drive\Users\username\AppData\Local\Adobe\Flash CC\language\Configuration\External Libraries\FLfile.dll*

- Mac OS X:

*Macintosh HD/Users/username/Library/Application Support/Adobe/Flash CC/language/Configuration/External Libraries/FLfile.dll*

**Note:** Don't confuse the shared libraries that contain symbols in your Flash documents with the JavaScript API shared libraries. They are two different things.

The FLfile methods work with files or folders (directories) on disk. Therefore, each method takes one or more parameters to specify the location of a file or folder. The location of the file or folder is expressed as a string in a form very similar to a website URL. It is called a file URI (Uniform Resource Identifier) and is formatted as shown here (including the quote marks):

```
"file:///drive|/folder 1/folder 2/.../filename"
```

For example, if you want to create a folder on the C drive called config and place it in the Program Files/MyApp folder, use the following command:

```
FLfile.createFolder("file:///C|/Program Files/MyApp/config");
```

If you then want to place a file called config.ini in that folder, use the following command:

```
FLfile.write("file:///C|/Program Files/MyApp/config/config.ini", "");
```

To create a folder on the Macintosh, you could use the following command:

```
FLfile.createFolder("file:///Macintosh/MyApp/config");
```

### Method summary

The following methods can be used with the FLfile object:

Method	Description
<a href="#">FLfile.copy()</a>	Copies a file.
<a href="#">FLfile.createFolder()</a>	Creates one or more folders.
<a href="#">FLfile.exists()</a>	Determines the existence of a file or folder.

Method	Description
<code>FLfile.getAttributes()</code>	Finds out whether a file is writable, read-only, hidden, visible, or a system folder.
<code>FLfile.getCreationDate()</code>	Specifies how many seconds have passed between January 1, 1970 and the time the file or folder was created.
<code>FLfile.getCreationDateObj()</code>	Gets the date a file or folder was created.
<code>FLfile.getModificationDate()</code>	Specifies how many seconds have passed between January 1, 1970 and the time the file or folder was last modified.
<code>FLfile.getModificationDateObj()</code>	Gets the date a file or folder was last modified.
<code>FLfile.getSize()</code>	Gets the size of a file.
<code>FLfile.listFolder()</code>	Lists the contents of a folder.
<code>FLfile.platformPathToURI()</code>	Converts a filename in a platform-specific format to a file:/// URI.
<code>FLfile.read()</code>	Reads the contents of a file.
<code>FLfile.remove()</code>	Deletes a file or folder.
<code>FLfile.setAttributes()</code>	Makes a file or folder read-only, writable, hidden, or visible.
<code>FLfile.uriToPlatformPath()</code>	Converts a filename expressed as a file:/// URI to a platform-specific format.
<code>FLfile.write()</code>	Creates, writes to, or appends to a file.

## FLfile.copy()

### Availability

Flash MX 2004 7.2.

### Usage

```
FLfile.copy(fileURI, copyURI)
```

### Parameters

**fileURI** A string, expressed as a file:/// URI, that specifies the file you want to copy.

**copyURI** A string, expressed as a file:/// URI, that specifies the location and name of the copied file.

### Returns

A Boolean value of `true` if successful; `false` otherwise.

### Description

Method; copies a file from one location to another. This method returns `false` if *copyURI* already exists.

### Example

The following example makes a backup copy of a configuration file named config.ini and places it inside the same folder in which it is located, with a new name:

```
var originalFileURI="file:///C|/Program Files/MyApp/config.ini";
var newFileURI="file:///C|/Program Files/MyApp/config_backup.ini";
FLfile.copy(originalFileURI, newFileURI);
```

If you prefer, you can perform the same task with a single command:

```
FLfile.copy("file:///C|:/Program Files/MyApp/config.ini", "file:///C|/Program
Files/MyApp/config_backup.ini");
```

## FLfile.createFolder()

### Availability

Flash MX 2004 7.2.

### Usage

```
FLfile.createFolder(folderURI)
```

### Parameters

**folderURI** A folder URI that specifies the folder structure you want to create.

### Returns

A Boolean value of `true` if successful; `false` if *folderURI* already exists.

### Description

Method; creates one or more folders at the specified location.

You can create multiple folders at one time. For example, the following command creates both the *MyData* and the *TempData* folders if they don't already exist:

```
FLfile.createFolder("file:///c|/MyData/TempData")
```

### Example

The following example creates a folder and a subfolder under the configuration folder (`fl.configURI`):

```
fl.trace(FLfile.createFolder(fl.configURI+"folder01/subfolder01"));
```

The following example attempts to create a folder called *tempFolder* at the root level on the C drive and displays an alert box indicating whether the operation was successful:

```
var folderURI = "file:///c|/tempFolder";
if (FLfile.createFolder(folderURI)) {
    alert("Created " + folderURI);
}
else {
    alert(folderURI + " already exists");
}
```

### See also

[FLfile.remove\(\)](#), [FLfile.write\(\)](#)

## FLfile.exists()

### Availability

Flash MX 2004 7.2.

### Usage

```
FLfile.exists(fileURI)
```

### Parameters

**fileURI** A string, expressed as a file:/// URI, that specifies the file you want to verify.

### Returns

A Boolean value of `true` if successful; `false` otherwise.

### Description

Method; determines whether a specified file exists. If you specify a folder and a filename, the folder must already exist. To create folders, see [FLfile.createFolder\(\)](#).

### Examples

The following example checks for a file called mydata.txt in the temp folder and displays an alert box indicating whether the file exists:

```
var fileURI = "file:///c|/temp/mydata.txt";
if (FLfile.exists(fileURI)) {
    alert( fileURI + " exists.");
}
else {
    alert( fileURI + " does not exist.");
}
```

The following example checks to see if a required configuration file exists in the MyApplication folder. If the file doesn't exist, it is created.

```
var configFile = "file:///C|/MyApplication/config.ini";
if (!FLfile.exists(configFile)) {
    FLfile.write(configFile, "");
}
```

### See also

[FLfile.write\(\)](#)

## FLfile.getAttributes()

### Availability

Flash MX 2004 7.2.

### Usage

```
FLfile.getAttributes(fileOrFolderURI)
```

**Parameters**

**fileOrFolderURI** A string, expressed as a file:/// URI, specifying the file or folder whose attributes you want to retrieve.

**Returns**

A string that represents the attributes of the specified file or folder.

Results are unpredictable if the file or folder doesn't exist. You should use [FLfile.exists\(\)](#) before using this method.

**Description**

Method; returns a string representing the attributes of the specified file or folder, or an empty string if the file has no specific attributes (that is, it is not read-only, not hidden, and so on). You should always use [FLfile.exists\(\)](#) to test for the existence of a file or folder before using this method.

Characters in the string represent the attributes as follows:

- R — *fileOrFolderURI* is read-only
- D — *fileOrFolderURI* is a folder (directory)
- H — *fileOrFolderURI* is hidden (Windows only)
- S — *fileOrFolderURI* is a system file or folder (Windows only)
- A — *fileOrFolderURI* is ready for archiving (Windows only)

For example, if *fileOrFolderURI* is a hidden folder, the string returned is "DH".

**Example**

The following example gets the attributes of the file mydata.txt and displays an alert box if the file is read-only.

```
var URI = "file:///c|/temp/mydata.txt";
if (FLfile.exists(URI)){
    var attr = FLfile.getAttributes(URI);
    if (attr && (attr.indexOf("R") != -1)) { // Returned string contains R.
        alert(URI + " is read only!");
    }
}
```

**See also**

[FLfile.setAttributes\(\)](#)

## FLfile.getCreationDate()

**Availability**

Flash MX 2004 7.2.

**Usage**

`FLfile.getCreationDate(fileOrFolderURI)`

**Parameters**

**fileOrFolderURI** A string, expressed as a file:/// URI, specifying the file or folder whose creation date and time you want to retrieve as a hexadecimal string.

**Returns**

A string containing a hexadecimal number that represents the number of seconds that have elapsed between January 1, 1970 and the time the file or folder was created, or "00000000" if the file or folder doesn't exist.

**Description**

Method; specifies how many seconds have passed between January 1, 1970 and the time the file or folder was created. This method is used primarily to compare the creation or modification dates of files or folders.

**Example**

The following example determines whether a file has been modified since it was created:

```
// Make sure the specified file exists
var fileURI = "file:///C|/MyApplication/MyApp.fla";
var creationTime = FLfile.getCreationDate(fileURI);
var modificationTime = FLfile.getModificationDate(fileURI);
if ( modificationTime > creationTime ) {
    alert("The file has been modified since it was created.");
}
else {
    alert("The file has not been modified since it was created.");
}
```

**See also**

[FLfile.getCreationDateObj\(\)](#), [FLfile.getModificationDate\(\)](#)

## FLfile.getCreationDateObj()

**Availability**

Flash MX 2004 7.2.

**Usage**

`FLfile.getCreationDateObj(fileOrFolderURI)`

**Parameters**

**fileOrFolderURI** A string, expressed as a file:/// URI, specifying the file or folder whose creation date and time you want to retrieve as a JavaScript Date object.

**Returns**

A JavaScript Date object that represents the date and time when the specified file or folder was created. If the file doesn't exist, the object contains information indicating that the file or folder was created at midnight GMT on December 31, 1969.

**Description**

Method; returns a JavaScript Date object that represents the date and time when the specified file or folder was created.

**Example**

The following example displays (in human-readable form) the date a file was created, in the Output panel:

```
// Make sure the specified file exists.  
var file1Date = FLfile.getCreationDateObj("file:///c|/temp/file1.txt");  
fl.trace(file1Date);
```

**See also**

[FLfile.getCreationDate\(\)](#), [FLfile.getModificationDateObj\(\)](#)

## FLfile.getModificationDate()

**Availability**

Flash MX 2004 7.2.

**Usage**

```
FLfile.getModificationDate(fileOrFolderURI)
```

**Parameters**

**fileOrFolderURI** A string, expressed as a file:/// URI, specifying the file whose modification date and time you want to retrieve as a hexadecimal string.

**Returns**

A string containing a hexadecimal number that represents the number of seconds that have elapsed between January 1, 1970 and the time the file or folder was last modified, or "00000000" if the file doesn't exist.

**Description**

Method; specifies how many seconds have passed between January 1, 1970 and the time the file or folder was last modified. This method is used primarily to compare the creation or modification dates of files or folders.

**Example**

The following example compares the modification dates of two files and determines which of the two was modified more recently:

```
// Make sure the specified files exist.  
file1 = "file:///C|/MyApplication/MyApp.fla";  
file2 = "file:///C|/MyApplication/MyApp.as";  
modificationTime1 = FLfile.getModificationDate(file1);  
modificationTime2 = FLfile.getModificationDate(file2) ;  
if(modificationTime1 > modificationTime2) {  
    alert("File 2 is older than File 1") ;  
}  
else if(modificationTime1 < modificationTime2) {  
    alert("File 1 is older than File 2") ;  
}  
else {  
    alert("File 1 and File 2 were saved at the same time") ;  
}
```

**See also**

[FLfile.getCreationDate\(\)](#), [FLfile.getModificationDateObj\(\)](#)

## FLfile.getModificationDateObj()

### Availability

Flash MX 2004 7.2.

### Usage

```
FLfile.getModificationDateObj (fileOrFolderURI)
```

### Parameters

**fileOrFolderURI** A string, expressed as a file:/// URI, specifying the file or folder whose modification date and time you want to retrieve as a JavaScript Date object.

### Returns

A JavaScript Date object that represents the date and time when the specified file or folder was last modified. If the file or folder doesn't exist, the object contains information indicating that the file or folder was created at midnight GMT on December 31, 1969.

### Description

Method; returns a JavaScript Date object that represents the date and time when the specified file or folder was last modified.

### Example

The following example displays (in human-readable form) the date a file was last modified, in the Output panel:

```
// Make sure the specified file exists.  
var file1Date = FLfile.getModificationDateObj ("file:///c|/temp/file1.txt");  
trace(file1Date);
```

### See also

[FLfile.getCreationDateObj\(\)](#), [FLfile.getModificationDate\(\)](#)

## FLfile.getSize()

### Availability

Flash MX 2004 7.2.

### Usage

```
FLfile.getSize (fileURI)
```

### Parameters

**fileURI** A string, expressed as a file:/// URI, specifying the file whose size you want to retrieve.

### Returns

An integer that represents the size of the specified file, in bytes, or 0 if the file doesn't exist.

**Description**

Method; returns an integer that represents the size of the specified file, in bytes, or 0 if the file doesn't exist. If the return value is 0, you can use [FLfile.exists\(\)](#) to determine whether the file is a zero-byte file or the file doesn't exist.

This method returns correct file size values only for files that are less than or equal to 2GB in size.

**Example**

The following example stores the size of the mydata.txt file in the `fileSize` variable:

```
var URL = "file:///c|/temp/mydata.txt";
var fileSize = FLfile.getSize(URL);
```

## FLfile.listFolder()

**Availability**

Flash MX 2004 7.2.

**Usage**

```
FLfile.listFolder(folderURI [, filesOrDirectories])
```

**Parameters**

**folderURI** A string, expressed as a file:/// URI, specifying the folder whose contents you want to retrieve. You can include a wildcard mask as part of *folderURI*. Valid wildcards are \* (matches one or more characters) and ? (matches a single character).

**filesOrDirectories** An optional string that specifies whether to return only filenames or only folder (directory) names. If omitted, both filenames and folder names are returned. Acceptable values are "files" and "directories".

**Returns**

An array of strings representing the contents of the folder. If the folder doesn't exist or if no files or folders match the specified criteria, returns an empty array.

**Description**

Method; returns an array of strings representing the contents of the folder.

**Examples**

The following example returns three arrays. The first represents all the files in the C:\temp folder, the second represents all the folders in the C:\temp folder, and the third represents the files and folders in the C:\temp folder:

```
var fileURI = "file:///C|/temp/" ;
var folderURI = "file:///C|/temp" ;
var fileList1 = FLfile.listFolder(fileURI, "files"); // files
var fileList2 = FLfile.listFolder(folderURI, "directories"); //folders
var fileList3 = FLfile.listFolder(folderURI); //files and folders
fl.trace("Files: " + fileList1);
fl.trace("");
fl.trace("Folders: " + fileList2);
fl.trace("");
fl.trace("Files and folders: " + fileList3);
```

The following example returns an array of all the text (.txt) files in the temp folder and displays the list in an alert box:

```
var folderURI = "file:///c|/temp";
var fileMask = "*.*txt";
var list = FLfile.listFolder(folderURI + "/" + fileMask, "files");
if (list) {
    alert(folderURI + " contains: " + list.join(" "));
}
```

The following example uses a file mask in the specified *folderURI* to return the names of all the executable files in the Windows application folder:

```
var executables = FLfile.listFolder("file:///C|/WINDOWS/*.*exe", "files");
alert(executables.join("\n"));
```

## FLfile.platformPathToURI()

### Availability

Flash CS4 Professional.

### Usage

```
FLfile.platformPathToURI(fileName)
```

### Parameters

**fileName** A string, expressed in a platform-specific format, specifying the filename you want to convert.

### Returns

A string expressed as a file:/// URI.

### Description

Method; converts a filename in a platform-specific format to a file:/// URI.

### Example

The following example converts a filename from a platform-specific format to a file:/// URI, which is passed to *outputPanel.save()*:

```
var myFilename = "C:\outputPanel.txt";
var myURI=FLfile.platformPathToURI(myFilename);
fl.outputPanel.save(myURI);
```

### See also

[FLfile.uriToPlatformPath\(\)](#)

## FLfile.read()

### Availability

Flash MX 2004 7.2.

**Usage**

```
FLfile.read(fileURI)
```

**Parameters**

**fileURI** A string, expressed as a file:/// URI, specifying the text-based file (such as .js, .txt, or .jsfl) that you want to read.

**Returns**

The contents of the specified file as a string, or `null` if the read fails.

**Description**

Method; returns the contents of the specified file as a string, or `null` if the read fails.

**Examples**

The following example reads the file mydata.txt and, if successful, displays an alert box with the contents of the file.

```
var fileURI = "file:///c|/temp/mydata.txt";
var str = FLfile.read( fileURI );
if (str) {
    alert( fileURL + " contains: " + str );
}
```

The following example reads the ActionScript code from a class file and stores it in the `code` variable:

```
var classFileURI = "file:///C|/MyApplication/TextCarousel.as";
var code = FLfile.read(classFileURI);
```

## FLfile.remove()

**Availability**

Flash MX 2004 7.2.

**Usage**

```
FLfile.remove(fileOrFolderURI)
```

**Parameters**

**fileOrFolderURI** A string, expressed as a file:/// URI, specifying the file or folder you want to remove (delete).

**Returns**

A Boolean value of `true` if successful; `false` otherwise.

**Description**

Method; deletes the specified file or folder. If the folder contains files, those files will be deleted as well. Files with the R (read-only) attribute cannot be removed.

**Examples**

The following example warns a user if a file exists and then deletes it if the user chooses to do so:

```
var fileURI = prompt ("Enter file/folder to be deleted: ", "file:///c|/temp/delete.txt");
if (FLfile.exists(fileURI)) {
    var confirm = prompt("File exists. Delete it? (y/n)", "y");
    if (confirm == "y" || confirm == "Y") {
        if(FLfile.remove(fileURI)) {
            alert(fileURI + " is deleted.");
        }
        else {
            alert("fail to delete " + fileURI);
        }
    }
}
else {
    alert(fileURI + " does not exist");
}
```

The following example deletes a configuration file created by an application:

```
if(FLfile.remove("file:///C|/MyApplication/config.ini")) {
    alert("Configuration file deleted");
}
```

The following example deletes the Configuration folder and its contents:

```
FLfile.remove("file:///C|/MyApplication/Configuration/");
```

#### See also

[FLfile.createFolder\(\)](#), [FLfile.getAttributes\(\)](#)

## FLfile.setAttributes()

#### Availability

Flash MX 2004 7.2.

#### Usage

```
FLfile.setAttributes(fileURI, strAttrs)
```

#### Parameters

**fileURI** A string, expressed as a file:/// URI, specifying the file whose attributes you want to set.

**strAttrs** A string specifying values for the attribute(s) you want to set. For acceptable values for *strAttrs*, see the “Description” section below.

#### Returns

A Boolean value of true if successful.

**Note:** Results are unpredictable if the file or folder doesn't exist. You should use [FLfile.exists\(\)](#) before using this method.

#### Description

Method; specifies system-level attributes for the specified file.

The following values are valid for *strAttrs*:

- **N** — No specific attributes (not read-only, not hidden, and so on)
- **A** — Ready for archiving (Windows only)
- **R** — Read-only (on the Macintosh, read-only means “locked”)
- **W** — Writable (overrides **R**)
- **H** — Hidden (Windows only)
- **V** — Visible (overrides **H**, Windows only)

If you include both **R** and **W** in *strAttrs*, the **R** is ignored and the file is set as writable. Similarly, if you pass **H** and **V**, the **H** is ignored and the file is set as visible.

If you want to make sure the archive attribute is not set, use this command with the **N** parameter before setting attributes. That is, there is no direct counterpart to **A** that turns off the archive attribute.

### Examples

The following example sets the file mydata.txt to be read-only and hidden. It has no effect on the archive attribute.

```
var URI = "file:///c|/temp/mydata.txt";
if (FLfile.exists(URI)) {
    FLfile.setAttributes(URI, "RH");
}
```

The following example sets the file mydata.txt to be read-only and hidden. It also ensures that the archive attribute is not set.

```
var URI = "file:///c|/temp/mydata.txt";

if (FLfile.exists(URI)) {
    FLfile.setAttributes(URI, "N");
    FLfile.setAttributes(URI, "RH");
}
```

### See also

[FLfile.getAttributes\(\)](#)

## FLfile.uriToPlatformPath()

### Availability

Flash CS4 Professional.

### Usage

```
FLfile.uriToPlatformPath(fileURI)
```

### Parameters

**fileURI** A string, expressed as a file:/// URI, specifying the filename you want to convert.

**Returns**

A string representing a platform-specific path.

**Description**

Method; converts a filename expressed as a file:/// URI to a platform-specific format.

**Example**

The following example converts a file:/// URI to a platform-specific format:

```
var dir =(fl.configDirectory);
var URI = FLfile.platformPathToURI(dir);
fl.trace(URI == fl.configURI); // displays "true"
```

**See also**

[FLfile.platformPathToURI\(\)](#)

## FLfile.write()

**Availability**

Flash MX 2004 7.2.

**Usage**

```
FLfile.write(fileURI, textToWrite, [ , strAppendMode])
```

**Parameters**

**fileURI** A string, expressed as a file:/// URI, specifying the file to which you want to write.

**textToWrite** A string representing the text you want to place in the file.

**strAppendMode** An optional string with the value "append", which specifies that you want to append *textToWrite* to the existing file. If omitted, *fileURI* is overwritten with *textToWrite*.

**Returns**

A Boolean value of `true` if successful; `false` otherwise.

**Description**

Method; writes the specified string to the specified file (as UTF-8). If the specified file does not exist, it is created. However, the folder in which you are placing the file must exist before you use this method. To create folders, use [FLfile.createFolder\(\)](#).

**Example**

The following example attempts to write the string "xxx" to the file mydata.txt and displays an alert message if the write succeeded. It then attempts to append the string "aaa" to the file and displays a second alert message if the write succeeded. After executing this script, the file mydata.txt will contain only the text "xxxxaa".

```
var URI = "file:///c|/temp/mydata.txt";
if (FLfile.write(URI, "xxx")) {
    alert("Wrote xxx to " + URI);
}
if (FLfile.write(URI, "aaa", "append")) {
    alert("Appended aaa to " + fileURI);
}
```

**See also**

[FLfile.createFolder\(\)](#), [FLfile.exists\(\)](#)

# Chapter 19: folderItem object

## folderItem summary

**Inheritance**    [Item object](#) > folderItem object

### Availability

Flash MX 2004.

### Description

The folderItem object is a subclass of the Item object. There are no unique methods or properties for folderItem. See [Item object](#).

# Chapter 20: fontItem object

## fontItem summary

**Inheritance** [Item object](#) > fontItem object

### Availability

Flash MX 2004.

### Description

The fontItem object is a subclass of the Item object (see [Item object](#)).

### Property summary

In addition to the Item object properties, the following properties are available for the fontItem object:

Property	Description
<code>fontItem.bitmap</code>	Specifies whether the Font item is bitmapped.
<code>fontItem.bold</code>	Specifies whether the Font item is bold.
<code>fontItem.embeddedCharacters</code>	Specifies characters to embed.
<code>fontItem.embedRanges</code>	Specifies items that can be selected in the Font Embedding dialog.
<code>fontItem.embedVariantGlyphs</code>	Specifies whether variant glyphs should be output in the font when publishing a SWF file.
<code>fontItem.font</code>	The name of the device font associated with the Font item.
<code>fontItem.isDefineFont4Symbol</code>	Specifies the format of the font that is output when publishing a SWF file.
<code>fontItem.italic</code>	Specifies whether the Font item is italic.
<code>fontItem.size</code>	The size of the Font item, in points.

## fontItem.bitmap

### Availability

Flash CS4 Professional.

### Usage

```
fontItem.bitmap
```

### Description

Property; a Boolean value that specifies whether the Font item is bitmapped (`true`) or not (`false`).

### Example

Assuming that the first item in the Library is a Font item, the following code displays `true` in the Output panel if it is bitmapped, `false` if it is not:

```
var theItem = fl.getDocumentDOM().library.items[0];
fl.trace("bitmap: "+ theItem.bitmap);
```

## fontItem.bold

### Availability

Flash CS4 Professional.

### Usage

```
fontItem.bold
```

### Description

Property; a Boolean value that specifies whether the Font item is bold (`true`) or not (`false`).

### Example

Assuming that the first item in the Library is a Font item, the following code displays `true` in the Output panel if it is bold, `false` if it is not, and then sets it to bold.

```
var theItem = fl.getDocumentDOM().library.items[0];
fl.outputPanel.clear();
fl.trace("bold: "+ theItem.bold);
theItem.bold=true;
fl.trace("bold: "+ theItem.bold);
```

## fontItem.embeddedCharacters

### Availability

Flash CS5 Professional.

### Usage

```
fontItem.embeddedCharacters
```

### Description

Property; a string value that allows you to specify characters to embed within a SWF file so that the characters do not need to be present on the devices the SWF file eventually plays back on. This property provides the same functionality as the Font Embedding dialog box.

This property can also be read, allowing you to find out what characters were specified with the Font Embedding dialog box for a given Font item.

### Example

Assuming that the first item in the Library is a Font item, the following code embeds the characters a, b, and c.

```
fl.getDocumentDOM().library.items[0].embeddedCharacters = "abc";
```

## fontItem.embedRanges

### Availability

Flash CS5 Professional.

### Usage

```
fontItem.embedRanges
```

### Description

Property; a string value that specifies a series of delimited integers that correspond to items that can be selected in the Font Embedding dialog box.

This property can also be read, allowing you to find out what characters were specified with the Font Embedding dialog box for a given Font item.

**Note:** The range numbers correspond to the *FontEmbedding/UnicodeTables.xml* file found in the configuration folder.

### Example

Assuming that the first item in the Library is a Font item, the following code embeds the ranges identified by the integers 1, 3, and 7.

```
f1.getDocumentDOM().library.items[0].embedRanges = "1|3|7";
```

Assuming that the first item in the Library is a Font item, the following code resets the ranges to embed.

```
f1.getDocumentDOM().library.items[0].embedRanges = "";
```

## fontItem.embedVariantGlyphs

### Availability

Flash CS4 Professional.

### Usage

```
fontItem.embedVariantGlyphs
```

### Description

**Note:** While this property is available in Flash CS5 Professional, it has no effect when applied to Text Layout Framework (TLF) text. Beginning in Flash Professional CS5, variant glyphs are always embedded in fonts used with TLF text. The *flash.text.engine* (FTE) referenced below is only available in Flash Professional CS4.

Property; a Boolean value that specifies whether variant glyphs should be output in the font when publishing a SWF file (`true`) or not (`false`). Setting this value to `true` increases the size of your SWF file. The default value is `false`.

Some languages dynamically substitute characters glyphs as you are typing (for example, Thai, Arabic, Hebrew, and Greek). If you are laying out or inputting text in these types of languages, set this property to `true`.

### Examples

Font symbols that are compatible with flash.text APIs appear in the Library and the user can manage them directly. However, font symbols that are compatible with the flash.text.engine (FTE) APIs do not appear in the Library, so you must manage them manually. The following function adds a new font to the Library that can be used with the FTE APIs.

```
function embedFontSymbol(symbolName, fontName, includeVariants) {
    var doc = fl.getDocumentDOM();
    if (doc) {
        // look up the item. if it exists, delete it.
        var index = doc.library.findItemIndex(symbolName);
        if (index > -1)
            doc.library.deleteItem(symbolName);

        // make a new font symbol in the library
        doc.library.addNewItem('font', symbolName);

        // look up the symbol by its name
        var index = doc.library.findItemIndex(symbolName);
        if (index > -1) {
            // get the item from the library and set the attributes of interest
            var fontObj = doc.library.items[index];
            fontObj.isDefineFont4Symbol = true;
            fontObj.font = fontName;
            fontObj.bold = false;
            fontObj.italic = false;
            fontObj.embedVariantGlyphs = includeVariants;
            // this is what forces the font into the SWF stream
            fontObj.linkageExportForAS = true;
            fontObj.linkageExportInFirstFrame = true;
        }
    }
}
```

The following function displays all the font symbols in the Output panel.

```
function dumpFontSymbols()
{
    var doc = fl.getDocumentDOM();
    if (doc) {
        var items = doc.library.items;
        fl.trace("items length = " + items.length);
        var i;
        for(i=0; i<items.length; i++) {
            var item = items[i];
            fl.trace("itemType = " + item.itemType);
            if (item.itemType == 'font') {
                fl.trace("name = " + item.name);
                fl.trace("DF4 symbol = " + item.isDefineFont4Symbol);
                fl.trace("font = " + item.font);
            }
        }
    }
}
```

**See also**

[fontItem.isDefineFont4Symbol](#), [text.embedVariantGlyphs](#)

## fontItem.font

**Availability**

Flash CS4 Professional.

**Usage**

`fontItem.font`

**Description**

Property; a string that specifies the name of the device font associated with the Font item. If you enter a string that does not correspond to an installed device font, an error message is displayed. To determine if a font exists on the system, use `f1.isFontInstalled()`.

**Note:** When you set this value, the resulting property value might be different from the string you enter. See the following example.

**Example**

Assuming that the first item in the Library is a Font item, the following code displays the name of the device font currently associated with the Font item, then changes it to Times:

```
f1.outputPanel.clear();
var theItem = f1.getDocumentDOM().library.items[0];
f1.trace(theItem.font);
theItem.font = "Times";
// depending on your system, the following may display something like "Times-Roman"
f1.trace(theItem.font);
```

## fontItem.isDefineFont4Symbol

**Availability**

Flash CS4 Professional.

**Usage**

`fontItem.isDefineFont4Symbol`

**Description**

Property; a Boolean value that specifies the format of the font that is output when publishing a SWF file. If this value is `true`, Flash outputs a font that can be used with the `flash.text.engine` (FTE) APIs. If this value is `false`, the font can be used with the `flash.text` APIs, including text fields. The default value is `false`.

**Example**

See [fontItem.embedVariantGlyphs](#).

## fontItem.italic

### Availability

Flash CS4 Professional.

### Usage

```
fontItem.italic
```

### Description

Property; a Boolean value that specifies whether the Font item is italic (`true`) or not (`false`).

### Example

Assuming that the first item in the Library is a Font item, the following code displays `true` in the Output panel if it is italic, `false` if it is not, and then sets it to italic.

```
var theItem = fl.getDocumentDOM().library.items[0];
fl.outputPanel.clear();
fl.trace("italic: " + theItem.italic);
theItem.italic=true;
fl.trace("italic: " + theItem.italic);
```

## fontItem.size

### Availability

Flash CS4 Professional.

### Usage

```
fontItem.size
```

### Description

Property; an integer that represents the size of the Font item, in points.

### Example

Assuming that the first item in the Library is a Font item, the following code displays the item's point size in the Output panel and then sets it to 24.

```
var theItem = fl.getDocumentDOM().library.items[0];
fl.outputPanel.clear();
fl.trace("font size: " + theItem.size);
theItem.size=24;
fl.trace("font size: " + theItem.size);
```

# Chapter 21: Frame object

## frame summary

### Availability

Flash MX 2004.

### Description

The Frame object represents frames in the layer.

### Method summary

The following methods can be used with the Frame object:

Method	Description
<code>frame.convertMotionObjectTo2D()</code>	Converts the selected motion object to a 2D motion object.
<code>frame.convertMotionObjectTo3D()</code>	Converts the selected motion object to a 3D motion object.
<code>frame.convertToFrameByFrameAnimation()</code>	Converts the current frame to Frame-by-Frame Animation.
<code>frame.getCustomEase()</code>	Returns an array of JavaScript objects, each of which has an x and y property.
<code>frame.getMotionObjectXML()</code>	Returns the motion XML from the selected motion object.
<code>frame.getSoundEnvelope()</code>	Gets the sound envelope data of any frame.
<code>frame.getSoundEnvelopeLimits()</code>	Gets the limits (start, end) for a custom Sound envelope that is applied to the frame sound.
<code>frame.hasMotionPath()</code>	Informs you whether or not the current selection has a motion tween.
<code>frame.is3DMotionObject()</code>	Informs you whether or not the current selection is a 3D motion object.
<code>frame.isEmpty()</code>	Informs you whether the frame contains any elements.
<code>frame.isMotionObject()</code>	Informs you whether or not the current selection is a motion object.
<code>frame.selectMotionPath()</code>	Selects or deselects the motion path of the current motion object.
<code>frame.setCustomEase()</code>	Specifies a cubic Bézier curve to be used as a custom ease curve.
<code>frame.setMotionObjectDuration()</code>	Specifies the duration (the tween span length) of the currently selected motion object.
<code>frame.setMotionObjectXML()</code>	Applies the specified motion XML to the selected motion object.
<code>frame.setSoundEnvelope()</code>	Sets the sound envelope data of a frame.
<code>frame.setSoundEnvelopeLimits()</code>	Sets the sound envelope limits of any frame with a sound file.

### Property summary

The following properties can be used with the Frame object:

Property	Description
<code>frame.actionScript</code>	A string representing ActionScript code.
<code>frame.duration</code>	Read-only; an integer that represents the number of frames in a frame sequence.
<code>frame.elements</code>	Read-only; an array of Element objects (see <a href="#">Element object</a> ).
<code>frame.hasCustomEase</code>	A Boolean value that specifies whether the frame gets its ease information from the custom ease curve.
<code>frame.labelType</code>	A string that specifies the type of Frame name.
<code>frame.motionTweenOrientToPath</code>	A Boolean value that specifies whether or not the tweened element rotates the element as it moves along a path to maintain its angle with respect to each point on the path.
<code>frame.motionTweenRotate</code>	A string that specifies how the tweened element rotates.
<code>frame.motionTweenRotateTimes</code>	An integer that specifies the number of times the tweened element rotates between the starting keyframe and the next keyframe.
<code>frame.motionTweenScale</code>	A Boolean value that specifies whether the tweened element scales to the size of the object in the following keyframe, increasing its size with each frame in the tween ( <code>true</code> ), or doesn't scale ( <code>false</code> ).
<code>frame.motionTweenSnap</code>	A Boolean value that specifies whether the tweened element automatically snaps to the nearest point on the motion guide layer associated with this frame's layer ( <code>true</code> ) or not ( <code>false</code> ).
<code>frame.motionTweenSync</code>	A Boolean value that if set to <code>true</code> , synchronizes the animation of the tweened object with the main timeline.
<code>frame.name</code>	A string that specifies the name of the frame.
<code>frame.shapeTweenBlend</code>	A string that specifies how a shape tween is blended between the shape in the keyframe at the start of the tween and the shape in the following keyframe.
<code>frame.soundEffect</code>	A string that specifies effects for a sound that is attached directly to a frame ( <code>frame.soundLibraryItem</code> ).
<code>frame.soundLibraryItem</code>	A library item (see <a href="#">SoundItem object</a> ) used to create a sound.
<code>frame.soundLoop</code>	An integer value that specifies the number of times a sound that is attached directly to a frame ( <code>frame.soundLibraryItem</code> ) plays.
<code>frame.soundLoopMode</code>	A string that specifies whether a sound that is attached directly to a frame ( <code>frame.soundLibraryItem</code> ) should play a specific number of times or loop indefinitely.
<code>frame.soundName</code>	A string that specifies the name of a sound that is attached directly to a frame ( <code>frame.soundLibraryItem</code> ), as stored in the library.
<code>frame.soundSync</code>	A string that specifies the sync behavior of a sound that is attached directly to a frame ( <code>frame.soundLibraryItem</code> ).
<code>frame.startFrame</code>	Read-only; the index of the first frame in a sequence.
<code>frame.tweenEasing</code>	An integer that specifies the amount of easing that should be applied to the tweened object.
<code>frame.tintColorName</code>	Assigns an instance name to the specified motion object.
<code>frame.tweenType</code>	A string that specifies the type of tween.
<code>frame.useSingleEaseCurve</code>	A Boolean value that specifies whether a single custom ease curve is used for easing information for all properties.

## frame.convertMotionObjectTo2D()

### Availability

Flash Professional CS5.

### Usage

```
frame.convertMotionObjectTo2D()
```

### Description

Method; Converts the selected motion object to a 2D motion object.

### Example

The following example converts the selected motion object to a 2D motion object:

```
var doc = fl.getDocumentDOM();
var my_t1 = doc.getTimeline();
this.getCurrentFrame = function(){
var layer = my_t1.layers[my_t1.currentLayer];
var frame = layer.frames[my_t1.currentFrame];
return frame;
}
var theFrame = getCurrentFrame();
if(theFrame.isMotionObject() && the()){
theFrame.convertMotionObjectTo2D();
} else{
fl.trace("It isn't motion or it's already a 2D motion");
}
```

## frame.convertMotionObjectTo3D()

### Availability

Flash Professional CS5.

### Usage

```
frame.convertMotionObjectTo3D()
```

### Description

Method; Converts the selected motion object to a 3D motion object.

### Example

The following example converts the selected motion object to a 3D motion object:

```
var doc = fl.getDocumentDOM();
var my_tl = doc.getTimeline();
this.getCurrentFrame = function() {
    var layer = my_tl.layers[my_tl.currentLayer];
    var frame = layer.frames[my_tl.currentFrame];
    return frame;
}
var theFrame = getCurrentFrame();
if(theFrame.isMotionObject() && !theFrame.is3DMotionObject()) {
    theFrame.convertMotionObjectTo3D();
} else {
    fl.trace("It isn't motion or it's already a 3D motion");
}
```

## frame.convertToFrameByFrameAnimation()

### Availability

Flash Professional CC.

### Usage

```
frame.convertToFrameByFrameAnimation()
```

### Returns

Returns boolean. Returns true if the frame contains animation that can be converted to frame by frame animation. For example: return true for Motion Tween frame or Classic Tween frame; return false for other type of frame such as static.

### Description

Method; Converts the current frame to Frame-by-Frame Animation.

### Example

The following example converts the frame 0 to Frame-by-Frame Animation:

```
var result =
fl.getDocumentDOM().getTimeline().layers[0].frames[0].convertToFrameByFrameAnimation();
fl.trace(result);
```

## frame.actionScript

### Availability

Flash MX 2004.

### Usage

```
frame.actionScript
```

### Description

Property; a string that represents ActionScript code. To insert a new line character, use "\n".

**Example**

The following example assigns `stop()` to first frame top layer action:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].actionScript = 'stop();';
```

## frame.duration

**Availability**

Flash MX 2004.

**Usage**

```
frame.duration
```

**Description**

Read-only property; an integer that represents the number of frames in a frame sequence.

**Example**

The following example stores the number of frames in a frame sequence that starts at the first frame in the top layer in the `frameSpan` variable:

```
var frameSpan = f1.getDocumentDOM().getTimeline().layers[0].frames[0].duration;
```

## frame.elements

**Availability**

Flash MX 2004.

**Usage**

```
frame.elements
```

**Description**

Read-only property; an array of Element objects (see [Element object](#)). The order of elements is the order in which they are stored in the FLA file. If there are multiple shapes on the Stage, and each is ungrouped, Flash treats them as one element. If each shape is grouped, so there are multiple groups on the Stage, Flash sees them as separate elements. In other words, Flash treats raw, ungrouped shapes as a single element, regardless of how many separate shapes are on the Stage. If a frame contains three raw, ungrouped shapes, for example, then `elements.length` in that frame returns a value of 1. To work around this issue, select each shape individually and group it .

**Example**

The following example stores an array of current elements in the top layer, first frame in the `myElements` variable:

```
var myElements = f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements;
```

## frame.getCustomEase()

### Availability

Flash 8.

### Usage

```
Frame.getCustomEase( [property] )
```

### Parameters

**property** An optional string that specifies the property for which you want to return the custom ease value. Acceptable values are "all", "position", "rotation", "scale", "color", and "filters". The default value is "all".

### Returns

Returns an array of JavaScript objects, each of which has an *x* and *y* property.

### Description

Method; returns an array of objects that represent the control points for the cubic Bézier curve that defines the ease curve.

### Example

The following example returns the custom ease value of the *position* property for the first frame in the top layer:

```
var theFrame = fl.getDocumentDOM().getTimeline().layers[0].frames[0]
var easeArray = theFrame.getCustomEase("position");
```

### See also

[frame.hasCustomEase](#), [frame.setCustomEase\(\)](#), [frame.useSingleEaseCurve](#)

## frame.getMotionObjectXML()

### Availability

Flash Professional CS5.

### Usage

```
Frame.getMotionObjectXML()
```

### Description

Returns a string of the motion XML from the selected motion object.

### Example

The following example returns the motion XML from the selected motion object.

```
var doc = fl.getDocumentDOM();
var my_tl = doc.getTimeline();
this.getCurrentFrame = function(){
var layer = my_tl.layers[my_tl.currentLayer];
var frame = layer.frames[my_tl.currentFrame];
return frame;
}
var theFrame = getCurrentFrame();
if(theFrame.isMotionObject()) {
//fl.trace(theFrame.getMotionObjectXML());
}else{
fl.trace("It is not motion.");
}
```

## frame.getSoundEnvelope()

### Availability

Flash Professional CC.

### Usage

```
frame.getSoundEnvelope()
```

### Parameters

None

### Returns

Returns a Sound object.

### Description

Method; Gets the sound envelope data of any frame.

### Example

The following example illustrates the use of getSoundEnvelope:

```
// Add a sound item to the first Frame

// Get the sound Envelope
var soundEnv = fl.getDocumentDOM().getTimeline().layers[0].frames[0].getSoundEnvelope();

//Assigning the sound 1 in the library to Frame 2
fl.getDocumentDOM().getTimeline().layers[0].frames[1].soundLibraryItem
=f1.getDocumentDOM().library.items[1];

//Set the Sound Envelope
f1.getDocumentDOM().getTimeline().layers[0].frames[1].setSoundEnvelope(soundEnv);
```

### See also

[frame.setSoundEnvelope\(\)](#)

## frame.getSoundEnvelopeLimits()

### Availability

Flash Professional CC.

### Usage

```
frame.getSoundEnvelopeLimits()
```

### Parameters

None

### Returns

Returns a structure that contain start and end fields.

### Description

Method; Gets the limits (start, end) for a custom Sound envelope that is applied to the frame sound.

### Example

The following example illustrates the use of getSoundEnvelopeLimits:

```
var limits = fl.getDocumentDOM().getTimeline().layers[0].frames[0].getSoundEnvelopeLimits();
fl.trace(limits.start);
fl.trace(limits.end);
```

### See also

[frame.setSoundEnvelopeLimits\(\)](#)

## frame.hasCustomEase

### Availability

Flash 8.

### Usage

```
frame.hasCustomEase
```

### Description

Property; a Boolean value. If `true`, the frame gets its ease information from the custom ease curve. If `false`, the frame gets its ease information from the ease value.

### Example

The following example specifies that the first frame in the top layer should get its ease information from the ease value rather than the custom ease curve:

```
var theFrame = fl.getDocumentDOM().getTimeline().layers[0].frames[0]
theFrame.hasCustomEase = false;
```

**See also**`frame.getCustomEase(), frame.setCustomEase(), frame.useSingleEaseCurve`

## frame.hasMotionPath()

**Availability**

Flash Professional CS5.

**Usage**`Frame.hasMotionPath()`**Description**

Method; a Boolean value. Lets you know whether the current selection includes a motion path.

**Example**

The following example returns a trace statement informing you if the current selection has a motion path.

```
var doc      = fl.getDocumentDOM();
var my_tl = doc.getTimeline() ;
this.getCurrentFrame = function(){
var layer = my_tl.layers[my_tl.currentLayer];
var frame = layer.frames[my_tl.currentFrame];
return frame;
}
var theFrame = getCurrentFrame();
if(theFrame.isMotionObject()){
if (theFrame.hasMotionPath()){
fl.trace("There is a motion path");
}else{
fl.trace("There is no motion path");
}}
```

## frame.is3DMotionObject()

**Availability**

Flash Professional CS5.

**Usage**`Frame.is3DMotionObject()`**Description**

Method; a Boolean value. Lets you know whether the current selection is a 3D motion object.

**Example**

The following example returns a trace statement informing you that the current selection is or is not a 3D motion object.

```
var doc = fl.getDocumentDOM();
var my_tl = doc.getTimeline();
this.getCurrentFrame = function() {
    var layer = my_tl.layers[my_tl.length - 1];
    var frame = layer.frames[my_tl.currentFrame];
    return frame;
}
var theFrame = getCurrentFrame();
if(theFrame.isMotionObject() && theFrame.is3DMotionObject()){
    fl.trace("This selection is 3D Motion");
} else{
    fl.trace("This selection is not 3D motion");
}
```

## frame.isEmpty()

### Availability

Flash Professional CC.

### Usage

```
frame.isEmpty()
```

### Description

Method; a Boolean value. Lets you know whether the frame contains any elements.

### Example

The following example illustrates use of this method.

```
var frame = fl.getDocumentDOM().getTimeline().layers[0].frames[0];
if (frame.isEmpty) fl.trace("first frame is empty");
```

## frame.isMotionObject()

### Availability

Flash Professional CS5.

### Usage

```
Frame.isMotionObject()
```

### Description

Method; a Boolean value. Lets you know whether the current selection is a motion object.

### Example

The following example returns a trace statement informing you that the current selection is or is not a motion object.

```
var my_tl = doc.getTimeline()      ;
this.getCurrentFrame = function(){
var layer = my_tl.layers[my_tl.currentLayer];<
var frame = layer.frames[my_tl.currentFrame];
return frame;
}
var theFrame = getCurrentFrame();
if(theFrame.isMotionObject()) {
fl.trace("This selection is motion.");
}else{
fl.trace("This selection is not motion.");
}
```

## frame.labelType

### Availability

Flash MX 2004.

### Usage

```
frame.labelType
```

### Description

Property; a string that specifies the type of Frame name. Acceptable values are "none", "name", "comment", and "anchor". Setting a label to "none" clears the [frame.name](#) property.

### Example

The following example sets the name of the first frame in the top layer to "First Frame" and then sets its label to "comment":

```
fl.getDocumentDOM().getTimeline().layers[0].frames[0].name = 'First Frame';
fl.getDocumentDOM().getTimeline().layers[0].frames[0].labelType = 'comment';
```

## frame.motionTweenOrientToPath

### Availability

Flash MX 2004.

### Usage

```
frame.motionTweenOrientToPath
```

### Description

Property; a Boolean value that specifies whether the tweened element rotates the element as it moves along a path to maintain its angle with respect to each point on the path (`true`) or whether it does not rotate (`false`).

If you want to specify a value for this property, you should set [frame.motionTweenRotate](#) to "none".

## frame.motionTweenRotate

### Availability

Flash MX 2004.

### Usage

```
frame.motionTweenRotate
```

### Description

Property; a string that specifies how the tweened element rotates. Acceptable values are "none", "auto", "clockwise", and "counter-clockwise". A value of "auto" means the object will rotate in the direction requiring the least motion to match the rotation of the object in the following keyframe.

If you want to specify a value for [frame.motionTweenOrientToPath](#), set this property to "none".

### Example

See [frame.motionTweenRotateTimes](#).

## frame.motionTweenRotateTimes

### Availability

Flash MX 2004.

### Usage

```
frame.motionTweenRotateTimes
```

### Description

Property; an integer that specifies the number of times the tweened element rotates between the starting keyframe and the next keyframe.

### Example

The following example rotates the element in this frame counter-clockwise three times by the time it reaches the next keyframe:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].motionTweenRotate = "counter-clockwise";  
f1.getDocumentDOM().getTimeline().layers[0].frames[0].motionTweenRotateTimes = 3;
```

## frame.motionTweenScale

### Availability

Flash MX 2004.

### Usage

```
frame.motionTweenScale
```

**Description**

Property; a Boolean value that specifies whether the tweened element scales to the size of the object in the following keyframe, increasing its size with each frame in the tween (`true`), or doesn't scale (`false`).

**Example**

The following example specifies that the tweened element should scale to the size of the object in the following keyframe, increasing its size with each frame in the tween.

```
fl.getDocumentDOM().getTimeline().layers[0].frames[0].motionTweenScale = true;
```

## frame.motionTweenSnap

**Availability**

Flash MX 2004.

**Usage**

```
frame.motionTweenSnap
```

**Description**

Property; a Boolean value that specifies whether the tweened element automatically snaps to the nearest point on the motion guide layer associated with this frame's layer (`true`) or not (`false`).

## frame.motionTweenSync

**Availability**

Flash MX 2004.

**Usage**

```
frame.motionTweenSync
```

**Description**

Property; a Boolean value that if set to `true`, synchronizes the animation of the tweened object with the main timeline.

**Example**

The following example specifies that tweened object should be synchronized with the timeline:

```
fl.getDocumentDOM().getTimeline().layers[0].frames[0].motionTweenSync = true;
```

## frame.name

**Availability**

Flash MX 2004.

**Usage**

```
frame.name
```

**Description**

Property; a string that specifies the name of the frame.

**Example**

The following example sets the name of the first frame, top layer to "First Frame" and then stores the name value in the frameLabel variable:

```
fl.getDocumentDOM().getTimeline().layers[0].frames[0].name = 'First Frame';
var frameLabel = fl.getDocumentDOM().getTimeline().layers[0].frames[0].name;
```

## frame.selectMotionPath()

**Availability**

Flash Professional CS5.

**Usage**

```
Frame.selectMotionPath()
```

**Description**

Method; a Boolean value. Selects (true) or deselects (false) the motion path of the current motion object.

**Example**

The example selects or deselects the motion path of the current motion object.

```
var doc = fl.getDocumentDOM();
var my_tl = doc.getTimeline();
t   his.getCurrentFrame = function(){
var layer = my_tl.layers[my_tl. c u rrentLayer];
var frame = layer.frames[my_tl.currentFrame];
return frame;
}
var theFrame = getCurrentFrame();
if(theFrame.isMotionObject()){
if (theFrame.hasMotionPath()){
theFrame.selectMotionPath(true);
}
else{
fl.trace("There is no motion path");
}
}else{
fl.trace("It is no motion");
}
```

## frame.setCustomEase()

### Availability

Flash 8.

### Usage

```
frame.setCustomEase(property, easeCurve)
```

### Parameters

**property** A string that specifies the property the ease curve should be used for. Acceptable values are "all", "position", "rotation", "scale", "color", and "filters".

**easeCurve** An array of objects that defines the ease curve. Each array element must be a JavaScript object with *x* and *y* properties.

### Returns

Nothing.

### Description

Method; specifies an array of control point and tangent endpoint coordinates that describe a cubic Bézier curve to be used as a custom ease curve. This array is constructed by the horizontal (ordinal: left to right) position of the control points and tangent endpoints.

### Example

The following example sets the ease curve for all properties of the first frame in the first layer to the Bézier curve specified by the *easeCurve* array:

```
var theFrame = fl.getDocumentDOM().getTimeline().layers[0].frames[0];
var easeCurve = [ {x:0,y:0}, {x:.3,y:.3}, {x:.7,y:.7}, {x:1,y:1} ];
theFrame.setCustomEase( "all", easeCurve );
```

### See also

[frame.getCustomEase\(\)](#), [frame.hasCustomEase](#), [frame.useSingleEaseCurve](#)

## frame.setMotionObjectDuration()

### Availability

Flash Professional CS5.

### Usage

```
Frame.setMotionObjectDuration( duration [, stretchExistingKeyframes] )
```

### Parameters

**duration** Specifies the number of frames for the tween span of the selected motion object.

**stretchExistingKeyframes** A boolean value that determines whether the tween span is stretched, or if frames are added, to the end of the last frame.

**Description**

Method; sets the duration (the tween span length) of the currently selected motion object.

**Example**

The following example specifies a duration of 11 frames for the selected motion object.

```
var doc = fl.getDocumentDOM();
var my_tl = doc.getTimeline();
this.getCurrentFrame = function(){
var layer = my_tl.layers[my_tl.currentLayer];
var frame = layer.frames[my_tl.currentFrame];
return frame;
}
var theFrame = getCurrentFrame();
if(theFrame.isMotionObject()){
theFrame.setMotionObjectDuration(11);
}else{
fl.trace("It isn't motion");
}
```

## frame.setMotionObjectXML()

**Availability**

Flash Professional CS5.

**Usage**

```
frame.setMotionObjectXML( xmlstr [, endAtCurrentLocation] )
```

**Parameters**

**xmlstr** A string value that specifies the XML string.

**endAtCurrentLocation** A boolean value that determines whether the tween starts or ends at the current position.

**Description**

Method; applies the specified motion XML to the selected motion object.

**Example**

This example specifies that the motion XML identified as `myMotionXML` be applied to the selected motion object.

```
var doc = fl.getDocumentDOM();
var my_tl = doc.getTimeline();
this.getCurrentFrame = function(){
var layer = my_tl.layers[my_tl.currentLayer];
var frame = layer.frames[my_tl.currentFrame];
return frame;
}
var theFrame = getCurrentFrame();
theFrame.setMotionObjectXML(myMotionXML.toString(), false);
```

## frame.setSoundEnvelope()

### Availability

Flash Professional CC.

### Usage

```
frame.setSoundEnvelope(soundEnv)
```

### Parameters

**soundEnv** A sound envelope.

### Returns

Nothing.

### Description

Method; Sets the sound envelope of any frame with sound file. The soundEnv object is an array and every element of array contains the following properties:

- mark
- leftChannel
- rightChannel

### Example

The following example illustrates the use of setSoundEnvelope:

```
// Add a sound item to the first Frame

// Get the sound Envelope
var soundEnv = fl.getDocumentDOM().getTimeline().layers[0].frames[0].getSoundEnvelope();

//Assigning the sound 1 in the library to Frame 2
fl.getDocumentDOM().getTimeline().layers[0].frames[1].soundLibraryItem
=fl.getDocumentDOM().library.items[1];

//Set the Sound Envelope
fl.getDocumentDOM().getTimeline().layers[0].frames[1].setSoundEnvelope(soundEnv);

for (int i=0; i<soundEnv.length; i++) {
    fl.trace(soundEnv[i].mark);
    fl.trace(soundEnv[i].leftChannel); fl.trace(soundEnv[i].rightChannel);
}
```

### See also

[frame.getSoundEnvelope\(\)](#)

## frame.setSoundEnvelopeLimits()

### Availability

Flash Professional CC.

### Usage

```
frame.setSoundEnvelopeLimits(limits)
```

### Parameters

**limits** A structure that contains `start` and `end` fields that signify the limits for a custom sound envelope.

### Returns

Nothing.

### Description

Method; Sets the sound envelope limits of any frame with a sound file.

### Example

The following example illustrates the use of `setSoundEnvelopeLimits`:

```
/var limits;  
limits.start = 2000; limits.end = 15000;  
fl.getDocumentDOM().getTimeline().layers[0].frames[0].setSoundEnvelopeLimits(limits);
```

### See also

[frame.getSoundEnvelopeLimits\(\)](#)

## frame.shapeTweenBlend

### Availability

Flash MX 2004.

### Usage

```
frame.shapeTweenBlend
```

### Description

Property; a string that specifies how a shape tween is blended between the shape in the keyframe at the start of the tween and the shape in the following keyframe. Acceptable values are "distributive" and "angular".

## frame.soundEffect

### Availability

Flash MX 2004.

**Usage**

```
frame.soundEffect
```

**Description**

Property; a string that specifies effects for a sound that is attached directly to a frame ([frame.soundLibraryItem](#)). Acceptable values are "none", "left channel", "right channel", "fade left to right", "fade right to left", "fade in", "fade out", and "custom".

**Example**

The following example specifies that the sound attached to the first frame should fade in:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].soundEffect = "fade in";
```

## frame.soundLibraryItem

**Availability**

Flash MX 2004.

**Usage**

```
frame.soundLibraryItem
```

**Description**

Property; a library item (see [SoundItem object](#)) used to create a sound. The sound is attached directly to the frame.

**Example**

The following example assigns the first item in the library to the `soundLibraryItem` property of the first frame:

```
// The first item in the library must be a sound object.  
f1.getDocumentDOM().getTimeline().layers[0].frames[0].soundLibraryItem  
=f1.getDocumentDOM().library.items[0];
```

## frame.soundLoop

**Availability**

Flash MX 2004.

**Usage**

```
frame.soundLoop
```

**Description**

Property; an integer value that specifies the number of times a sound that is attached directly to a frame ([frame.soundLibraryItem](#)) plays. If you want to specify a value for this property, set [frame.soundLoopMode](#) to "repeat".

**Example**

See [frame.soundLoopMode](#).

## frame.soundLoopMode

### Availability

Flash MX 2004.

### Usage

```
frame.soundLoopMode
```

### Description

Property; a string that specifies whether a sound that is attached directly to a frame ([frame.soundLibraryItem](#)) should play a specific number of times or loop indefinitely. Acceptable values are "repeat" and "loop". To specify the number of times the sound should play, set a value for [frame.soundLoop](#).

### Example

The following example specifies that a sound should play two times:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].soundLoopMode = "repeat";
f1.getDocumentDOM().getTimeline().layers[0].frames[0].soundLoop = 2;
```

## frame.soundName

### Availability

Flash MX 2004.

### Usage

```
frame.soundName
```

### Description

Property; a string that specifies the name of a sound that is attached directly to a frame ([frame.soundLibraryItem](#)), as stored in the library.

### Example

The following example changes the `soundName` property of the first frame to "song1.mp3"; song1.mp3 must exist in the library:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].soundName = "song1.mp3";
```

## frame.soundSync

### Availability

Flash MX 2004.

### Usage

```
frame.soundSync
```

**Description**

Property; a string that specifies the sync behavior of a sound that is attached directly to a frame ([frame.soundLibraryItem](#)). Acceptable values are "event", "stop", "start", and "stream".

**Example**

The following example specifies that a sound should stream:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].soundSync = 'stream';
```

## frame.startFrame

**Availability**

Flash MX 2004.

**Usage**

```
frame.startFrame
```

**Description**

Read-only property; the index of the first frame in a sequence.

**Example**

In the following example, `stFrame` is the index of the first frame in the frame sequence. In this example, a frame sequence is spanning the six frames from Frame 5 to Frame 10. Therefore, the value of `stFrame` at any frame between Frame 5 and Frame 10 is 4 (remember that index values are different from frame number values).

```
var stFrame = f1.getDocumentDOM().getTimeline().layers[0].frames[4].startFrame;
fl.trace(stFrame); // 4
var stFrame = f1.getDocumentDOM().getTimeline().layers[0].frames[9].startFrame;
fl.trace(stFrame); // 4
```

## frame.TweenEasing

**Availability**

Flash MX 2004.

**Usage**

```
frame.TweenEasing
```

**Description**

Property; an integer that specifies the amount of easing that should be applied to the tweened object. Acceptable values are -100 to 100. To begin the motion tween slowly and accelerate the tween toward the end of the animation, use a value between -1 and -100. To begin the motion tween rapidly and decelerate the tween toward the end of the animation, use a positive value between 1 and 100.

**Example**

The following example specifies that the motion of the tweened object should begin fairly rapidly and decelerate toward the end of the animation:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].tweenEasing = 50;
```

## frame.TweenInstanceName

**Availability**

Flash Professional CS5.

**Usage**

```
Frame.TweenInstanceName()
```

**Description**

Property; a string that assigns an instance name to the selected motion object.

**Example**

The following example assigns the instance name `MyMotionTween` to the specified motion object.

```
theFrame.TweenInstanceName = "MyMotionTween";
```

## frame.TweenType

**Availability**

Flash MX 2004.

**Usage**

```
frame.TweenType
```

**Description**

Property; a string that specifies the type of tween; acceptable values are "motion", "shape", or "none". The value "none" removes the motion tween. Use the `timeline.createMotionTween()` method to create a motion tween.

If you specify "motion", the object in the frame must be a symbol, text field, or grouped object. It will be tweened from its location in the current keyframe to the location in the following keyframe.

If you specify "shape", the object in the frame must be a shape. It will blend from its shape in the current keyframe to the shape in the following keyframe.

**Example**

The following example specifies that the object is a motion tween, and therefore, it should be tweened from its location in the current keyframe to the location in the following keyframe:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].tweenType = "motion";
```

## frame.useSingleEaseCurve

### Availability

Flash 8.

### Usage

```
frame.useSingleEaseCurve
```

### Description

Property; a Boolean value. If `true`, a single custom ease curve is used for easing information for all properties. If `false`, each property has its own ease curve.

This property is ignored if the frame doesn't have custom easing applied.

### Example

The following example specifies that a single custom ease curve should be used for all properties of the first frame on the first layer:

```
var theFrame = fl.getDocumentDOM().getTimeline().layers[0].frames[0]
theFrame.useSingleEaseCurve = true;
```

### See also

[frame.getCustomEase\(\)](#), [frame.hasCustomEase](#), [frame.setCustomEase\(\)](#)

# Chapter 22: HalfEdge object

## halfEdge summary

### Availability

Flash MX 2004.

### Description

The HalfEdge object is the directed side of the edge of a [Shape object](#). An edge has two half edges. You can transverse the contours of a shape by “walking around” these half edges. For example, starting from a half edge, you can trace all the half edges around a contour of a shape, and return to the original half edge.

Half edges are ordered. One half edge represents one side of the edge; the other half edge represents the other side.

### Method summary

The following methods are available for the HalfEdge object:

Method	Description
<code>halfEdge.getEdge()</code>	Gets the <a href="#">Edge object</a> for the HalfEdge object.
<code>halfEdge.getNext()</code>	Gets the next half edge on the current contour.
<code>halfEdge.getOppositeHalfEdge()</code>	Gets the HalfEdge object on the other side of the edge.
<code>halfEdge.getPrev()</code>	Gets the preceding HalfEdge object on the current contour.
<code>halfEdge.getVertex()</code>	Gets the <a href="#">Vertex object</a> at the head of the HalfEdge object.

### Property summary

The following properties are available for the HalfEdge object:

Property	Description
<code>halfEdge.id</code>	Read-only; a unique integer identifier for the HalfEdge object.

## halfEdge.getEdge()

### Availability

Flash MX 2004.

### Usage

```
halfEdge.getEdge()
```

### Parameters

None.

**Returns**

An [Edge object](#).

**Description**

Method; gets the Edge object for the HalfEdge object. See [Edge object](#).

**Example**

The following example illustrates getting an edge and a half edge for the specified shape:

```
var shape = fl.getDocumentDOM().selection[0];
var hEdge = shape.edges[0].getHalfEdge(0);
var edge = hEdge.getEdge();
```

## halfEdge.getNext()

**Availability**

Flash MX 2004.

**Usage**

```
halfEdge.getNext()
```

**Parameters**

None.

**Returns**

A HalfEdge object.

**Description**

Method; gets the next half edge on the current contour.

**Note:** Although half edges have a direction and a sequence order, edges do not.

**Example**

The following example stores the next half edge of the specified contour in the `nextHalfEdge` variable:

```
var shape = fl.getDocumentDOM().selection[0];
var hEdge = shape.edges[0].getHalfEdge( 0 );
var nextHalfEdge = hEdge.getNext();
```

## halfEdge.getOppositeHalfEdge()

**Availability**

Flash MX 2004.

**Usage**

```
halfEdge.getOppositeHalfEdge()
```

**Parameters**

None.

**Returns**

A HalfEdge object.

**Description**

Method; gets the HalfEdge object on the other side of the edge.

**Example**

The following example stores the half edge opposite `hEdge` in the `otherHalfEdge` variable:

```
var shape = fl.getDocumentDOM().selection[0];
var hEdge = shape.edges[0].getHalfEdge(0);
var otherHalfEdge = hEdge.getOppositeHalfEdge();
```

## halfEdge.getPrev()

**Availability**

Flash MX 2004.

**Usage**

```
halfEdge.getPrev()
```

**Parameters**

None.

**Returns**

A HalfEdge object.

**Description**

Method; gets the preceding HalfEdge object on the current contour.

**Note:** Although half edges have a direction and a sequence order, edges do not.

**Example**

The following example stores the previous half edge of the specified contour in the `prevHalfEdge` variable:

```
var shape = fl.getDocumentDOM().selection[0];
var hEdge = shape.edges[0].getHalfEdge( 0 );
var prevHalfEdge = hEdge.getPrev();
```

## halfEdge.getVertex()

**Availability**

Flash MX 2004.

**Usage**

```
halfEdge.getVertex()
```

**Parameters**

None.

**Returns**

A [Vertex object](#)

**Description**

Method; gets the Vertex object at the head of the HalfEdge object. See [Vertex object](#)

**Example**

The following example stores the Vertex object at the head of hEdge in the vertex variable:

```
var shape = fl.getDocumentDOM().selection[0];
var edge = shape.edges[0];
var hEdge = edge.getHalfEdge(0);
var vertex = hEdge.getVertex();
```

## halfEdge.id

**Availability**

Flash MX 2004.

**Usage**

```
halfEdge.id
```

**Description**

Read-only property; a unique integer identifier for the HalfEdge object.

**Example**

The following example displays a unique identifier for the specified half edge in the Output panel:

```
var shape = fl.getDocumentDOM().selection[0];
alert(shape.contours[0].getHalfEdge().id);
```

# Chapter 23: Instance object

## instance summary

**Inheritance** [Element object](#) > Instance object

### Availability

Flash MX 2004.

### Description

Instance is a subclass of the [Element object](#).

### Property summary

In addition to all of the Element object properties, Instance has the following properties:

Property	Description
<code>instance.instanceType</code>	Read-only; a string that represents the type of instance.
<code>instance.libraryItem</code>	Library item used to instantiate this instance.

## instance.instanceType

### Availability

Flash MX 2004; possible value of "video" added in Flash 8.

### Usage

```
instance.instanceType
```

### Description

Read-only property; a string that represents the type of instance. Possible values are "symbol", "bitmap", "embedded video", "linked video", "video", and "compiled clip".

In Flash MX 2004, the value of `instance.instanceType` for an item added to the library using `library.addNewItem("video")` is "embedded\_video". In Flash 8 and later, the value is "video". See [library.addNewItem\(\)](#).

### Example

The following example shows that the instance type of a movie clip is symbol:

```
// Select a movie clip and then run this script.
var type = fl.getDocumentDOM().selection[0].instanceType;
fl.trace("This instance type is " + type);
```

## instance.libraryItem

### Availability

Flash MX 2004.

### Usage

```
instance.libraryItem
```

### Description

Property; a library item used to instantiate this instance. You can change this property only to another library item of the same type (that is, you cannot set a `symbol` instance to refer to a bitmap). See [library object](#).

### Example

The following example changes the selected symbol to refer to the first item in the library:

```
f1.getDocumentDOM().selection[0].libraryItem = f1.getDocumentDOM().library.items[0];
```

# Chapter 24: Item object

## item summary

### Availability

Flash MX 2004.

### Description

The Item object is an abstract base class. Anything in the library derives from Item. See also [library object](#).

### Method summary

The following methods are available for the Item object:

Method	Description
<code>item.addData()</code>	Adds specified data to a library item.
<code>item.getData()</code>	Retrieves the value of the specified data.
<code>item.getPublishData()</code>	Indicates whether publishing of the specified persistent data is enabled for the specified format on a specified library item.
<code>item.hasData()</code>	Determines whether the library item has the named data.
<code>item.removeData()</code>	Removes persistent data from the library item.
<code>item.setPublishData()</code>	Enables publishing of persistent data for a library item.

### Property summary

The following properties are available for the Item object:

Property	Description
<code>item.itemType</code>	Read-only; a string that specifies the type of element.
<code>item.linkageBaseClass</code>	A string that specifies the ActionScript 3.0 class that will be associated with the symbol.
<code>item.linkageClassName</code>	A string that specifies the ActionScript 2.0 class that will be associated with the symbol.
<code>item.linkageExportForAS</code>	A Boolean value. If <code>true</code> , the item is exported for ActionScript.
<code>item.linkageExportForRS</code>	A Boolean value. If <code>true</code> , the item is exported for run-time sharing.
<code>item.linkageExportInFirstFrame</code>	A Boolean value. If <code>true</code> , the item is exported in the first frame.
<code>item.linkageIdentifier</code>	A string that specifies the name Flash will use to identify the asset when linking to the destination SWF file.
<code>item.linkageImportForRS</code>	A Boolean value. If <code>true</code> , the item is imported for run-time sharing.
<code>item.linkageURL</code>	A string that specifies the URL where the SWF file containing the shared asset is located.
<code>item.name</code>	A string that specifies the name of the library item, which includes the folder structure.

## item.addData()

### Availability

Flash MX 2004.

### Usage

```
item.addData(name, type, data)
```

### Parameters

**name** A string that specifies the name of the data.

**type** A string that specifies the type of data. Valid types are "integer", "integerArray", "double", "doubleArray", "string", and "byteArray".

**data** The data to add to the specified library item. The type of data depends on the value of the type parameter. For example, if type is "integer", the value of data must be an integer, and so on.

### Returns

Nothing.

### Description

Method; adds specified data to a library item.

### Example

The following example adds data named `myData` with an integer value of 12 to the first item in the library:

```
f1.getDocumentDOM().library.items[0].addData("myData", "integer", 12);
```

## item.getData()

### Availability

Flash MX 2004.

### Usage

```
item.getData(name)
```

### Parameters

**name** A string that specifies the name of the data to retrieve.

### Returns

The data specified by the `name` parameter. The type of data returned depends on the type of stored data.

### Description

Method; retrieves the value of the specified data.

**Example**

The following example gets the value of the data named `myData` from the first item in the library and stores it in the variable `libData`:

```
var libData = fl.getDocumentDOM().library.items[0].getData("myData");
```

## item.getPublishData()

**Availability**

Flash Professional CC.

**Usage**

```
library.getPublishData(name, format)
```

**Parameters**

**name** A string that contains the name of the persistent data item, as specified in “[item.addData \(\)](#)” on page 345.

**format** A string that specifies the publishing format.

**Note:** `_EMBED_SWF` is a special built-in publishing format for persistent data. If set, the persistent data is embedded in the SWF file every time a document is published. The persistent data can then be accessed via ActionScript with the `.metaData` property. This feature applies to SWF version 19 (Flash Player 11.6) and above and only for symbol instances onstage. Other custom publishing formats may be specified for custom JSFL scripts if `getPublishPersistentData ()` is called with the same format.

**Returns**

A Boolean value that indicates whether publishing of the specified persistent data is enabled for the specified format on this library item.

**Description**

Method; Indicates whether publishing of the specified persistent data is enabled for the specified format on a specified library item.

**Example**

The following example illustrates use of this method:

```
var doc = fl.getDocumentDOM();
// example setting library data
if (doc) {
    var libItem = doc.library.items[0];
    if (libItem) {
        libItem.addData("sampleData", "string", "Hello! I am persistent Data.");
        libItem.setPublishData("sampleData", "_EXTERN_JSON_", true);
        // enable JSON publishing for this document
        doc.setPublishDocumentData("_EXTERN_JSON_", true);
    }
}
// example getting instance data
if (doc && doc.getPublishDocumentData("_EXTERN_JSON_")) {
    var libItem = doc.library.items[0];
    if (libItem) {
        if (libItem.hasData("sampleData") && libItem.getPublishData("sampleData",
"_EXTERN_JSON_")) {
            alert("publish persistent data for libElem: sampleData = '" +
libItem.getData("sampleData") + "'");
        }
    }
}
```

**See also**

[item.setPublishData\(\)](#)

## item.hasData()

**Availability**

Flash MX 2004.

**Usage**

`item.hasData(name)`

**Parameters**

**name** A string that specifies the name of the data to check for in the library item.

**Returns**

A Boolean value: `true` if the specified data exists; `false` otherwise.

**Description**

Method; determines whether the library item has the named data.

**Example**

The following example shows a message in the Output panel if the first item in the library contains data named `myData`:

```
if (fl.getDocumentDOM().library.items[0].hasData("myData")){
    fl.trace("Yep, it's there!");
}
```

## item.itemType

### Availability

Flash MX 2004.

### Usage

```
item.itemType
```

### Description

Read-only property; a string that specifies the type of element. The value is one of the following: "undefined", "component", "movie clip", "graphic", "button", "folder", "font", "sound", "bitmap", "compiled clip", "screen", or "video". If this property is "video", you can determine the type of video; see [videoItem.videoType](#).

### Example

The following example shows the type of the specified library item in the Output panel:

```
f1.trace(f1.getDocumentDOM().library.items[0].itemType);
```

## item.linkageBaseClass

### Availability

Flash CS3 Professional.

### Usage

```
item.linkageBaseClass
```

### Description

Property; a string that specifies the ActionScript 3.0 class that will be associated with the symbol. The value specified here appears in the Linkage dialog box in the authoring environment, and in other dialog boxes that include the Linkage dialog box controls, such as the Symbol Properties dialog box. (To specify this value for an ActionScript 2.0 class, use [item.linkageClassName](#).)

If the base class is the default for the symbol type (for example, "flash.display.MovieClip" for movie clips, "flash.display.SimpleButton" for buttons, and so on), this property is an empty string (""). Similarly, to make an item the default base class, set this value to an empty string.

When you set this value, none of the checks performed by the Linkage dialog box are performed, and no errors are thrown if Flash is unable to set the base class to the specified value. For example, setting this value in the Linkage dialog box forces checks to make sure that the base class can be found in the FLA file's classpath. It ensures that ActionScript 3.0 is chosen in the Flash tab of the Publish Settings dialog box, and so on. These checks are not performed when you set this property in a script.

### Example

The following lines of code show a few ways to use this property:

```
// sets the library item base class to "Sprite"  
fl.getDocumentDOM().library.items[0].linkageBaseClass = "flash.display.Sprite";  
// sets the library item base class to the default for that item type  
fl.getDocumentDOM().library.items[0].linkageBaseClass = "";  
// finds and displays the library item's base class  
fl.trace(fl.getDocumentDOM().library.items[0].linkageBaseClass);
```

**See also**

[document.docClass](#)

## item.linkageClassName

**Availability**

Flash MX 2004.

**Usage**

`item.linkageClassName`

**Description**

Property; a string that specifies the ActionScript 2.0 class that will be associated with the symbol. (To specify this value for an ActionScript 3.0 class, use [item.linkageBaseClass](#).)

For this property to be defined, the [item.linkageExportForAS](#) and/or [item.linkageExportForRS](#) properties must be set to `true`, and the [item.linkageImportForRS](#) property must be set to `false`.

**Example**

The following example specifies that the ActionScript 2.0 class name associated with the first item in the library is `myClass`:

```
fl.getDocumentDOM().library.items[0].linkageClassName = "myClass";
```

## item.linkageExportForAS

**Availability**

Flash MX 2004.

**Usage**

`item.linkageExportForAS`

**Description**

Property; a Boolean value. If this property is `true`, the item is exported for ActionScript. You can also set the [item.linkageExportForRS](#) and [item.linkageExportInFirstFrame](#) properties to `true`.

If you set this property to `true`, the [item.linkageImportForRS](#) property must be set to `false`. Also, you must specify an identifier ([item.linkageIdentifier](#)) and a URL ([item.linkageURL](#)).

**Example**

The following example sets this property for the specified library item:

```
f1.getDocumentDOM().library.items[0].linkageExportForAS = true;
```

## item.linkageExportForRS

**Availability**

Flash MX 2004.

**Usage**

```
item.linkageExportForRS
```

**Description**

Property; a Boolean value. If this property is `true`, the item is exported for run-time sharing. You can also set the `item.linkageExportForAS` and `item.linkageExportInFirstFrame` properties to `true`.

If you set this property to `true`, the `item.linkageImportForRS` property must be set to `false`. Also, you must specify an identifier (`item.linkageIdentifier`) and a URL (`item.linkageURL`).

**Example**

The following example sets this property for the specified library item:

```
f1.getDocumentDOM().library.items[0].linkageExportForRS = true;
```

## item.linkageExportInFirstFrame

**Availability**

Flash MX 2004.

**Usage**

```
item.linkageExportInFirstFrame
```

**Description**

Property; a Boolean value. If `true`, the item is exported in the first frame; if `false`, the item is exported in the frame of the first instance. If the item does not appear on the Stage, it isn't exported.

This property can be set to `true` only when `item.linkageExportForAS` and/or `item.linkageExportForRS` are set to `true`.

**Example**

The following example specifies that the specified library item is exported in the first frame:

```
f1.getDocumentDOM().library.items[0].linkageExportInFirstFrame = true;
```

## item.linkageIdentifier

### Availability

Flash MX 2004.

### Usage

```
item.linkageIdentifier
```

### Description

Property; a string that specifies the name Flash will use to identify the asset when linking to the destination SWF file. Flash ignores this property if [item.linkageImportForRS](#), [item.linkageExportForAS](#), and [item.linkageExportForRS](#) are set to `false`. Conversely, this property must be set when any of those properties are set to `true`.

### Example

The following example specifies that the string `my_mc` will be used to identify the library item when it is linked to the destination SWF file to which it is being exported:

```
f1.getDocumentDOM().library.items[0].linkageIdentifier = "my_mc";
```

### See also

[item.linkageURL](#)

## item.linkageImportForRS

### Availability

Flash MX 2004.

### Usage

```
item.linkageImportForRS
```

### Description

Property; a Boolean value: if `true`, the item is imported for run-time sharing. If this property is set to `true`, both [item.linkageExportForAS](#) and [item.linkageExportForRS](#) must be set to `false`. Also, you must specify an identifier ([item.linkageIdentifier](#)) and a URL ([item.linkageURL](#)).

### Example

The following example sets this property to `true` for the specified library item:

```
f1.getDocumentDOM().library.items[0].linkageImportForRS = true;
```

## item.linkageURL

### Availability

Flash MX 2004.

**Usage**

```
item.linkageURL
```

**Description**

Property; a string that specifies the URL where the SWF file containing the shared asset is located. Flash ignores this property if `item.linkageImportForRS`, `item.linkageExportForAS`, and `item.linkageExportForRS` are set to `false`. Conversely, this property must be set when any of those properties are set to `true`. You can specify a web URL or a filename in platform-dependent format (that is, forward slashes [/] or backward slashes [\], depending on the platform).

**Example**

The following example specifies a linkage URL for the specified library item:

```
f1.getDocumentDOM().library.items[0].linkageURL = "theShareSWF.swf";
```

**See also**

[item.linkageIdentifier](#)

## item.name

**Availability**

Flash MX 2004.

**Usage**

```
item.name
```

**Description**

Method; a string that specifies the name of the library item, which includes the folder structure. For example, if Symbol\_1 is inside a folder called Folder\_1, the `name` property of Symbol\_1 is "Folder\_1/Symbol\_1".

**Example**

The following example shows the name of the specified library item in the Output panel:

```
f1.trace(f1.getDocumentDOM().library.items[0].name);
```

## item.removeData()

**Availability**

Flash MX 2004.

**Usage**

```
item.removeData(name)
```

**Parameters**

`name` Specifies the name of the data to remove from the library item.

**Returns**

Nothing.

**Description**

Property; removes persistent data from the library item.

**Example**

The following example removes the data named `myData` from the first item in the library:

```
fl.getDocumentDOM().library.items[0].removeData("myData");
```

## item.setPublishData()

**Availability**

Flash Professional CC.

**Usage**

```
library.setPublishData(name, format, publish)
```

**Parameters**

**name** A string that contains the name of the persistent data item, as specified in “[item.addData\(\)](#)” on page 345.

**format** A string that specifies the publishing format.

**Note:** *\_EMBED\_SWF\_* is a special built-in publishing format for persistent data. If set, the persistent data is embedded in the SWF file every time a document is published. The persistent data can then be accessed via ActionScript with the `.metaData` property. This feature applies to SWF version 19 (Flash Player 11.6) and above and only for symbol instances onstage. Other custom publishing formats may be specified for custom JSFL scripts if `getPublishPersistentData()` is called with the same format.

**publish** A Boolean that indicates whether to enable or disable publishing of persistent data for the specified format.

**Returns**

None.

**Description**

Method; Enables publishing of persistent data for a library item.

**Example**

The following example illustrates use of this method:

```
var doc = fl.getDocumentDOM();
// example setting library data
if (doc) {
    var libItem = doc.library.items[0];
    if (libItem) {
        libItem.addData("sampleData", "string", "Hello! I am persistent Data.");
        libItem.setPublishData("sampleData", "_EXTERN_JSON_", true);
        // enable JSON publishing for this document
        doc.setPublishDocumentData("_EXTERN_JSON_", true);
    }
}
// example getting instance data
if (doc && doc.getPublishDocumentData("_EXTERN_JSON_")) {
    var libItem = doc.library.items[0];
    if (libItem) {
        if (libItem.hasData("sampleData") && libItem.getPublishData("sampleData",
"_EXTERN_JSON_")) {
            alert("publish persistent data for libElem: sampleData = '" +
libItem.getData("sampleData") + "'");
        }
    }
}
```

**See also**

[item.getPublishData\(\)](#)

# Chapter 25: Layer object

## layer summary

### Availability

Flash MX 2004.

### Description

The Layer object represents a layer in the timeline. The `timeline.layers` property contains an array of Layer objects, which can be accessed by `f1.getDocumentDOM().getTimeline().layers`.

### Property summary

The following properties are available for the Layer object:

Property	Description
<code>layer.animationType</code>	The layer type: "none", "motion object", or "IK pose".
<code>layer.color</code>	A string, hexadecimal value, or integer that specifies the color assigned to outline the layer.
<code>layer.frameCount</code>	Read-only; an integer that specifies the number of frames in the layer.
<code>layer.frames</code>	Read-only; an array of Frame objects.
<code>layer.height</code>	An integer that specifies the percentage layer height; equivalent to the Layer height value in the Layer Properties dialog box.
<code>layer.layerType</code>	A string that specifies the current use of the layer; equivalent to the Type setting in the Layer Properties dialog box.
<code>layer.locked</code>	A Boolean value that specifies the locked status of the layer.
<code>layer.name</code>	A string that specifies the name of the layer.
<code>layer.outline</code>	A Boolean value that specifies the status of outlines for all objects in the layer.
<code>layer.parentLayer</code>	A Layer object that represents the layer's containing folder, guiding, or masking layer.
<code>layer.visible</code>	A Boolean value that specifies whether the layer's objects on the Stage are shown or hidden.

## layer.animationType

### Availability

Flash Pro CS6.

### Usage

`layer.animationType`

**Description**

Read-only property; a string value indicating the animation type of the layer. Possible values include: "none", "motion object", "IK pose".

**Example**

The following example returns the layer type of the first layer of the root timeline:

```
var layer = fl.getDocumentDOM().getTimeline().layers[0];
fl.trace("animationType: " + layer.animationType);
```

## layer.color

**Availability**

Flash MX 2004.

**Usage**

```
layer.color
```

**Description**

Property; the color assigned to outline the layer, in one of the following formats:

- A string in the format "#RRGGBB" or "#RRGGBBAA"
- A hexadecimal number in the format 0xRRGGBB
- An integer that represents the decimal equivalent of a hexadecimal number

This property is equivalent to the Outline color setting in the Layer Properties dialog box.

**Example**

The following example stores the value of the first layer in the `colorValue` variable:

```
var colorValue = fl.getDocumentDOM().getTimeline().layers[0].color;
```

The following example shows three ways to set the color of the first layer to red:

```
fl.getDocumentDOM().getTimeline().layers[0].color=16711680;
fl.getDocumentDOM().getTimeline().layers[0].color="#ff0000";
fl.getDocumentDOM().getTimeline().layers[0].color=0xFF0000;
```

## layer.frameCount

**Availability**

Flash MX 2004.

**Usage**

```
layer.frameCount
```

**Description**

Read-only property; an integer that specifies the number of frames in the layer.

**Example**

The following example stores the number of frames in the first layer in the fcNum variable:

```
var fcNum = fl.getDocumentDOM().getTimeline().layers[0].frameCount;
```

## layer.frames

**Availability**

Flash MX 2004.

**Usage**

```
layer.frames
```

**Description**

Read-only property; an array of Frame objects (see [Frame object](#)).

**Example**

The following example sets the variable frameArray to the array of Frame objects for the frames in the current document:

```
var frameArray = fl.getDocumentDOM().getTimeline().layers[0].frames;
```

To determine if a frame is a keyframe, check whether the `frame.startFrame` property matches the array index, as shown in the following example:

```
var frameArray = fl.getDocumentDOM().getTimeline().layers[0].frames;
var n = frameArray.length;
for (i=0; i<n; i++) {
    if (i==frameArray[i].startFrame) {
        alert("Keyframe at: " + i);
    }
}
```

## layer.height

**Availability**

Flash MX 2004.

**Usage**

```
layer.height
```

**Description**

Property; an integer that specifies the percentage layer height; equivalent to the Layer height value in the Layer Properties dialog box. Acceptable values represent percentages of the default height: 100, 200, or 300.

**Example**

The following example stores the percentage value of the first layer's height setting:

```
var layerHeight = fl.getDocumentDOM().getTimeline().layers[0].height;
```

The following example sets the height of the first layer to 300 percent:

```
f1.getDocumentDOM().getTimeline().layers[0].height = 300;
```

## layer.layerType

### Availability

Flash MX 2004.

### Usage

```
layer.layerType
```

### Description

Property; a string that specifies the current use of the layer; equivalent to the Type setting in the Layer Properties dialog box. Acceptable values are "normal", "guide", "guided", "mask", "masked", and "folder".

### Example

The following example sets the first layer in the timeline to type folder:

```
f1.getDocumentDOM().getTimeline().layers[0].layerType = "folder";
```

## layer.locked

### Availability

Flash MX 2004.

### Usage

```
layer.locked
```

### Description

Property; a Boolean value that specifies the locked status of the layer. If set to `true`, the layer is locked. The default value is `false`.

### Example

The following example stores the Boolean value for the status of the first layer in the `lockStatus` variable:

```
var lockStatus = f1.getDocumentDOM().getTimeline().layers[0].locked;
```

The following example sets the status of the first layer to unlocked:

```
f1.getDocumentDOM().getTimeline().layers[0].locked = false;
```

## layer.name

### Availability

Flash MX 2004.

**Usage**

```
layer.name
```

**Description**

Property; a string that specifies the name of the layer.

**Example**

The following example sets the name of the first layer in the current document to `foreground`:

```
f1.getDocumentDOM().getTimeline().layers[0].name = "foreground";
```

## layer.outline

**Availability**

Flash MX 2004.

**Usage**

```
layer.outline
```

**Description**

Property; a Boolean value that specifies the status of outlines for all objects in the layer. If set to `true`, all objects in the layer appear only with outlines. If `false`, objects appear as they were created.

**Example**

The following example makes all objects on the first layer appear only with outlines:

```
f1.getDocumentDOM().getTimeline().layers[0].outline = true;
```

## layer.parentLayer

**Availability**

Flash MX 2004.

**Usage**

```
layer.parentLayer
```

**Description**

Property; a Layer object that represents the layer's containing folder, guiding, or masking layer. The parent layer must be a folder, guide, or mask layer that precedes the layer, or the `parentLayer` of the preceding or following layer. Setting the layer's `parentLayer` does not move the layer's position in the list; trying to set a layer's `parentLayer` to a layer that would require moving it has no effect. Uses `null` for a top-level layer.

**Example**

The following example uses two layers at the same level on the same timeline. The first layer (`layers[0]`) is converted into a folder and then set as the parent folder of the second layer (`layers[1]`). This action moves the second layer inside the first layer.

```
var parLayer = fl.getDocumentDOM().getTimeline().layers[0];
parLayer.layerType = "folder";
fl.getDocumentDOM().getTimeline().layers[1].parentLayer = parLayer;
```

## layer.visible

### Availability

Flash MX 2004.

### Usage

```
layer.visible
```

### Description

Property; a Boolean value that specifies whether the layer's objects on the Stage are shown or hidden. If set to `true`, all objects in the layer are visible; if `false`, they are hidden. The default value is `true`.

### Example

The following example makes all objects in the first layer invisible:

```
fl.getDocumentDOM().getTimeline().layers[0].visible = false;
```

# Chapter 26: library object

## library summary

### Availability

Flash MX 2004.

### Description

The library object represents the Library panel. It is a property of the Document object (see `document.library`) and can be accessed by `f1.getDocumentDOM().library`.

The library object contains an array of items of different types, including symbols, bitmaps, sounds, and video.

### Method summary

The following methods are available for the library object:

Method	Description
<code>library.addItemToDocument()</code>	Adds the current or specified item to the Stage at the specified position.
<code>library.addNewItem()</code>	Creates a new item of the specified type in the Library panel and sets the new item to the currently selected item.
<code>library.deleteItem()</code>	Deletes the current items or a specified item from the Library panel.
<code>library.duplicateItem()</code>	Makes a copy of the currently selected or specified item.
<code>library.editItem()</code>	Opens the currently selected or specified item in Edit mode.
<code>library.findIndex()</code>	Returns the library item's index value (zero-based).
<code>library.getItemProperty()</code>	Gets the property for the selected item.
<code>library.getItemType()</code>	Gets the type of object currently selected or specified by a library path.
<code>library.getSelectedItems()</code>	Gets the array of all currently selected items in the library.
<code>library.itemExists()</code>	Checks to see if a specified item exists in the library.
<code>library.moveToFolder()</code>	Moves the currently selected or specified library item to a specified folder.
<code>library.newFolder()</code>	Creates a new folder with the specified name, or a default name ("untitled folder #") if no folderName parameter is provided, in the currently selected folder.
<code>library.renameItem()</code>	Renames the currently selected library item in the Library panel.
<code>library.selectAll()</code>	Selects or deselects all items in the library.
<code>library.selectItem()</code>	Selects a specified library item.
<code>library.selectNone()</code>	Deselects all the library items.
<code>library.setItemProperty()</code>	Sets the property for all selected library items (ignoring folders).
<code>library.updateItem()</code>	Updates the specified item.

**Property summary for the library object**

The following property is available for the library object:

Property	Description
<code>library.items</code>	An array of Item objects in the library
<code>library.unusedItems</code>	An array of library items that are not used in the document.

## library.addItemToDocument()

**Availability**

Flash MX 2004.

**Usage**

```
library.addItemToDocument(position [, namePath])
```

**Parameters**

**position** A point that specifies the *x,y* position of the center of the item on the Stage.

**namePath** A string that specifies the name of the item. If the item is in a folder, you can specify its name and path using slash notation. If *namePath* is not specified, the current library selection is used. This parameter is optional.

**Returns**

A Boolean value: `true` if the item is successfully added to the document; `false` otherwise.

**Description**

Method; adds the current or specified item to the Stage at the specified position.

**Example**

The following example adds the currently selected item to the Stage at the (3, 60) position:

```
f1.getDocumentDOM().library.addItemToDocument({x:3, y:60});
```

The following example adds the item `symbol1` located in `folder1` of the library to the Stage at the (550, 485) position:

```
f1.getDocumentDOM().library.addItemToDocument({x:550.0, y:485.0}, "folder1/Symbol1");
```

## library.addNewItem()

**Availability**

Flash MX 2004.

**Usage**

```
library.addNewItem(type [, namePath])
```

**Parameters**

**type** A string that specifies the type of item to create. The only acceptable values for *type* are "video", "movie clip", "button", "graphic", "bitmap", "screen", and "folder" (so, for example, you cannot add a sound to the library with this method). Specifying a folder path is the same as using `library.newFolder()` before calling this method.

**namePath** A string that specifies the name of the item to be added. If the item is in a folder, specify its name and path using slash notation. This parameter is optional.

**Returns**

A Boolean value: `true` if the item is successfully created; `false` otherwise.

**Description**

Method; creates a new item of the specified type in the Library panel and sets the new item to the currently selected item. For more information on importing items into the library, including items such as sounds, see `document.importFile()`.

**Example**

The following example creates a new button item named `start` in a new folder named `folderTwo`:

```
f1.getDocumentDOM().library.addItem("button", "folderTwo/start");
```

## library.deleteItem()

**Availability**

Flash MX 2004.

**Usage**

```
library.deleteItem([namePath])
```

**Parameters**

**namePath** A string that specifies the name of the item to be deleted. If the item is in a folder, you can specify its name and path using slash notation. If you pass a folder name, the folder and all its items are deleted. If no name is specified, Flash deletes the currently selected item or items. To delete all the items in the Library panel, select all items before using this method. This parameter is optional.

**Returns**

A Boolean value: `true` if the items are successfully deleted; `false` otherwise.

**Description**

Method; deletes the current items or a specified item from the Library panel. This method can affect multiple items if several are selected.

**Example**

The following example deletes the currently selected item:

```
f1.getDocumentDOM().library.deleteItem();
```

The following example deletes the item `symbol_1` from the library folder `Folder_1`:

```
f1.getDocumentDOM().library.deleteItem("Folder_1/Symbol_1");
```

## library.duplicateItem()

### Availability

Flash MX 2004.

### Usage

```
library.duplicateItem( [ namePath ] )
```

### Parameters

**namePath** A string that specifies the name of the item to duplicate. If the item is in a folder, you can specify its name and path using slash notation. This parameter is optional.

### Returns

A Boolean value: `true` if the item is duplicated successfully; `false` otherwise. If more than one item is selected, Flash returns `false`.

### Description

Method; makes a copy of the currently selected or specified item. The new item has a default name (such as `item copy`) and is set as the currently selected item. If more than one item is selected, the command fails.

### Example

The following example creates a copy of the item `square` in the library folder `test`:

```
f1.getDocumentDOM().library.duplicateItem("test/square");
```

## library.editItem()

### Availability

Flash MX 2004.

### Usage

```
library.editItem( [namePath] )
```

### Parameters

**namePath** A string that specifies the name of the item. If the item is in a folder, you can specify its name and path using slash notation. If `namePath` is not specified, the single selected library item opens in Edit mode. If none or more than one item in the library is currently selected, the first scene in the main timeline appears for editing. This parameter is optional.

### Returns

A Boolean value: `true` if the specified item exists and can be edited; `false` otherwise.

**Description**

Method; opens the currently selected or specified item in Edit mode.

**Example**

The following example opens the item `circle` in the `test` folder of the library for editing:

```
f1.getDocumentDOM().library.editItem("test/circle");
```

## library.findItemIndex()

**Availability**

Flash MX 2004.

**Usage**

```
library.findItemIndex(namePath)
```

**Parameters**

**namePath** A string that specifies the name of the item. If the item is in a folder, you can specify its name and path using slash notation.

**Returns**

An integer value representing the item's zero-based index value.

**Description**

Method; returns the library item's index value (zero-based). The library index is flat, so folders are considered part of the main index. Folder paths can be used to specify a nested item.

**Example**

The following example stores the zero-based index value of the library item `square`, which is in the `test` folder, in the variable `sqIndex`, and then displays the index value in a dialog box:

```
var sqIndex = f1.getDocumentDOM().library.findItemIndex("test/square");
alert(sqIndex);
```

## library.getItemProperty()

**Availability**

Flash MX 2004.

**Usage**

```
library.getItemProperty(property)
```

**Parameters**

**property** A string. For a list of values that you can use as a *property* parameter, see the Property summary table for the [Item object](#), along with property summaries for its subclasses.

**Returns**

A string value for the property.

**Description**

Method; gets the property for the selected item.

**Example**

The following example shows a dialog box that contains the Linkage Identifier value for the symbol when referencing it using ActionScript or for run-time sharing:

```
alert(f1.getDocumentDOM().library.getItemProperty("linkageIdentifier"));
```

## library.getSelectedItemType()

**Availability**

Flash MX 2004.

**Usage**

```
library.getSelectedItemType( [namePath] )
```

**Parameters**

**namePath** A string that specifies the name of the item. If the item is in a folder, specify its name and path using slash notation. If *namePath* is not specified, Flash provides the type of the current selection. If more than one item is currently selected and no *namePath* is provided, Flash ignores the command. This parameter is optional.

**Returns**

A string value specifying the type of object. For possible return values, see [item.itemType](#).

**Description**

Method; gets the type of object currently selected or specified by a library path.

**Example**

The following example shows a dialog box that contains the item type of *Symbol\_1* located in the *Folder\_1/Folder\_2* folder:

```
alert(f1.getDocumentDOM().library.getSelectedItemType("Folder_1/Folder_2/Symbol_1"));
```

## library.getSelectedItems()

**Availability**

Flash MX 2004.

**Parameters**

None.

**Returns**

An array of values for all currently selected items in the library.

**Description**

Method; gets the array of all currently selected items in the library.

**Example**

The following example stores the array of currently selected library items (in this case, several audio files) in the `selItems` variable and then changes the `sampleRate` property of the first audio file in the array to 11 kHz:

```
var selItems = fl.getDocumentDOM().library.getSelectedItems();
selItems[0].sampleRate = "11 kHz";
```

## library.itemExists()

**Availability**

Flash MX 2004.

**Usage**

```
library.itemExists(namePath)
```

**Parameters**

`namePath` A string that specifies the name of the item. If the item is in a folder, specify its name and path using slash notation.

**Returns**

A Boolean value: `true` if the specified item exists in the library; `false` otherwise.

**Description**

Method; checks to see if a specified item exists in the library.

**Example**

The following example displays `true` or `false` in a dialog box, depending on whether the item `Symbol_1` exists in the `Folder_1` library folder:

```
alert(f1.getDocumentDOM().library.itemExists('Folder_1/Symbol_1'));
```

## library.items

**Availability**

Flash MX 2004.

**Usage**

```
library.items
```

**Description**

Property; an array of item objects in the library.

**Example**

The following example stores the array of all library items in the `itemArray` variable:

```
var itemArray = fl.getDocumentDOM().library.items;
```

## library.moveToFolder()

**Availability**

Flash MX 2004.

**Usage**

```
library.moveToFolder(folderPath [, itemToMove [, bReplace]])
```

**Parameters**

**folderPath** A string that specifies the path to the folder in the form "FolderName" or "FolderName/FolderName". To move an item to the top level, specify an empty string ("") for `folderPath`.

**itemToMove** A string that specifies the name of the item to move. If `itemToMove` is not specified, the currently selected items move. This parameter is optional.

**bReplace** A Boolean value. If an item with the same name already exists, specifying `true` for the `bReplace` parameter replaces the existing item with the item being moved. If `false`, the name of the dropped item changes to a unique name. The default value is `false`. This parameter is optional.

**Returns**

A Boolean value: `true` if the item moves successfully; `false` otherwise.

**Description**

Method; moves the currently selected or specified library item to a specified folder. If the `folderPath` parameter is empty, the items move to the top level.

**Example**

The following example moves the item `Symbol_1` to the library folder `new` and replaces the item in that folder with the same name:

```
fl.getDocumentDOM().library.moveToFolder("new", "Symbol_1", true);
```

## library.newFolder()

**Availability**

Flash MX 2004.

**Usage**

```
library.newFolder([FolderPath])
```

**Parameters**

**FolderPath** A string that specifies the name of the folder to be created. If it is specified as a path, and the path doesn't exist, the path is created. This parameter is optional.

**Returns**

A Boolean value: `true` if folder is created successfully; `false` otherwise.

**Description**

Method; creates a new folder with the specified name, or a default name ("untitled folder #") if no *folderName* parameter is provided, in the currently selected folder.

**Example**

The following example creates two new library folders. The second folder is a subfolder of the first folder:

```
f1.getDocumentDOM().library.newFolder("first/second");
```

## library.renameItem()

**Availability**

Flash MX 2004.

**Usage**

```
library.renameItem(name)
```

**Parameters**

**name** A string that specifies a new name for the library item.

**Returns**

A Boolean value of `true` if the name of the item changes successfully, `false` otherwise. If multiple items are selected, no names are changed, and the return value is `false` (to match user interface behavior).

**Description**

Method; renames the currently selected library item in the Library panel.

**Example**

The following example renames the currently selected library item to `new_name`:

```
f1.getDocumentDOM().library.renameItem("new_name");
```

## library.selectAll()

**Availability**

Flash MX 2004.

**Usage**

```
library.selectAll([bSelectAll])
```

**Parameters**

**bSelectAll** A Boolean value that specifies whether to select or deselect all items in the library. Omit this parameter or use the default value of `true` to select all the items in the library; `false` deselects all library items. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; selects or deselects all items in the library.

**Example**

The following examples select all the items in the library:

```
f1.getDocumentDOM().library.selectAll();  
f1.getDocumentDOM().library.selectAll(true);
```

The following examples deselect all the items in the library:

```
f1.getDocumentDOM().library.selectAll(false);  
f1.getDocumentDOM().library.selectNone();
```

## library.selectItem()

**Availability**

Flash MX 2004.

**Usage**

```
library.selectItem(namePath [, bReplaceCurrentSelection [, bSelect]])
```

**Parameters**

**namePath** A string that specifies the name of the item. If the item is in a folder, you can specify its name and path using slash notation.

**bReplaceCurrentSelection** A Boolean value that specifies whether to replace the current selection or add the item to the current selection. The default value is `true` (replace current selection). This parameter is optional.

**bSelect** A Boolean value that specifies whether to select or deselect an item. The default value is `true` (select). This parameter is optional.

**Returns**

A Boolean value: `true` if the specified item exists; `false` otherwise.

**Description**

Method; selects a specified library item.

**Example**

The following example changes the current selection in the library to `Symbol_1` inside `untitled Folder_1`:

```
f1.getDocumentDOM().library.selectItem("untitled Folder_1/Symbol_1");
```

The following example extends what is currently selected in the library to include `Symbol_1` inside `untitled Folder_1`:

```
f1.getDocumentDOM().library.selectItem("untitled Folder_1/Symbol_1", false);
```

The following example deselects `Symbol_1` inside `untitled Folder_1` and does not change other selected items:

```
f1.getDocumentDOM().library.selectItem("untitled Folder_1/Symbol_1", true, false);
```

## library.selectNone()

**Availability**

Flash MX 2004.

**Usage**

```
library.selectNone()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; deselects all the library items.

**Example**

The following examples deselect all the items in the library:

```
f1.getDocumentDOM().library.selectNone();  
f1.getDocumentDOM().library.selectAll(false);
```

## library.setItemProperty()

**Availability**

Flash MX 2004.

**Usage**

```
library.setItemProperty(property, value)
```

**Parameters**

**property** A string that is the name of the property to set. For a list of properties, see the Property summary table for the [Item object](#) and property summaries for its subclasses. To see which objects are subclasses of the Item object, see “[Summary of the DOM structure](#)” on page 14.

**value** The value to assign to the specified property.

**Returns**

Nothing.

**Description**

Method; sets the property for all selected library items (ignoring folders).

**Example**

The following example assigns the value button to the symbolType property for the selected library item or items. In this case, the item must be a [SymbolItem object](#); symbolType is a valid property for SymbolItem objects.

```
f1.getDocumentDOM().library.setItemProperty("symbolType", "button");
```

## library.unusedItems

**Availability**

Flash Professional CC.

**Usage**

```
library.unusedItems
```

**Description**

Property; an array of Library Items that are not used in the document. This is the equivalent of the “Select Unused Items” menu item in the Library panel.

**Example**

The following example illustrates the use of this property:

```
var items = f1.getDocumentDOM().library.unusedItems;
fl.trace("number of unused items found: " + items.length);
for (var i in items)
    fl.trace("'" + items[i].name);
```

## library.updateItem()

**Availability**

Flash MX 2004.

**Usage**

```
library.updateItem( [namePath] )
```

**Parameters**

**namePath** A string that specifies the name of the item. If the item is in a folder, specify its name and path using slash notation. This is the same as right-clicking on an item and selecting Update from the menu in the user interface. If no name is provided, the current selection is updated. This parameter is optional.

**Returns**

A Boolean value: `true` if Flash updated the item successfully; `false` otherwise.

**Description**

Method; updates the specified item.

**Example**

The following example displays a dialog box that shows whether the currently selected item is updated (`true`) or not (`false`):

```
alert(f1.getDocumentDOM().library.updateItem());
```

# Chapter 27: Math object

## Math summary

### Availability

Flash MX 2004.

### Description

The Math object is available as a read-only property of the flash object; see [f1.Math](#). This object provides methods that perform common mathematical operations.

### Method summary

The following methods are available for the Math object:

Method	Description
<a href="#">Math.concatMatrix()</a>	Performs a matrix concatenation and returns the result.
<a href="#">Math.invertMatrix()</a>	Returns the inverse of the specified matrix.
<a href="#">Math.pointDistance()</a>	Computes the distance between two points.
<a href="#">Math.transformPoint()</a>	Applies a matrix to a point.

## Math.concatMatrix()

### Availability

Flash MX 2004.

### Usage

```
Math.concatMatrix(mat1, mat2)
```

### Parameters

**mat1, mat2** Specify the Matrix objects to be concatenated (see [Matrix object](#)). Each parameter must be an object with fields a, b, c, d, tx, and ty.

### Returns

A concatenated object matrix.

### Description

Method; performs a matrix concatenation and returns the result.

### Example

The following example stores the currently selected object in the `elt` variable, multiplies the object matrix by the view matrix, and stores that value in the `mat` variable:

```
var elt = fl.getDocumentDOM().selection[0];
var mat = fl.Math.concatMatrix( elt.matrix , fl.getDocumentDOM().viewMatrix );
```

## Math.invertMatrix()

### Availability

Flash MX 2004.

### Usage

```
Math.invertMatrix(mat)
```

### Parameters

**mat** Indicates the Matrix object to invert (see [Matrix object](#)). It must have the following fields: `a`, `b`, `c`, `d`, `tx`, and `ty`.

### Returns

A Matrix object that is the inverse of the original matrix.

### Description

Method; returns the inverse of the specified matrix.

### Example

The following example stores the currently selected object in the `elt` variable, assigns that matrix to the `mat` variable, and stores the inverse of the matrix in the `inv` variable:

```
var elt = fl.getDocumentDOM().selection[0];
var mat = elt.matrix;
var inv = fl.Math.invertMatrix( mat );
```

## Math.pointDistance()

### Availability

Flash MX 2004.

### Usage

```
Math.pointDistance(pt1, pt2)
```

### Parameters

**pt1, pt2** Specify the points between which distance is measured.

### Returns

A floating-point value that represents the distance between the points.

### Description

Method; computes the distance between two points.

**Example**

The following example stores the value for the distance between *pt1* and *pt2* in the *dist* variable:

```
var pt1 = {x:10, y:20}
var pt2 = {x:100, y:200}
var dist = fl.Math.pointDistance(pt1, pt2);
```

## Math.transformPoint()

**Availability**

Flash CS6.

**Usage**

```
Math.transformPoint(matrix, point)
```

**Parameters**

**matrix** Contains the matrix obejct applied to the point.

**point** Contains the point to which the matrix is applied.

**Returns**

The transformed point.

**Description**

Method; applies a matrix to a point.

**Example**

The following example gets a matrix from the first object in Frame 1, creates a point with x:100 and y:200, and transforms this point using the matrix in the first line:

```
var mat = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].matrix;
var point = {x:100, y:200};
var retPoint = fl.Math.transformPoint(mat, point);
```

# Chapter 28: Matrix object

## matrix summary

### Availability

Flash MX 2004.

### Description

The Matrix object represents a transformation matrix.

### Property summary

The following properties are available for the Matrix object:

Property	Description
<code>matrix.a</code>	A floating-point value that specifies the (0,0) element in the transformation matrix.
<code>matrix.b</code>	A floating-point value that specifies the (0,1) element in the matrix.
<code>matrix.c</code>	A floating-point value that specifies the (1,0) element in the matrix.
<code>matrix.d</code>	A floating-point value that specifies the (1,1) element in the matrix.
<code>matrix.tx</code>	A floating-point value that specifies the x-axis location of a symbol's registration point or the center of a shape.
<code>matrix.ty</code>	A floating-point value that specifies the y-axis location of a symbol's registration point or the center of a shape.

## matrix.a

### Availability

Flash MX 2004.

### Usage

```
matrix.a
```

### Description

Property; a floating-point value that specifies the (0,0) element in the transformation matrix. This value represents the scale factor of the object's *x*-axis.

### Example

The `a` and `d` properties in a matrix represent scaling. In the following example, the values are set to 2 and 3, respectively, to scale the selected object to two times its width and three times its height:

```
var mat = fl.getDocumentDOM().selection[0].matrix;
mat.a = 2;
mat.d = 3;
fl.getDocumentDOM().selection[0].matrix = mat;
```

You can rotate an object by setting the `a`, `b`, `c`, and `d` matrix properties relative to one another, where `a = d` and `b = -c`. For example, values of 0.5, 0.8, -0.8, and 0.5 rotate the object 60°:

```
var mat = fl.getDocumentDOM().selection[0].matrix;
mat.a = 0.5;
mat.b = 0.8;
mat.c = 0.8*(-1);
mat.d = 0.5;
fl.getDocumentDOM().selection[0].matrix = mat;
```

You can set `a = d = 1` and `c = b = 0` to reset the object back to its original shape.

## matrix.b

### Availability

Flash MX 2004.

### Usage

```
matrix.b
```

### Description

Property; a floating-point value that specifies the (0,1) element in the matrix. This value represents the vertical skew of a shape; it causes Flash to move the shape's right edge along the vertical axis.

The `matrix.b` and `matrix.c` properties in a matrix represent skewing (see [matrix.c](#)).

### Example

In the following example, you can set `b` and `c` to -1 and 0, respectively; these settings skew the object at a 45° vertical angle:

```
var mat = fl.getDocumentDOM().selection[0].matrix;
mat.b = -1;
mat.c = 0;
fl.getDocumentDOM().selection[0].matrix = mat;
```

To skew the object back to its original shape, you can set `b` and `c` to 0.

See also the [matrix.a](#) example.

## matrix.c

### Availability

Flash MX 2004.

### Usage

```
matrix.c
```

**Description**

Property; a floating-point value that specifies the (1,0) element in the matrix. This value causes Flash to skew the object by moving its bottom edge along a horizontal axis.

The `matrix.b` and `matrix.c` properties in a matrix represent skewing.

**Example**

See the [matrix.b](#) example.

## matrix.d

**Availability**

Flash MX 2004.

**Usage**

`matrix.d`

**Description**

Property; a floating-point value that specifies the (1,1) element in the matrix. This value represents the scale factor of the object's *y*-axis.

**Example**

See the [matrix.a](#) example.

## matrix.tx

**Availability**

Flash MX 2004.

**Usage**

`matrix.tx`

**Description**

Property; a floating-point value that specifies the *x*-axis location of a symbol's registration point (also *origin point* or *zero point*) or the center of a shape. It defines the *x* translation of the transformation.

You can move an object by setting the `matrix.tx` and `matrix.ty` properties (see [matrix.ty](#)).

**Example**

In the following example, setting `tx` and `ty` to 0 moves the registration point of the object to point 0,0 in the document:

```
var mat = fl.getDocumentDOM().selection[0].matrix;
mat.tx = 0;
mat.ty = 0;
fl.getDocumentDOM().selection[0].matrix = mat;
```

## matrix.ty

### Availability

Flash MX 2004.

### Usage

`matrix.ty`

### Description

Property; a floating-point value that specifies the *y*-axis location of a symbol's registration point or the center of a shape. It defines the *y* translation of the transformation.

You can move an object by setting the `matrix.tx` and `matrix.ty` properties.

### Example

See the [matrix.tx](#) example.

# Chapter 29: outputPanel object

## outputPanel summary

### Availability

Flash MX 2004.

### Description

This object represents the Output panel, which displays troubleshooting information such as syntax errors. To access this object, use `f1.outputPanel` (or `flash.outputPanel`). See [f1.outputPanel](#).

### Method summary

The outputPanel object uses the following methods:

Method	Description
<code>outputPanel.clear()</code>	Clears the contents of the Output panel.
<code>outputPanel.save()</code>	Saves the contents of the Output panel to a local text file.
<code>outputPanel.trace()</code>	Adds a line to the contents of the Output panel, terminated by a new line.

## outputPanel.clear()

### Availability

Flash MX 2004.

### Usage

```
outputPanel.clear()
```

### Parameters

None.

### Returns

Nothing.

### Description

Method; clears the contents of the Output panel. You can use this method in a batch processing application to clear a list of errors, or to save them incrementally by using this method with `outputPanel.save()`.

### Example

The following example clears the current contents of the Output panel:

```
f1.outputPanel.clear();
```

## outputPanel.save()

### Availability

Flash MX 2004; *bUseSystemEncoding* parameter added in Flash 8.

### Usage

```
outputPanel.save(fileURI [, bAppendToFile [, bUseSystemEncoding]])
```

### Parameters

**fileURI** A string, expressed as a file:/// URI, that specifies the local file to contain the contents of the Output panel.

**bAppendToFile** An optional Boolean value. If `true`, it appends the Output panel's contents to the output file, and if `false`, the method overwrites the output file if it already exists. The default value is `false`.

**bUseSystemEncoding** An optional Boolean value. If `true`, it saves the Output panel text using the system encoding; if `false`, it saves the Output panel text using UTF-8 encoding, with Byte Order Mark characters at the beginning of the text. The default value is `false`.

### Returns

Nothing.

### Description

Method; saves the contents of the Output panel to a local text file, either by overwriting the file or by appending to the file.

If *fileURI* is invalid or unspecified, an error is reported.

This method is useful for batch processing. For example, you can create a JSFL file that compiles several components. Any compile errors appear in the Output panel, and you can use this method to save the resulting errors to a text file, which can be automatically parsed by the build system in use.

### Example

The following example saves the Output panel's contents to the batch.log file in the /tests folder, overwriting the batch.log file if it already exists:

```
f1.outputPanel.save("file:///c|/tests/batch.log");
```

## outputPanel.trace()

### Availability

Flash MX 2004.

### Usage

```
outputPanel.trace(message)
```

### Parameters

**message** A string that contains the text to add to the Output panel.

**Returns**

Nothing.

**Description**

Method; sends a text string to the Output panel, terminated by a new line, and displays the Output panel if it is not already visible. This method is identical to [f1.trace\(\)](#), and works in the same way as the `trace()` statement in ActionScript.

To send a blank line, use `outputPanel.trace("")` or `outputPanel.trace("\n")`. You can use the latter command inline, making `\n` a part of the *message* string.

**Example**

The following example displays several lines of text in the Output panel:

```
f1.outputPanel.clear();
f1.outputPanel.trace("Hello World!!!");
var myPet = "cat";
f1.outputPanel.trace("\nI have a " + myPet);
f1.outputPanel.trace("");
f1.outputPanel.trace("I love my " + myPet);
f1.outputPanel.trace("Do you have a " + myPet +"?");
```

# Chapter 30: Oval object

## OvalObject summary

**Inheritance** [Element object](#) > [Shape object](#) > Oval object

### Availability

Flash CS3 Professional.

### Description

The Oval object is a shape that is drawn using the Oval Primitive tool. To determine if an item is an Oval object, use [shape.is OvalObject](#).

### Property summary

In addition to the [Shape object](#) properties, you can use the following properties with the Oval object. To set the properties of an Oval object, use [document.setOvalObjectProperty\(\)](#).

Property	Description
<a href="#">OvalObject.closePath</a>	Read-only; a Boolean value that specifies whether the Close Path check box in the Property inspector is selected.
<a href="#">OvalObject.endAngle</a>	Read-only; a float value that specifies the end angle of the Oval object.
<a href="#">OvalObject.innerRadius</a>	Read-only; a float value that specifies the inner radius of the Oval object as a percentage.
<a href="#">OvalObject.startAngle</a>	Read-only; a float value that specifies the start angle of the Oval object.

## OvalObject.closePath

### Availability

Flash CS3 Professional.

### Usage

`OvalObject.closePath`

### Description

Read-only property; a Boolean value that specifies whether the Close Path check box in the Property inspector is selected. If the start angle and end angle values for the object are the same, setting this property has no effect until the values change.

To set this value, use [document.setOvalObjectProperty\(\)](#).

### Example

The following example deselects the `OvalObject.closePath` property:

```
f1.getDocumentDOM().setOvalObjectProperty("closePath", false);
```

**See also**

[document.setOvalObjectProperty\(\)](#), [shape.isOvalObject](#)

## OvalObject.endAngle

**Availability**

Flash CS3 Professional.

**Usage**

OvalObject.endAngle

**Description**

Read-only property; a float value that specifies the end angle of the Oval object. Acceptable values are from 0 to 360.

To set this value, use [document.setOvalObjectProperty\(\)](#).

**Example**

The following example sets the end angle of selected Oval objects to 270.

```
f1.getDocumentDOM().setOvalObjectProperty("endAngle", 270);
```

**See also**

[document.setOvalObjectProperty\(\)](#), [OvalObject.startAngle](#), [shape.isOvalObject](#)

## OvalObject.innerRadius

**Availability**

Flash CS3 Professional.

**Usage**

OvalObject.innerRadius

**Description**

Read-only property; a float value that specifies the inner radius of the Oval object as a percentage. Acceptable values are from 0 to 99.

To set this value, use [document.setOvalObjectProperty\(\)](#).

**Example**

The following example sets the inner radius of selected Oval objects to 50 percent:

```
f1.getDocumentDOM().setOvalObjectProperty("innerRadius", 50);
```

**See also**

[document.setOvalObjectProperty\(\)](#), [shape.isOvalObject](#)

## OvalObject.startAngle

### Availability

Flash CS3 Professional.

### Usage

`OvalObject.startAngle`

### Description

Read-only property; a float value that specifies the start angle of the Oval object. Acceptable values are from 0 to 360.

To set this value, use [document.setOvalObjectProperty\(\)](#).

### Example

The following example sets the start angle of selected Oval objects to 270:

```
f1.getDocumentDOM().setOvalObjectProperty("startAngle", 270);
```

### See also

[document.setOvalObjectProperty\(\)](#), [OvalObject.endAngle](#), [shape.isOvalObject](#)

# Chapter 31: Parameter object

## parameter summary

### Availability

Flash MX 2004.

### Description

The Parameter object type is accessed from the `componentInstance.parameters` array (which corresponds to the component Property inspector in the authoring tool).

### Method summary

The following methods are available for the Parameter object:

Method	Description
<code>parameter.insertItem()</code>	Inserts an item into an object or array.
<code>parameter.removeItem()</code>	Removes an element of the object or array type of a screen or component parameter.

### Property summary

The following properties are available for the Parameter object:

Property	Description
<code>parameter.category</code>	A string that specifies the <code>category</code> property for the <code>screen</code> parameter or <code>componentInstance</code> parameter.
<code>parameter.listIndex</code>	An integer that specifies the value of the selected list item.
<code>parameter.name</code>	Read-only; a string that specifies the name of the parameter.
<code>parameter.value</code>	Corresponds to the Value field in the Parameters tab of the Component inspector, the Parameters tab of the Property inspector, or the screen Property inspector.
<code>parameter.valueType</code>	Read-only; a string that indicates the type of the screen or component parameter.
<code>parameter.verbose</code>	Specifies where the parameter is displayed.

## parameter.category

### Availability

Flash MX 2004.

### Usage

`parameter.category`

**Description**

Property; a string that specifies the `category` property for the `screen` parameter or `componentInstance` parameter. This property provides an alternative way of presenting a list of parameters. This functionality is not available through the Flash user interface.

## **parameter.insertItem()**

**Availability**

Flash MX 2004.

**Usage**

```
parameter.insertItem(index, name, value, type)
```

**Parameters**

**index** A zero-based integer index that indicates where the item will be inserted in the object or array. If the index is 0, the item is inserted at the beginning. If the index is greater than the list size, the new item is inserted at the end.

**name** A string that specifies the name of the item to insert. This is a required parameter for object parameters.

**value** A string that specifies the value of the item to insert.

**type** A string that specifies the type of item to insert.

**Returns**

Nothing.

**Description**

Method; inserts an item in an object or array. If a parameter is an object or array, the `value` property is an array.

**Example**

The following example inserts the value of `New Value` into the `labelPlacement` parameter:

```
// Select an instance of a Button component on the Stage.  
var parms = fl.getDocumentDOM().selection[0].parameters;  
parms[3].insertItem(0, "name", "New Value", "String");  
var values = parms[3].value;  
for(var prop in values){  
    fl.trace("labelPlacement parameter value = " + values[prop].value);  
}
```

## **parameter.listIndex**

**Availability**

Flash MX 2004.

**Usage**

```
parameter.listIndex
```

**Description**

Property; the value of the selected list item. This property is valid only if `parameter.valueType` is "List".

**Example**

The following example sets the first parameter for a Slide, which is the `autoKeyNav` parameter. To set the parameter to one of its acceptable values (true, false, or inherit) `parameter.listIndex` is set to the index of the item in the list (0 for true, 1 for false, 2 for inherit).

```
var parms = fl.getDocumentDOM().screenOutline.screens[1].parameters;  
parms[0].listIndex = 1;
```

## parameter.name

**Availability**

Flash MX 2004.

**Usage**

```
parameter.name
```

**Description**

Read-only property; a string that specifies the name of the parameter.

**Example**

The following example shows the name of the fifth parameter for the selected component:

```
var parms = fl.getDocumentDOM().selection[0].parameters;  
fl.trace("name: " + parms[4].name);
```

The following example shows the name of the fifth parameter for the specified screen:

```
var parms = fl.getDocumentDOM().screenOutline.screens[1].parameters; fl.trace("name: " +  
parms[4].name);
```

## parameter.removeItem()

**Availability**

Flash MX 2004.

**Usage**

```
parameter.removeItem(index)
```

**Parameters**

**index** The zero-based integer index of the item to be removed from the screen or component property.

**Returns**

Nothing.

**Description**

Method; removes an element of the object or array type of a screen or component parameter.

**Example**

The following example removes the element at index 1 from the `labelPlacement` parameter of a component:

```
// Select an instance of a Button component on the Stage.  
var parms = fl.getDocumentDOM().selection[0].parameters;  
var values = parms[2].value;  
fl.trace("--Original--");  
for(var prop in values){  
    fl.trace("labelPlacement value = " + values[prop].value);  
}  
parms[2].removeItem(1);  
  
var newValues = parms[2].value;  
fl.trace("--After Removing Item--");  
for(var prop in newValues){  
    fl.trace("labelPlacement value = " + newValues[prop].value);  
}
```

The following example removes the element at index 1 from the `autoKeyNav` parameter of a screen:

```
// Open a presentation document.  
var parms = fl.getDocumentDOM().screenOutline.screens[1].parameters;  
var values = parms[0].value;  
fl.trace("--Original--");  
for(var prop in values){  
    fl.trace("autoKeyNav value = " + values[prop].value);  
}  
parms[0].removeItem(1);  
  
var newValues = parms[0].value;  
fl.trace("--After Removing Item--");  
for(var prop in newValues){  
    fl.trace("autoKeyNav value = " + newValues[prop].value);  
}
```

## parameter.value

**Availability**

Flash MX 2004.

**Usage**

`parameter.value`

**Description**

Property; corresponds to the Value field in the Parameters tab of the Component inspector, the Parameters tab of the Property inspector, or the screen Property inspector. The type of the `value` property is determined by the `valueType` property for the parameter (see [parameter.valueType](#)).

## parameter.valueType

### Availability

Flash MX 2004.

### Usage

```
parameter.valueType
```

### Description

Read-only property; a string that indicates the type of the screen or component parameter. The type can be one of the following values: "Default", "Array", "Object", "List", "String", "Number", "Boolean", "Font Name", "Color", "Collection", "Web Service URL", or "Web Service Operation".

### See also

[parameter.value](#)

## parameter.verbose

### Availability

Flash MX 2004.

### Usage

```
parameter.verbose
```

### Description

Property; specifies where the parameter is displayed. If the value of this property is 0 (nonverbose), the parameter is displayed only in the Component inspector. If it is 1 (verbose), the parameter is displayed in the Component inspector and in the Parameters tab of the Property inspector.

# Chapter 32: Path object

## path summary

### Availability

Flash MX 2004.

### Description

The Path object defines a sequence of line segments (straight, curved, or both), which you typically use when creating extensible tools. The following example shows an instance of a Path object being returned from the flash object:

```
path = fl.drawingLayer newPath();
```

See also the [drawingLayer object](#).

### Method summary

The following methods are available for the Path object:

Method	Description
<code>path.addCubicCurve()</code>	Appends a cubic Bézier curve segment to the path.
<code>path.addCurve()</code>	Appends a quadratic Bézier segment to the path.
<code>path.addPoint()</code>	Adds a point to the path.
<code>path.clear()</code>	Removes all points from the path.
<code>path.close()</code>	Appends a point at the location of the first point of the path and extends the path to that point, which closes the path.
<code>path.makeShape()</code>	Creates a shape on the Stage by using the current stroke and fill settings.
<code>path.newContour()</code>	Starts a new contour in the path.

### Property summary

The following properties are available for the Path object:

Property	Description
<code>path.nPts</code>	Read-only; an integer representing the number of points in the path.

## path.addCubicCurve()

### Availability

Flash MX 2004.

### Usage

```
path.addCubicCurve(xAnchor, yAnchor, x2, y2, x3, y3, x4, y4)
```

**Parameters**

**xAnchor** A floating-point number that specifies the *x* position of the first control point.

**yAnchor** A floating-point number that specifies the *y* position of the first control point.

**x2** A floating-point number that specifies the *x* position of the second control point.

**y2** A floating-point number that specifies the *y* position of the second control point.

**x3** A floating-point number that specifies the *x* position of the third control point.

**y3** A floating-point number that specifies the *y* position of the third control point.

**x4** A floating-point number that specifies the *x* position of the fourth control point.

**y4** A floating-point number that specifies the *y* position of the fourth control point.

**Returns**

Nothing.

**Description**

Method; appends a cubic Bézier curve segment to the path.

**Example**

The following example creates a new path, stores it in the `myPath` variable, and assigns the curve to the path:

```
var myPath = fl.drawingLayer newPath();
myPath.addCubicCurve(0, 0, 10, 20, 20, 20, 30, 0);
```

## path.addCurve()

**Availability**

Flash MX 2004.

**Usage**

```
path.addCurve(xAnchor, yAnchor, x2, y2, x3, y3)
```

**Parameters**

**xAnchor** A floating-point number that specifies the *x* position of the first control point.

**yAnchor** A floating-point number that specifies the *y* position of the first control point.

**x2** A floating-point number that specifies the *x* position of the second control point.

**y2** A floating-point number that specifies the *y* position of the second control point.

**x3** A floating-point number that specifies the *x* position of the third control point.

**y3** A floating-point number that specifies the *y* position of the third control point.

**Returns**

Nothing.

**Description**

Method; appends a quadratic Bézier segment to the path.

**Example**

The following example creates a new path, stores it in the `myPath` variable, and assigns the curve to the path:

```
var myPath = fl.drawingLayer newPath();
myPath.addCurve(0, 0, 10, 20, 20, 0);
```

## path.addPoint()

**Availability**

Flash MX 2004.

**Usage**

```
path.addPoint(x, y)
```

**Parameters**

**x** A floating-point number that specifies the *x* position of the point.

**y** A floating-point number that specifies the *y* position of the point.

**Returns**

Nothing.

**Description**

Method; adds a point to the path.

**Example**

The following example creates a new path, stores it in the `myPath` variable, and assigns the new point to the path:

```
var myPath = fl.drawingLayer newPath();
myPath.addPoint(10, 100);
```

## path.clear()

**Availability**

Flash MX 2004.

**Usage**

```
path.clear()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; removes all points from the path.

**Example**

The following example removes all points from a path stored in the myPath variable:

```
var myPath = fl.drawingLayer newPath();
myPath.clear();
```

## path.close()

**Availability**

Flash MX 2004.

**Usage**

```
path.close()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; appends a point at the location of the first point of the path and extends the path to that point, which closes the path. If the path has no points, no points are added.

**Example**

The following example creates a closed path:

```
var myPath = fl.drawingLayer newPath();
myPath.close();
```

## path.makeShape()

**Availability**

Flash MX 2004.

**Usage**

```
path.makeShape( [bSuppressFill [, bSuppressStroke]])
```

**Parameters**

**bSuppressFill** A Boolean value that, if set to `true`, suppresses the fill that would be applied to the shape. The default value is `false`. This parameter is optional.

**bSuppressStroke** A Boolean value that, if set to `true`, suppresses the stroke that would be applied to the shape. The default value is `false`. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; creates a shape on the Stage by using the current stroke and fill settings. The path is cleared after the shape is created. This method has two optional parameters for suppressing the fill and stroke of the resulting shape object. If you omit these parameters or set them to `false`, the current values for fill and stroke are used.

**Example**

The following example creates a shape with the current fill and no stroke:

```
var myPath = fl.drawingLayer newPath();
myPath.makeShape(false, true);
```

## path.newContour()

**Availability**

Flash MX 2004.

**Usage**

```
path.newContour()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; starts a new contour in the path.

**Example**

The following example creates a hollow square:

```
var myPath = fl.drawingLayer newPath();
myPath.addPoint(0, 0);
myPath.addPoint(0, 30);
myPath.addPoint(30, 30);
myPath.addPoint(30, 0);
myPath.addPoint(0, 0);

myPath.newContour();
myPath.addPoint(10, 10);
myPath.addPoint(10, 20);
myPath.addPoint(20, 20);
myPath.addPoint(20, 10);
myPath.addPoint(10, 10);

myPath.makeShape();
```

## path.nPts

### Availability

Flash MX 2004.

### Usage

path.nPts

### Description

Read-only property; an integer representing the number of points in the path. A new path has 0 points.

### Example

The following example uses the Output panel to show the number of points in the path referenced by the myPath variable:

```
var myPath = fl.drawingLayer newPath();
var numOfPoints = myPath.nPts;
fl.trace("Number of points in the path: " + numOfPoints);
// Displays: Number of points in the path: 0
```

# Chapter 33: presetItem object

## presetItem summary

### Availability

Flash CS4 Professional.

### Description

The presetItem object represents an item (preset or folder) in the Motion Presets panel (Window > Motion Presets). The array of presetItem objects is a property of the presetPanel object ([presetPanel.items](#)).

All properties of the presetItem object are read only. To perform tasks such as deleting, renaming, or moving items, use the methods of the [presetPanel object](#).

### Property summary

You can use the following properties with the presetItem object:

Property	Description
<a href="#">presetItem.isDefault</a>	Specifies whether the item is installed along with Flash or is a custom item that you or someone else has created.
<a href="#">presetItem.isFolder</a>	Specifies whether the item in the Motion Presets panel is a folder or a preset.
<a href="#">presetItem.level</a>	The level of the item in the folder structure of the Motion Presets panel.
<a href="#">presetItem.name</a>	The name of the preset or folder, without path information.
<a href="#">presetItem.open</a>	Specifies whether a folder in the Motion Presets panel is currently expanded.
<a href="#">presetItem.path</a>	The path to the item in the Motion Presets panel folder tree, and the item name.

## presetItem.isDefault

### Availability

Flash CS4 Professional.

### Usage

```
presetItem.isDefault
```

### Description

Read-only property: a Boolean value that specifies whether the item is installed along with Flash (`true`) or is a custom item that you or someone else has created (`false`). If this value is `true`, you can consider it a “read-only” item; it can’t be moved, deleted, or have any similar operations applied to it.

### Example

The following example displays the contents of the Motion Presets panel and indicates whether an item is installed along with Flash:

```
f1.outputPanel.clear();
var presetItemArray=f1.presetPanel.items;
for (i=0;i<presetItemArray.length; i++){
    var presetItem = presetItemArray[i];
    fl.trace(presetItem.name +", default =" + presetItem.isDefault);
}
```

## **presetItem.isFolder**

### **Availability**

Flash CS4 Professional.

### **Usage**

```
presetItem.isFolder
```

### **Description**

Read-only property: a Boolean value that specifies whether the item in the Motion Presets panel is a folder (`true`) or a preset (`false`).

### **Example**

The following example shows that the first item in the Motion Presets panel is a folder and the second is a preset:

```
var presetItemArray=f1.presetPanel.items;
fl.trace(presetItemArray[0].isFolder);
fl.trace(presetItemArray[1].isFolder);
```

## **presetItem.level**

### **Availability**

Flash CS4 Professional.

### **Usage**

```
presetItem.level
```

### **Description**

Read-only property: an integer that specifies the level of the item in the folder structure of the Motion Presets panel. The Default Folder and Custom Presets folder are level 0.

### **Example**

The following example shows that the first item in the Motion Presets panel is level 0 and the second is level 1:

```
var presetItemArray=f1.presetPanel.items;
fl.trace(presetItemArray[0].level);
fl.trace(presetItemArray[1].level);
```

## presetItem.name

### Availability

Flash CS4 Professional.

### Usage

`presetItem.name`

### Description

Read-only property: a string that represents the name of the preset or folder, without path information.

### Example

See [presetItem.path](#).

## presetItem.open

### Availability

Flash CS4 Professional.

### Usage

`presetItem.open`

### Description

Read-only property: specifies whether a folder in the Motion Presets panel is currently expanded (`true`) or not (`false`).

This property is `true` if the item is not a folder. To determine if an item is a folder or a preset, use [presetItem.isFolder](#).

### Example

The following example displays information on whether folders in the Motion Presets panel are expanded or collapsed:

```
fl.outputPanel.clear();
var presetItemArray=fl.presetPanel.items;
for (i=0;i<presetItemArray.length; i++) {
    var presetItem = presetItemArray[i];
    if (presetItem.isFolder) {
        var status = presetItem.open ? "Open" : "Closed"
        fl.trace(presetItem.level + "-" + presetItem.name +" folder is " + status);
    }
}
```

## presetItem.path

### Availability

Flash CS4 Professional.

**Usage**`presetItem.path`**Description**

Read-only property: a string that represents the path to the item in the Motion Presets panel folder tree, and the item name.

**Example**

The following example illustrates the difference between the values in `presetItem.name` and `presetItem.path`.

```
fl.outputPanel.clear();
var presetItemArray=fl.presetPanel.items;
for (i=0;i<presetItemArray.length; i++){
    var presetItem = presetItemArray[i];
    fl.trace("Name: " + presetItem.name + "\n" + "Path: " + presetItem.path);
    fl.trace("");
}
```

# Chapter 34: presetPanel object

## presetPanel summary

### Availability

Flash CS4 Professional.

### Description

The presetPanel object represents the Motion Presets panel (Window > Motion Presets). It is a property of the flash object ([fl.presetPanel](#)).

### Method summary

You can use the following methods with the presetPanel object:

Method	Description
<code>presetPanel.addNewItem()</code>	If a single motion tween is currently selected on the Stage, adds that motion to the Motion Presets panel.
<code>presetPanel.applyPreset()</code>	Applies the specified or currently selected preset to the currently selected item on the Stage.
<code>presetPanel.deleteFolder()</code>	Deletes the specified folder and any of its subfolders from the folder tree of the Motion Presets panel.
<code>presetPanel.deleteItem()</code>	Deletes the specified preset from the Motion Presets panel.
<code>presetPanel.expandFolder()</code>	Expands or collapses the currently selected folder or folders in the Motion Presets panel.
<code>presetPanel.exportItem()</code>	Exports the currently selected or the specified preset to an XML file.
<code>presetPanel.findIndex()</code>	Returns an integer that represents the index location of an item in the Motion Presets panel.
<code>presetPanel.getSelectedItems()</code>	Returns an array of presetItem objects corresponding to the currently selected items in the Motion Presets panel.
<code>presetPanel.importItem()</code>	Adds a preset to the Motion Presets panel from a specified XML file.
<code>presetPanel.moveToFolder()</code>	Moves the specified item to the specified folder.
<code>presetPanel.newFolder()</code>	Creates a folder in the folder tree of the Motion Presets panel.
<code>presetPanel.renameItem()</code>	Renames the currently selected preset or folder to a specified name.
<code>presetPanel.selectItem()</code>	Selects or deselects an item in the Motion Presets panel.

### Property summary

You can use the following property with the presetPanel object:

Property	Description
<code>presetPanel.items</code>	An array of presetItem objects in the Motion Presets panel.

## presetPanel.addItem()

### Availability

Flash CS4 Professional.

### Usage

```
fl.presetPanel.addItem( [namePath] );
```

### Parameters

**namePath** A string that specifies the path and name of the item to add to the Motion Presets panel. This parameter is optional.

### Returns

A Boolean value of `true` if the item was successfully added; `false` otherwise.

### Description

Method; if a single motion tween is currently selected on the Stage, adds that motion to the Motion Presets panel in the specified folder with the specified name. The path specified in *namePath* must exist in the panel.

If a preset matching *namePath* exists, this method has no effect, and returns `false`.

If you don't pass a value for *namePath*, the item is added to the Custom Presets folder with the name "Custom preset *n*," where *n* is incremented each time you add an item in this fashion.

### Example

Assuming that a single motion tween is selected on the Stage, the following code adds a preset named `Bouncing Ball` to the Custom Presets folder:

```
fl.presetPanel.addItem("Custom Presets/Bouncing Ball");
```

### See also

[presetPanel.newFolder\(\)](#)

## presetPanel.applyPreset()

### Availability

Flash CS4 Professional.

### Usage

```
presetPanel.applyPreset( [presetPath] )
```

### Parameters

**presetPath** A string that specifies the full path and name of the preset to be applied, as it appears in the Motion Presets panel. This parameter is optional; if you don't pass a value, the currently selected preset is applied.

### Returns

A Boolean value of `true` if the preset is successfully applied, `false` otherwise.

**Description**

Method; applies the specified or currently selected preset to the currently selected item on the Stage. The item must be a motion tween, a symbol, or an item that can be converted to a symbol. If the item is a motion tween, its current motion is replaced with the selected preset without requesting user confirmation.

This method fails in the following situations:

- The path you specify as *presetPath* doesn't exist.
- You don't pass a value for *presetPath* and no preset is selected.
- You don't pass a value for *presetPath* and multiple presets are selected.
- The selected item on the Stage is not a symbol and can't be converted to a symbol.

**Example**

The following example applies the aDribble preset to the currently selected item on the Stage:

```
var result = fl.presetPanel.applyPreset("Custom Presets/Bounces/aDribble");
fl.trace(result);
```

## **presetPanel.deleteFolder()**

**Availability**

Flash CS4 Professional.

**Usage**

```
presetPanel.deleteFolder( [folderPath] )
```

**Parameters**

**FolderPath** A string that specifies the folder to delete from the Motion Presets panel. This parameter is optional.

**Returns**

A Boolean value of `true` if the folder or folders are successfully deleted; `false` otherwise.

**Description**

Method; deletes the specified folder and any of its subfolders from the folder tree of the Motion Presets panel. Any presets in the folders are also deleted. You can't delete folders from the Default Presets folder.

If you don't pass a value for *FolderPath*, any folders that are currently selected are deleted.

**Note:** Folders are deleted without requesting user confirmation, and there is no way to undo this action.

**Example**

The following code deletes a folder named Bouncing below the Custom Presets folder; any subfolders of Bouncing are also deleted:

```
fl.presetPanel.deleteFolder("Custom Presets/Bouncing");
```

**See also**

[presetPanel.deleteItem\(\)](#)

## presetPanel.deleteItem()

### Availability

Flash CS4 Professional.

### Usage

```
presetPanel.deleteItem( [namePath] )
```

### Parameters

**namePath** A string that specifies the path and name of the item to delete from the Motion Presets panel. This parameter is optional.

### Returns

A Boolean value of `true` if the item or items are successfully deleted; `false` otherwise.

### Description

Method; deletes the specified preset from the Motion Presets panel. If you don't pass a value for `namePath`, any presets that are currently selected are deleted. You can't delete items from the Default Presets folder.

*Note: Items are deleted without requesting user confirmation, and there is no way to undo this action.*

### Example

The following code deletes a preset named `aDribble` from the Custom Presets folder:

```
f1.presetPanel.deleteItem("Custom Presets/aDribble");
```

### See also

[presetPanel.deleteFolder\(\)](#)

## presetPanel.expandFolder()

### Availability

Flash CS4 Professional.

### Usage

```
presetPanel.expandFolder( [bExpand [, bRecurse [, folderPath] ] ] )
```

### Parameters

**bExpand** A Boolean value that specifies whether to expand the folder (`true`) or collapse it (`false`). This parameter is optional; the default value is `true`.

**bRecurse** A Boolean value that specifies whether to expand or collapse the folder's subfolders (`true`) or not (`false`). This parameter is optional; the default value is `false`.

**folderPath** A string that specifies the path to the folder to expand or collapse. This parameter is optional.

### Returns

A Boolean value of `true` if the folder or folders are successfully expanded or collapsed; `false` otherwise.

**Description**

Method; expands or collapses the currently selected folder or folders in the Motion Presets panel. To expand or collapse folders other than the folders that are currently selected, pass a value for *folderPath*.

**Example**

The following example expands the Custom Presets folder but does not expand its subfolders:

```
f1.presetPanel.expandFolder(true, false, "Custom Presets");
```

The following example expands the Custom Presets folder and all its subfolders:

```
f1.presetPanel.expandFolder(true, true, "Custom Presets");
```

## **presetPanel.exportItem()**

**Availability**

Flash CS4 Professional.

**Usage**

```
presetPanel.exportItem(fileURI [, namePath] )
```

**Parameters**

**fileURI** A string, expressed as a file:/// URI, that specifies the path and optionally a filename for the exported file. See “Description,” below, for more information.

**namePath** A string that specifies the path and name of the item to select from the Motion Presets panel. This parameter is optional.

**Returns**

A Boolean value of `true` if the preset was exported successfully; `false` otherwise.

**Description**

Method; exports the currently selected or the specified preset to an XML file. Only presets can be exported; the method fails if you try to export a folder. This method also fails if you try to overwrite a file on disk.

If you don't specify a filename as part of *fileURI* (that is, if the last character of *fileURI* is a slash (/)), the exported file is saved with the same name as the preset being exported. If you don't specify a value for *namePath*, the currently selected preset is exported. See the example below.

**Example**

The following example demonstrates what files are created when different parameters are passed to this method, and informs you if the specified file was successfully created. Before running this example, select the fly-in-left preset in the Default Presets folder and create the `My Presets` folder on disk.

```
//Exports fly-in-left to C:\My Presets\fly-in-left.xml
fl.presetPanel.exportItem("file:///C:/My Presets/");
//Exports fly-in-left to C:\My Presets\myFavoritePreset.xml
fl.presetPanel.exportItem("file:///C:/My Presets/myFavoritePreset.xml");
// Exports the "pulse" preset to C:\My Presets\pulse.xml
fl.presetPanel.exportItem("file:///C:/My Presets/", "Default Presets/pulse");
// Exports the "pulse" preset to C:\My Presets\thePulsePreset.xml
fl.presetPanel.exportItem("file:///C:/My Presets/thePulsePreset.xml", "Default
Presets/pulse");
```

**See also**

[presetPanel.importItem\(\)](#)

## **presetPanel.findIndex()**

**Availability**

Flash CS4 Professional.

**Usage**

```
presetPanel.findIndex([presetName])
```

**Parameters**

**presetName** A string that specifies the name of the preset for which the index value is returned. This parameter is optional.

**Returns**

An integer that represents the index of the specified preset in the `presetPanel.items` array. If you don't pass a value for `presetName`, the index of the currently specified preset is returned. This method returns -1 in the following situations:

- You don't pass a value for `presetName` and no preset is selected.
- You don't pass a value for `presetName` and multiple presets are selected.
- You pass a value for `presetName` that doesn't correspond to an item in the panel.

**Description**

Method; returns an integer that represents the index location of an item in the Motion Presets panel.

**Example**

The following code displays the index value and full pathname of the currently selected preset:

```
// Select one preset in the Motions Preset panel before running this code
var selectedPreset = fl.presetPanel.findIndex();
fl.trace(selectedPreset);
fl.trace(fl.presetPanel.items[selectedPreset].path);
```

## presetPanel.getSelectedItems()

### Availability

Flash CS4 Professional.

### Usage

```
presetPanel.getSelectedItems()
```

### Parameters

None.

### Returns

An array of presetItem objects.

### Description

Method; returns an array of presetItem objects corresponding to the currently selected items in the Motion Presets panel (see [presetItem object](#)). Each item in the array represents either a folder or a preset.

### Example

The following code displays the full pathnames of the currently selected items in the Motion Presets panel:

```
var itemArray = fl.presetPanel.getSelectedItems();
var length = itemArray.length
for (x=0; x<length; x++) {
    fl.trace(itemArray[x].path);
}
```

### See also

[presetPanel.items](#)

## presetPanel.importItem()

### Availability

Flash CS4 Professional.

### Usage

```
presetPanel.importItem(fileURI [,namePath ])
```

### Parameters

**fileURI** A string, expressed as a file:/// URI, that specifies the XML file to be imported as a preset in the Motion Presets panel.

**namePath** A string that specifies in which folder to place the imported file and what to name it. This parameter is optional.

### Returns

A Boolean value of `true` if the file is successfully imported; `false` otherwise.

**Description**

Method; adds a preset to the Motion Presets panel from a specified XML file. The path specified in *namePath* must exist in the panel.

To create XML files that can be imported, use [presetPanel.exportItem\(\)](#).

If you don't pass a value for *namePath*, the imported preset is placed in the Custom Presets folder and given the same name as the imported file (without the XML extension).

**Example**

The following example imports a preset into the Custom Presets/Pulse folder, and names it `fastPulse`.

```
f1.presetPanel.importItem("file:///C|/My Presets/thePulsePreset.xml", "Custom Presets/Pulse/fastPulse");
```

**See also**

[presetPanel.exportItem\(\)](#)

## **presetPanel.items**

**Availability**

Flash CS4 Professional.

**Usage**

`presetPanel.items`

**Description**

Property; an array of `presetItem` objects in the Motion Presets panel (see [presetItem object](#)). Each item in the array represents either a folder or a preset.

**Example**

The following code displays the full pathnames of the items in the Motion Presets panel:

```
var itemArray = f1.presetPanel.items;
var length = itemArray.length
for (x=0; x<length; x++) {
    f1.trace(itemArray[x].path);
}
```

**See also**

[presetPanel.getSelectedItems\(\)](#)

## **presetPanel.moveToFolder()**

**Availability**

Flash CS4 Professional.

**Usage**

```
presetPanel.moveToFolder(folderPath [, namePath] )
```

**Parameters**

**folderPath** A string that specifies the path to the folder in the Motion Presets panel to which the item or items are moved.

**namePath** A string that specifies the path and name of the item to move. This parameter is optional.

**Returns**

A Boolean value of `true` if the items are successfully moved; `false` otherwise.

**Description**

Method; moves the specified item to the specified folder.

If you pass an empty string ("") for *folderPath*, the items are moved to the Custom Presets folder. If you don't pass a value for *namePath*, the currently selected items are moved.

You can't move items to or from the Default Presets folder.

**Example**

In the following example, the currently selected items are moved to the Custom Presets/Bouncing folder, and then the Fast Bounce preset is moved to the same folder:

```
f1.presetPanel.moveToFolder("Custom Presets/Bouncing");
f1.presetPanel.moveToFolder("Custom Presets/Bouncing" , "Custom Presets/Fast Bounce");
```

## **presetPanel.newFolder()**

**Availability**

Flash CS4 Professional.

**Usage**

```
presetPanel.newFolder( [folderPath] )
```

**Parameters**

**folderPath** A string that specifies where to add a new folder in the Motion Presets panel, and the name of the new folder. This parameter is optional.

**Returns**

A Boolean value of `true` if the folder is successfully added; `false` otherwise.

**Description**

Method; creates a folder in the folder tree of the Motion Presets panel. You can create only one new folder level with this method. That is, if you pass "Custom Presets/My First Folder/My Second Folder" for *folderPath*, "Custom Presets/My First Folder" must exist in the folder tree.

If you don't pass a value for *folderPath*, a folder named "Untitled folder *n*" is created at the first level below "Custom Presets," where *n* is incremented each time a folder is added in this fashion.

**Note:** You can't add folders to the Default Presets folder.

#### Example

The following example adds a folder named `Bouncing` below the Custom Presets folder:

```
fl.presetPanel.newFolder("Custom Presets/Bouncing");
```

#### See also

[presetPanel.addItem\(\)](#)

## presetPanel.renameItem()

#### Availability

Flash CS4 Professional.

#### Usage

```
presetPanel.renameItem(newName)
```

#### Parameters

`newName` A string that specifies the new name for the preset or folder.

#### Returns

A Boolean value of `true` if the preset or folder is successfully renamed; `false` otherwise.

#### Description

Method; renames the currently selected preset or folder to a specified name. This method succeeds only if a single preset or folder in the Custom Presets folder is selected. This method fails in the following situations:

- No item is selected.
- Multiple items are selected.
- The selected item is in the Default Presets folder.
- An item named `newName` exists in the same location as the selected item.

#### Example

The following example renames the currently selected preset in the Custom Presets folder to `Bounce Faster`.

```
var renamed = fl.presetPanel.renameItem("Bounce Faster");
fl.trace(renamed);
```

## presetPanel.selectItem()

#### Availability

Flash CS4 Professional.

**Usage**

```
presetPanel.selectItem(namePath [, bReplaceCurrentSelection [, bSelect] ])
```

**Parameters**

**namePath** A string that specifies the path and name of the item to select from the Motion Presets panel.

**bReplaceCurrentSelection** A Boolean value that specifies whether the specified item replaces any current selection (`true`) or is added to the current selection (`false`). This parameter is optional; the default value is `true`.

**bSelect** A Boolean value that specifies whether to select the item (`true`) or deselect the item (`false`). This parameter is optional; the default value is `true`. If you pass `false` for `bSelect`, the value of `bReplaceCurrentSelection` is ignored.

**Returns**

A Boolean value of `true` if the item was successfully selected or deselected; `false` otherwise.

**Description**

Method; selects or deselects an item in the Motion Presets panel, optionally replacing any items currently selected.

**Example**

The following code adds the `fly-in-blur-right` preset to the currently selected presets (if any) in the Motion Presets panel:

```
f1.presetPanel.selectItem("Default Presets/fly-in-blur-right", false);
```

# Chapter 35: Rectangle object

## RectangleObject summary

**Inheritance** [Element object](#) > [Shape object](#) > Rectangle object

### Availability

Flash CS3 Professional.

### Description

The Rectangle object is a shape that is drawn using the Rectangle Primitive tool. To determine if an item is a Rectangle object, use [shape.isRectangleObject](#).

### Property summary

In addition to the [Shape object](#) properties, you can use the following properties with the Rectangle object. To set the properties of a Rectangle object, use [document.setRectangleObjectProperty\(\)](#).

Property	Description
<a href="#">RectangleObject.bottomLeftRadius</a>	Read-only; a float value that sets the radius of the bottom-left corner of the Rectangle object.
<a href="#">RectangleObject.bottomRightRadius</a>	Read-only; a float value that sets the radius of the bottom-right corner of the Rectangle object.
<a href="#">RectangleObject.lockFlag</a>	Read-only; a Boolean value that determines whether different corners of the rectangle can have different radius values.
<a href="#">RectangleObject.topLeftRadius</a>	Read-only; a float value that sets the radius of all corners of the rectangle or that sets only the radius of the top-left corner of the Rectangle object.
<a href="#">RectangleObject.topRightRadius</a>	Read-only; a float value that sets the radius of the top-right corner of the Rectangle object.

## RectangleObject.bottomLeftRadius

### Availability

Flash CS3 Professional.

### Usage

`RectangleObject.bottomLeftRadius`

### Description

Read-only property; a float value that sets the radius of the bottom-left corner of the Rectangle object. If [RectangleObject.lockFlag](#) is true, trying to set this value has no effect.

To set this value, use [document.setRectangleObjectProperty\(\)](#).

**See also**

`document.setRectangleObjectProperty()`, `RectangleObject.bottomRightRadius`,  
`RectangleObject.lockFlag`, `RectangleObject.topLeftRadius`, `RectangleObject.topRightRadius`

## RectangleObject.bottomRightRadius

**Availability**

Flash CS3 Professional.

**Usage**

`RectangleObject.bottomRightRadius`

**Description**

Read-only property; a float value that sets the radius of the bottom-right corner of the Rectangle object. If `RectangleObject.lockFlag` is true, trying to set this value has no effect.

To set this value, use `document.setRectangleObjectProperty()`.

**See also**

`document.setRectangleObjectProperty()`, `RectangleObject.bottomLeftRadius`,  
`RectangleObject.lockFlag`, `RectangleObject.topLeftRadius`, `RectangleObject.topRightRadius`

## RectangleObject.lockFlag

**Availability**

Flash CS3 Professional.

**Usage**

`RectangleObject.lockFlag`

**Description**

Read-only property; a Boolean value that determines whether different corners of the rectangle can have different radius values. If this value is `true`, all corners have the value assigned to `RectangleObject.topLeftRadius`. If it is `false`, each corner radius can be set independently.

To set this value, use `document.setRectangleObjectProperty()`.

**See also**

`document.setRectangleObjectProperty()`, `RectangleObject.bottomLeftRadius`,  
`RectangleObject.bottomRightRadius`, `RectangleObject.topLeftRadius`,  
`RectangleObject.topRightRadius`

## RectangleObject.topLeftRadius

### Availability

Flash CS3 Professional.

### Usage

```
RectangleObject.topLeftRadius
```

### Description

Read-only property; a float value that sets the radius of all corners of the rectangle (if `RectangleObject.lockFlag` is `true`) or that sets only the radius of the top-left corner (if `RectangleObject.lockFlag` is `false`).

To set this value, use [document.setRectangleObjectProperty\(\)](#).

### See also

[document.setRectangleObjectProperty\(\)](#), `RectangleObject.bottomLeftRadius`,  
`RectangleObject.bottomRightRadius`, `RectangleObject.lockFlag`, `RectangleObject.topRightRadius`

## RectangleObject.topRightRadius

### Availability

Flash CS3 Professional.

### Usage

```
RectangleObject.topRightRadius
```

### Description

Read-only property; a float value that sets the radius of the top-right corner of the Rectangle object. If `RectangleObject.lockFlag` is `true`, trying to set this value has no effect.

To set this value, use [document.setRectangleObjectProperty\(\)](#).

### See also

[document.setRectangleObjectProperty\(\)](#), `RectangleObject.bottomLeftRadius`,  
`RectangleObject.bottomRightRadius`, `RectangleObject.lockFlag`, `RectangleObject.topLeftRadius`

# Chapter 36: Shape object

## shape summary

**Inheritance** [Element object](#) > Shape object

### Availability

Flash MX 2004.

### Description

The Shape object is a subclass of the Element object. The Shape object provides more precise control than the drawing APIs when manipulating or creating geometry on the Stage. This control is necessary so that scripts can create useful effects and other drawing commands (see [Element object](#)).

All Shape methods and properties that change a shape or any of its subordinate parts must be placed between `shape.beginEdit()` and `shape.endEdit()` calls to function correctly.

### Method summary

In addition to the Element object methods, you can use the following methods with the Shape object:

Method	Description
<code>shape.getCubicSegmentPoints()</code>	Returns an array of points that define a cubic curve.
<code>shape.beginEdit()</code>	Defines the start of an edit session.
<code>shape.deleteEdge()</code>	Deletes the specified edge.
<code>shape.endEdit()</code>	Defines the end of an edit session for the shape.

### Property summary

In addition to the Element object properties, the following properties are available for the Shape object:

Property	Description
<code>shape.contours</code>	Read-only; an array of Contour objects for the shape (see <a href="#">Contour object</a> ).
<code>shape.edges</code>	Read-only; an array of Edge objects (see <a href="#">Edge object</a> ).
<code>shape.isDrawingObject</code>	Read-only; if true, the shape is a drawing object.
<code>shape.isFloating</code>	Read-only; if true, the shape is floating above the parent frame's (or group's) shape.
<code>shape.isGroup</code>	Read-only; if true, the shape is a group.
<code>shape.isOvalObject</code>	Read-only; if true, the shape is a primitive Oval object (was created using the Oval tool).
<code>shape.isRectangleObject</code>	Read-only; if true, the shape is a primitive Rectangle object (was created using the Rectangle tool).
<code>shape.members</code>	An array of objects in the currently selected group.
<code>shape.numCubicSegments</code>	Read-only; the number of cubic segments in the shape.
<code>shape.vertices</code>	Read-only; an array of Vertex objects (see <a href="#">Vertex object</a> ).

## shape.beginEdit()

### Availability

Flash MX 2004.

### Usage

```
shape.beginEdit()
```

### Parameters

None.

### Returns

Nothing.

### Description

Method; defines the start of an edit session. You must use this method before issuing any commands that change the Shape object or any of its subordinate parts.

### Example

The following example takes the currently selected shape and removes the first edge in the edge array from it:

```
var shape = fl.getDocumentDOM().selection[0];
shape.beginEdit();
shape.deleteEdge(0);
shape.endEdit();
```

## shape.contours

### Availability

Flash MX 2004.

### Usage

```
shape.contours
```

### Description

Read-only property; an array of Contour objects for the shape (see [Contour object](#)).

### Example

The following example stores the first contour in the contours array in the *c* variable and then stores the [HalfEdge object](#) of that contour in the *he* variable:

```
var c = fl.getDocumentDOM().selection[0].contours[0];
var he = c.getHalfEdge();
```

## shape.deleteEdge()

### Availability

Flash MX 2004.

### Usage

```
shape.deleteEdge(index)
```

### Parameters

**index** A zero-based index that specifies the edge to delete from the `shape.edges` array. This method changes the length of the `shape.edges` array.

### Returns

Nothing.

### Description

Method; deletes the specified edge. You must call `shape.beginEdit()` before using this method.

### Example

The following example takes the currently selected shape and removes the first edge in the edge array:

```
var shape = fl.getDocumentDOM().selection[0];
shape.beginEdit();
shape.deleteEdge(0);
shape.endEdit();
```

## shape.edges

### Availability

Flash MX 2004.

### Usage

```
shape.edges
```

### Description

Read-only property; an array of Edge objects (see [Edge object](#)).

## shape.endEdit()

### Availability

Flash MX 2004.

### Usage

```
shape.endEdit()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; defines the end of an edit session for the shape. All changes made to the Shape object or any of its subordinate parts will be applied to the shape. You must use this method after issuing any commands that change the Shape object or any of its subordinate parts.

**Example**

The following example takes the currently selected shape and removes the first edge in the edge array from it:

```
var shape = fl.getDocumentDOM().selection[0];
shape.beginEdit();
shape.deleteEdge(0);
shape.endEdit();
```

## shape.get~~CubicSegmentPoints()~~

**Availability**

Flash CS4 Professional.

**Usage**

```
shape.getCubicSegmentPoints(cubicSegmentIndex)
```

**Parameters**

**cubicSegmentIndex** An integer that specifies the cubic segment for which points are returned.

**Returns**

An array of points that define a cubic curve for the Edge object that corresponds to the specified *cubicSegmentIndex* (see [edge.cubicSegmentIndex](#)).

**Description**

Method; returns an array of points that define a cubic curve.

**Example**

The following example displays the *x* and *y* values for each point on the cubic curve of the first edge of the selection:

```
var elem = fl.getDocumentDOM().selection[0];
var index = elem.edges[0].cubicSegmentIndex;
var cubicPoints = elem.getCubicSegmentPoints(index);
for (i=0; i<cubicPoints.length; i++) {
    fl.trace("index " + i + " x: " + cubicPoints[i].x + " y: " + cubicPoints[i].y);
}
```

## shape.isDrawingObject

### Availability

Flash 8.

### Usage

```
shape.isDrawingObject
```

### Description

Read-only property; if true, the shape is a drawing object.

### Example

The following example stores the first selected object in the sel variable and then uses the `elementType` and `isDrawingObject` properties to determine if the selected item is a drawing object:

```
var sel = fl.getDocumentDOM().selection[0];
var shapeDrawingObject = (sel.elementType == "shape") && sel.isDrawingObject;
fl.trace(shapeDrawingObject);
```

### See also

[document.crop\(\)](#), [document.deleteEnvelope\(\)](#), [document.intersect\(\)](#), [document.punch\(\)](#),  
[document.union\(\)](#), [shape.isGroup](#)

## shape.isFloating

### Availability

Flash CS6.

### Usage

```
shape.isFloating
```

### Description

Read-only property; if true, the shape is floating above the parent frame's (or group's) shape. Also, if true, this type of shape will have its own matrix, similar to a drawing object.

### Example

The following example displays whether a specified shape is floating:

```
var myShape = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];
fl.trace("is shape floating? " + myShape.isFloating);
```

## shape.isGroup

### Availability

Flash MX 2004.

**Usage**

```
shape.isGroup
```

**Description**

Read-only property; if `true`, the shape is a group. A group can contain different types of elements, such as text elements and symbols. However, the group itself is considered a shape, and you can use the `shape.isGroup` property no matter what types of elements the group contains.

**Example**

The following example stores the first selected object in the `sel` variable and then uses the `elementType` and `shape.isGroup` properties to determine if the selected item is a group:

```
var sel = fl.getDocumentDOM().selection[0];
var shapeGroup = (sel.elementType == "shape") && sel.isGroup;
fl.trace(shapeGroup);
```

**See also**

[shape.isDrawingObject](#)

## shape.isOvalObject

**Availability**

Flash CS3 Professional.

**Usage**

```
shape.isOvalObject
```

**Description**

Read-only property; if `true`, the shape is a primitive Oval object (was created using the Oval Primitive tool).

**Example**

The following example displays "`true`" if the first selected item is a primitive Oval object, and "`false`" if it is not:

```
var sel = fl.getDocumentDOM().selection[0];
fl.trace(sel.isOvalObject);
```

**See also**

[shape.isRectangleObject](#)

## shape.isRectangleObject

**Availability**

Flash CS3 Professional.

**Usage**

```
shape.isRectangleObject
```

**Description**

Read-only property; if true, the shape is a primitive Rectangle object (was created using the Rectangle Primitive tool).

**Example**

The following example displays "true" if the first selected item is a primitive Rectangle object, "false" if it is not:

```
var sel = fl.getDocumentDOM().selection[0];
fl.trace(sel.isRectangleObject);
```

**See also**

[shape.isOvalObject](#)

## shape.members

**Availability**

Flash CS4 Professional.

**Usage**

`shape.members`

**Description**

Read-only property; an array of objects in the currently selected group. This property is available only if the value of `shape.isGroup` is true). Raw shapes in the group are not included in the `shape.members` array.

For example, if the group contains three drawing objects and three raw shapes, the `shape.members` array contains three entries, one for each of the drawing objects. If the group contains only raw shapes, the array is empty.

**Example**

The following code displays the number of cubic segments of each drawing object in the currently selected group:

```
var shapesArray = fl.getDocumentDOM().selection[0].members;
for (i=0; i<shapesArray.length; i++) {
    fl.trace(shapesArray[i].numCubicSegments);
}
```

**See also**

[shape.isGroup](#)

## shape.numCubicSegments

**Availability**

Flash CS4 Professional.

**Usage**

`shape.numCubicSegments`

**Description**

Read-only property; the number of cubic segments in the shape.

**Example**

Assuming a square or rectangle shape is selected, the following code displays “4” in the Output panel:

```
var theShape = fl.getDocumentDOM().selection[0];
fl.trace(theShape.numCubicSegments);
```

## shape.vertices

**Availability**

Flash MX 2004.

**Usage**

```
shape.vertices
```

**Description**

Read-only property; an array of Vertex objects (see [Vertex object](#)).

**Example**

The following example stores the first selected object in the `someShape` variable, and then shows the number of vertices for that object in the Output panel:

```
var someShape = fl.getDocumentDOM().selection[0];
fl.trace("The shape has " + someShape.vertices.length + " vertices.");
```

# Chapter 37: SoundItem object

## soundItem summary

**Inheritance** [Item object](#) > SoundItem object

### Availability

Flash MX 2004.

### Description

The SoundItem object is a subclass of the Item object. It represents a library item used to create a sound. See also [frame.soundLibraryItem](#) and [Item object](#).

### Method summary

In addition to the Item object methods, the SoundItem object has the following method:

Method	Description
<a href="#">soundItem.exportToFile()</a>	Exports the specified item to an MP3 or WAV file (Macintosh and Windows).

### Property summary

In addition to the Item object properties, the following properties are available for the SoundItem object:

Property	Description
<a href="#">soundItem.bitRate</a>	A string that specifies the bit rate of a sound in the library. Available only for the MP3 compression type.
<a href="#">soundItem.bits</a>	A string that specifies the bits value for a sound in the library that has ADPCM compression.
<a href="#">soundItem.compressionType</a>	A string that specifies the compression type for a sound in the library.
<a href="#">soundItem.convertStereoToMono</a>	A Boolean value available only for MP3 and Raw compression types.
<a href="#">soundItem.fileLastModifiedDate</a>	Read-only; a string containing a hexadecimal number that represents the number of seconds that have elapsed between January 1, 1970, and the modification date of the original file (on disk) at the time the file was imported to the library.
<a href="#">soundItem.lastModifiedDate</a>	Read-only; the modification date of the sound item in the Library.
<a href="#">soundItem.originalCompressionType</a>	Read-only; a string that specifies whether the specified item was imported as an MP3 file.
<a href="#">soundItem.quality</a>	A string that specifies the playback quality of a sound in the library. Available only for the MP3 compression type.
<a href="#">soundItem.sampleRate</a>	A string that specifies the sample rate for the audio clip.
<a href="#">soundItem.sourceFileExists</a>	Read-only; a Boolean value that specifies whether the file that was imported to the Library still exists in the location from where it was imported.

Property	Description
<code>soundItem.sourceFileIsCurrent</code>	Read-only; a Boolean value that specifies whether the file modification date of the Library item is the same as the modification date on disk of the file that was imported.
<code>soundItem.sourceFilePath</code>	Read-only; a string, expressed as a file:/// URI, that represents the path and name of the file that was imported into the Library.
<code>soundItem.useImportedMP3Quality</code>	A Boolean value; if <code>true</code> , all other properties are ignored, and the imported MP3 quality is used.

## soundItem.bitRate

### Availability

Flash MX 2004.

### Usage

```
soundItem.bitRate
```

### Description

Property; a string that specifies the bit rate of a sound in the library. This property is available only for the MP3 compression type. Acceptable values are "8 kbps", "16 kbps", "20 kbps", "24 kbps", "32 kbps", "48 kbps", "56 kbps", "64 kbps", "80 kbps", "112 kbps", "128 kbps", and "160 kbps". Stereo sounds exported at 8 Kbps or 16 Kbps are converted to mono. The property is undefined for other compression types.

If you want to specify a value for this property, set `soundItem.useImportedMP3Quality` to `false`.

### Example

The following example displays the `bitRate` value in the Output panel if the specified item in the library has the MP3 compression type:

```
alert(f1.getDocumentDOM().library.items[0].bitRate);
```

### See also

`soundItem.compressionType`, `soundItem.convertStereoToMono`

## soundItem.bits

### Availability

Flash MX 2004.

### Usage

```
soundItem.bits
```

### Description

Property; a string that specifies the bits value for a sound in the library that has ADPCM compression. Acceptable values are "2 bit", "3 bit", "4 bit", and "5 bit".

If you want to specify a value for this property, set `soundItem.useImportedMP3Quality` to false.

#### Example

The following example displays the bits value in the Output panel if the currently selected item in the library has the ADPCM compression type:

```
alert(f1.getDocumentDOM().library.items[0].bits);
```

#### See also

`soundItem.compressionType`

## soundItem.compressionType

#### Availability

Flash MX 2004.

#### Usage

`soundItem.compressionType`

#### Description

Property; a string that specifies the compression type for a sound in the library. Acceptable values are "Default", "ADPCM", "MP3", "Raw", and "Speech".

If you want to specify a value for this property, set `soundItem.useImportedMP3Quality` to false.

#### Example

The following example changes an item in the library to compression type Raw:

```
f1.getDocumentDOM().library.items[0].compressionType = "Raw";
```

The following example changes the compression type of the selected library items to Speech:

```
f1.getDocumentDOM().library.getSelectedItems().compressionType = "Speech";
```

#### See also

`soundItem.originalCompressionType`

## soundItem.convertStereoToMono

#### Availability

Flash MX 2004.

#### Usage

`soundItem.convertStereoToMono`

**Description**

Property; a Boolean value available only for MP3 and Raw compression types. Setting this value to `true` converts a stereo sound to mono; `false` leaves it as stereo. For the MP3 compression type, if `soundItem.bitRate` is less than 20 Kbps, this property is ignored and forced to `true` (see `soundItem.bitRate`).

If you want to specify a value for this property, set `soundItem.useImportedMP3Quality` to `false`.

**Example**

The following example converts an item in the library to mono only if the item has the MP3 or Raw compression type:

```
f1.getDocumentDOM().library.items[0].convertStereoToMono = true;
```

**See also**

`soundItem.compressionType`

## soundItem.exportToFile()

**Availability**

Flash CS4 Professional.

**Usage**

```
soundItem.exportToFile(fileURI)
```

**Parameters**

`fileURI` A string, expressed as a file:/// URI, that specifies the path and name of the exported file.

**Returns**

A Boolean value of `true` if the file was exported successfully; `false` otherwise.

**Description**

Method; exports the specified item to a WAV or MP3 file. Export settings are based on the item being exported.

When exporting sound items, you should check if the `soundItem.originalCompressionType` property is equal to "RAW." If this check is false, you can only export the file as MP3. (Optionally, you can try exporting as a WAV file, and if the function returns false, then try to export to MP3.)

**Example**

Assuming that the first item in the Library is a sound item, the following code exports it as a WAV file:

```
var soundFileURL = "file:///C|/out.wav";
var libItem = f1.getDocumentDOM().library.items[0];
libItem.exportToFile(soundFileURL);
```

## soundItem.fileLastModifiedDate

**Availability**

Flash CS4 Professional.

**Usage**

```
soundItem.fileLastModifiedDate
```

**Description**

Read-only property: a string containing a hexadecimal number that represents the number of seconds that have elapsed between January 1, 1970, and the modification date of the original file (on disk) at the time the file was imported to the library. If the file no longer exists, this value is "00000000".

**Example**

Assuming that the first item in the Library is a sound item, the following code displays a hexadecimal number as described above.

```
var libItem = fl.getDocumentDOM().library.items[0];
fl.trace("Mod date when imported = " + libItem.fileLastModifiedDate);
```

**See also**

[soundItem.sourceFileExists](#), [soundItem.sourceFileIsCurrent](#), [soundItem.sourceFilePath](#),  
[FLfile.getModificationDate\(\)](#)

## soundItem.lastModifiedDate

**Availability**

Flash Pro CS6.

**Usage**

```
soundItem.lastModifiedDate
```

**Description**

Read-only property; a hexadecimal value indicating the modification date and time of the sound item. This value is incremented every time the sound item is imported. For example, selecting the Update button from the Sound Properties dialog will trigger an import.

**Example**

Assuming the first item in the Library is a sound item, the following code displays a hex number as described above.

```
var libItem = fl.getDocumentDOM().library.items[0];
fl.trace("Mod date when imported = " + libItem.lastModifiedDate);
```

## soundItem.originalCompressionType

**Availability**

Flash CS4 Professional.

**Usage**

```
soundItem.originalCompressionType
```

**Description**

Read-only property: a string that specifies whether the specified item was imported as an mp3 file. Possible values for this property are "RAW" and "MP3".

**Example**

Assuming that the first item in the Library is a sound item, the following code displays "MP3" if the file was imported into the Library as an MP3 file, or "RAW" if it was not:

```
var libItem = fl.getDocumentDOM().library.items[0];
fl.trace("Imported compression type = "+ libItem.originalCompressionType);
```

**See also**

[soundItem.compressionType](#)

## soundItem.quality

**Availability**

Flash MX 2004.

**Usage**

`soundItem.quality`

**Description**

Property; a string that specifies the playback quality of a sound in the library. This property is available only for the MP3 compression type. Acceptable values are "Fast", "Medium", and "Best".

If you want to specify a value for this property, set [soundItem.useImportedMP3Quality](#) to false.

**Example**

The following example sets the playback quality of an item in the library to Best if the item has the MP3 compression type:

```
fl.getDocumentDOM().library.items[0].quality = "Best";
```

**See also**

[soundItem.compressionType](#)

## soundItem.sampleRate

**Availability**

Flash MX 2004.

**Usage**

`soundItem.sampleRate`

**Description**

Property; a string that specifies the sample rate for the audio clip. This property is available only for the ADPCM, Raw, and Speech compression types. Acceptable values are "5 kHz", "11 kHz", "22 kHz", and "44 kHz".

If you want to specify a value for this property, set `soundItem.useImportedMP3Quality` to false.

**Example**

The following example sets the sample rate of an item in the library to 5 kHz if the item has the ADPCM, Raw, or Speech compression type:

```
f1.getDocumentDOM().library.items[0].sampleRate = "5 kHz";
```

**See also**

`soundItem.compressionType`

## soundItem.sourceFileExists

**Availability**

Flash CS4 Professional.

**Usage**

```
soundItem.sourceFileExists
```

**Description**

Read-only property: a Boolean value of `true` if the file that was imported to the Library still exists in the location from where it was imported; `false` otherwise.

**Example**

Assuming that the first item in the Library is a sound item, the following code displays "true" if the file that was imported into the Library still exists.

```
var libItem = f1.getDocumentDOM().library.items[0];
f1.trace("sourceFileExists = "+ libItem.sourceFileExists);
```

**See also**

`soundItem.sourceFileIsCurrent`, `soundItem.sourceFilePath`

## soundItem.sourceFileIsCurrent

**Availability**

Flash CS4 Professional.

**Usage**

```
soundItem.sourceFileIsCurrent
```

**Description**

Read-only property: a Boolean value of `true` if the file modification date of the Library item is the same as the modification date on disk of the file that was imported; `false` otherwise.

**Example**

Assuming that the first item in the Library is a sound item, the following code displays "true" if the file that was imported has not been modified on disk since it was imported.

```
var libItem = fl.getDocumentDOM().library.items[0];
fl.trace("fileIsCurrent = "+ libItem.sourceFileIsCurrent);
```

**See also**

[soundItem.fileLastModifiedDate](#), [soundItem.sourceFilePath](#)

## soundItem.sourceFilePath

**Availability**

Flash CS4 Professional.

**Usage**

`soundItem.sourceFilePath`

**Description**

Read-only property: a string, expressed as a file:/// URI, that represents the path and name of the file that was imported into the Library.

**Example**

The following example displays the name and source file path of any items in the library that are of type "sound":

```
for (idx in fl.getDocumentDOM().library.items) {
if (fl.getDocumentDOM().library.items[idx].itemType == "sound") {
    var myItem = fl.getDocumentDOM().library.items[idx];
    fl.trace(myItem.name + " source is " + myItem.sourceFilePath);
}
}
```

**See also**

[soundItem.sourceFileExists](#)

## soundItem.useImportedMP3Quality

**Availability**

Flash MX 2004.

**Usage**

`soundItem.useImportedMP3Quality`

**Description**

Property; a Boolean value. If true, all other properties are ignored, and the imported MP3 quality is used.

**Example**

The following example sets an item in the library to use the imported MP3 quality:

```
f1.getDocumentDOM().library.items[0].useImportedMP3Quality = true;
```

**See also**

[soundItem.compressionType](#)

# Chapter 38: SpriteSheetExporter object

## SpriteSheetExporter summary

**Inheritance** [Item object](#) > SpriteSheetExporter object

### Availability

Flash Pro CS6.

### Description

The SpriteSheetExporter object is a subclass of the [Item object](#).

### Method summary

In addition to the Item object methods, you can use the following methods with the SpriteSheetExporter object:

Method	Description
<a href="#">SpriteSheetExporter.addBitmap()</a>	Adds a bitmap or bitmapItem to the sprite sheet.
<a href="#">SpriteSheetExporter.addSymbol()</a>	Adds a symbol to be used to generate the sprite sheet.
<a href="#">SpriteSheetExporter.beginExport()</a>	Initializes the SpriteSheetExporter to create a new sprite sheet.
<a href="#">SpriteSheetExporter.changeSymbol()</a>	Changes which symbol frames will be added to the sprite sheet.
<a href="#">SpriteSheetExporter.exportSpriteSheet()</a>	Export the sprite sheet into a an image file.
<a href="#">SpriteSheetExporter.removeBitmap()</a>	Remove a bitmap object from the sprite sheet.
<a href="#">SpriteSheetExporter.removeSymbol()</a>	Remove a symbol from the sprite sheet.

### Property summary

In addition to the Item object properties, the following properties are available for the SpriteSheetExporter object:

Property	Description
<a href="#">SpriteSheetExporter.algorithm</a>	Sets the encoding algorithm for the sprite sheet.
<a href="#">SpriteSheetExporter.allowRotate</a>	Allows sprites to be rotated when added to the sprite sheet.
<a href="#">SpriteSheetExporter.allowTrimming</a>	Allows trimming of whitespace around sprites.
<a href="#">SpriteSheetExporter.app</a>	A string indicating the name of the application generating the sprite sheet.
<a href="#">SpriteSheetExporter.autoSize</a>	Automatically size the sprite sheet to fit all sprites.
<a href="#">SpriteSheetExporter.borderPadding</a>	The amount of padding around the sprite sheet borders, in pixels.
<a href="#">SpriteSheetExporter.canBorderPad</a>	A Boolean value indicating whether border padding is supported by the framework specified by the layoutFormat property.
<a href="#">SpriteSheetExporter.canRotate</a>	A Boolean value indicating whether sprite rotation is supported by the framework specified by the layoutFormat property.
<a href="#">SpriteSheetExporter.canShapePad</a>	A Boolean value indicating whether shape padding is supported by the framework specified by the layoutFormat property.

Property	Description
<code>SpriteSheetExporter.canStackDuplicateFrames</code>	A Boolean value indicating whether sprite stacking is supported by the framework specified by the <code>layoutFormat</code> property.
<code>SpriteSheetExporter.canTrim</code>	A Boolean value indicating whether shape trimming is supported by the framework specified by the <code>layoutFormat</code> property.
<code>SpriteSheetExporter.format</code>	The format of the sprite sheet image file.
<code>SpriteSheetExporter.image</code>	The name of the image file of the sprite sheet.
<code>SpriteSheetExporter.layoutFormat</code>	The format of the sprite sheet metadata.
" <code>SpriteSheetExporter.maxSheetHeight</code> " on page 442	Controls the maximum height of the generated sprite sheet when <code>autoSize</code> = true, clipped to a maximum value of 8192.
" <code>SpriteSheetExporter.maxSheetWidth</code> " on page 442	Controls the maximum width of the generated sprite sheet when <code>autoSize</code> = true, clipped to a maximum value of 8192.
<code>SpriteSheetExporter.overflowed</code>	A Boolean value indicating whether all the specified frames can fit in the specified sprite sheet size.
<code>SpriteSheetExporter.shapePadding</code>	The amount of padding around each sprite, in pixels.
<code>SpriteSheetExporter.sheetHeight</code>	The height of the sprite sheet, in pixels.
<code>SpriteSheetExporter.sheetWidth</code>	The width of the sprite sheet, in pixels.
<code>SpriteSheetExporter.stackDuplicateFrames</code>	A Boolean value indicating whether to stack identical symbol frames in the sprite sheet.
<code>SpriteSheetExporter.version</code>	The version number of the application creating the sprite sheet.

## SpriteSheetExporter.addBitmap()

### Availability

Flash Pro CS6.

### Usage

```
SpriteSheetExporter.addBitmap(bitmap)
```

### Parameters

**bitmap** The BitmapItem or Bitmap to include in the sprite sheet.

### Returns

Nothing.

### Description

Method; Adds the specified bitmap or bitmapItem to the sprite sheet object.

## SpriteSheetExporter.addSymbol()

### Availability

Flash Pro CS6.

### Usage

```
SpriteSheetExporter.addSymbol(symbol [, name] [, beginFrame] [, endFrame])
```

### Parameters

**symbol** Object; The SymbolItem or SymbolInstance to include in the sprite sheet.

**name** String; The name of the symbol instance to add to the sprite sheet.

**beginFrame** The beginning frame of the symbol to include in the sprite sheet.

**endFrame** The last frame of the symbol to include in the sprite sheet.

### Returns

Boolean.

### Description

Method; Adds the specified SymbolItem or SymbolInstance to be used to generate the sprite sheet.

## SpriteSheetExporter.algorithm

### Availability

Flash Pro CS6.

### Usage

```
SpriteSheetExporter.algorithm
```

### Description

Property; A string indicating which algorithm to use to pack the sprite sheet. Valid values are "basic" (the default), and "maxRects".

### Example

```
var exporter = new SpriteSheetExporter;  
exporter.algorithm = "maxRects";
```

## SpriteSheetExporter.allowRotate

### Availability

Flash Pro CS6.

### Usage

```
SpriteSheetExporter.allowRotate
```

**Description**

Property; A boolean value indicating whether the symbol frames can be rotated when packed into the sprite sheet.

**Example**

```
var exporter = new SpriteSheetExporter;  
exporter.allowRotate = true;
```

## SpriteSheetExporter.allowTrimming

**Availability**

Flash Pro CS6.

**Usage**

```
SpriteSheetExporter.allowTrimming
```

**Description**

Property; A boolean value indicating whether the symbol frames can be trimmed of any extra whitespace when packed into the sprite sheet. The default value is true.

**Example**

```
var exporter = new SpriteSheetExporter;  
exporter.allowTrimming = false;
```

## SpriteSheetExporter.app

**Availability**

Flash Pro CS6.

**Usage**

```
SpriteSheetExporter.app
```

**Description**

Read-only property; A string indicating the name of the application that is generating the sprite sheet. This property is provided for use by any future sprite sheet generator plugins that may be created for Flash Pro.

**Example**

```
var exporter = new SpriteSheetExporter;  
alert(exporter.app);  
// "Flash Pro CS6"
```

## SpriteSheetExporter.autoSize

### Availability

Flash Pro CS6.

### Usage

```
SpriteSheetExporter.autoSize
```

### Description

Property; A boolean value indicating whether the sprite sheet exporter should calculate the overall size of the sprite sheet on its own.

### Example

```
var exporter = new SpriteSheetExporter;  
exporter.autoSize = false;
```

## SpriteSheetExporter.beginExport()

### Availability

Flash Pro CS6.

### Usage

```
SpriteSheetExporter.beginExport()
```

### Parameters

None.

### Returns

Nothing.

### Description

Method; Initializes the SpriteSheetExporter to create a new sprite sheet. This method is not necessary if you create the exporter from new. It is necessary if you reuse the same exporter to make multiple sprite sheets.

## SpriteSheetExporter.borderPadding

### Availability

Flash Pro CS6.

### Usage

```
SpriteSheetExporter.borderPadding
```

### Description

Property; An integer indicating the number of pixels of padding to add around each sprite in the sprite sheet.

**Example**

```
var exporter = new SpriteSheetExporter;  
exporter.borderPadding = 5;
```

## SpriteSheetExporter.canBorderPad

**Availability**

Flash Pro CS6.

**Usage**

```
SpriteSheetExporter.canBorderPad
```

**Description**

Read-only property; A boolean value indicating whether the framework specified by the `SpriteSheetExporter.layoutFormat` property supports border padding.

**Example**

```
var exporter = new SpriteSheetExporter;  
alert(exporter.canBorderPad);
```

## SpriteSheetExporter.canRotate

**Availability**

Flash Pro CS6.

**Usage**

```
SpriteSheetExporter.canRotate
```

**Description**

Read-only property; A boolean value indicating whether the framework specified by the `SpriteSheetExporter.layoutFormat` property may rotate symbol frames when adding them to the sprite sheet.

**Example**

```
var exporter = new SpriteSheetExporter;  
alert(exporter.canRotate);
```

## SpriteSheetExporter.canTrim

**Availability**

Flash Pro CS6.

**Usage**

```
SpriteSheetExporter.canTrim
```

**Description**

Read-only property; A boolean value indicating whether the framework specified by the `SpriteSheetExporter.layoutFormat` property supports trimming extra whitespace from symbol frames when adding them to the sprite sheet.

**Example**

```
var exporter = new SpriteSheetExporter;
alert(exporter.canTrim);
```

## **SpriteSheetExporter.canShapePad**

**Availability**

Flash Pro CS6.

**Usage**

```
SpriteSheetExporter.canShapePad
```

**Description**

Read-only property; A boolean value indicating whether the framework specified by the `SpriteSheetExporter.layoutFormat` property supports shape padding.

**Example**

```
var exporter = new SpriteSheetExporter;
alert(exporter.canShapePad);
```

## **SpriteSheetExporter.canStackDuplicateFrames**

**Availability**

Flash Pro CS6.

**Usage**

```
SpriteSheetExporter.canStackDuplicateFrames
```

**Description**

Read-only property; A boolean value indicating whether the framework specified by the `SpriteSheetExporter.layoutFormat` property supports stacking duplicate symbol frames within the sprite sheet.

**Example**

```
var exporter = new SpriteSheetExporter;
alert(exporter.canStackDuplicateFrames);
```

## SpriteSheetExporter.changeSymbol()

### Availability

Flash Pro CS6.

### Usage

```
SpriteSheetExporter.changeSymbol( symbol [, beginFrame] [, endFrame] )
```

### Parameters

**symbol** Object; A SymbolItem or SymbolInstance that has already been added to the sprite sheet.

**beginFrame** Optional. The beginning frame of the symbol to include in the sprite sheet.

**endFrame** Optional. The last frame of the symbol to include in the sprite sheet.

### Returns

Boolean.

### Description

Method; Changes the frame range of the symbol that will be used in the sprite sheet.

## SpriteSheetExporter.exportSpriteSheet()

### Availability

Flash Pro CS6.

### Usage

```
SpriteSheetExporter.exportSpriteSheet( path, imageFormat [, writeMetaData] )
```

### Parameters

**path** String; A file path with no extension to be used as the base name of the exported files.

**imageFormat** String or Object. A string or an object that describes the type of image format to generate.

- String: Valid values are either "png" or "jpg", the exporter use whatever values where last used.
- Object: If you pass an object, it needs at least a string property "format" which is either "png" or "jpg". Optionally the object may include "backgroundColor," which is a valid color value (alpha may be included); "quality" (jpg only), which is an integer from 1 to 100; and "bitDepth" (png only) which can be 8, 24 or 32.

**writeMetaData** Optional. Boolean; Whether or not to write the metadata file with the image file. The default value is true.

### Returns

String.

### Description

Method; Exports the sprite sheet into a an image file and a metadata file based on the path parameter. The return string is the metadata generated by the sprite sheet plugin or nothing if an error occurred.

**Example**

The following examples exports to a PNG with transparency:

```
var sse = new SpriteSheetExporter;
sse.exportSpriteSheet("file:///C|/xxx/sprite-sheet/test" , {format:"png", bitDepth:32,
backgroundColor:"#00000000"})
```

## **SpriteSheetExporter.format**

**Availability**

Flash Pro CS6.

**Usage**

```
SpriteSheetExporter.format
```

**Description**

Read-only property; A string value indicating the format of the sprite sheet image file. Possible values include “RGBA8888”, “RGB888x”, and “RGB8”. This property is provided for use by sprite sheet generator plugins.

**Example**

```
var exporter = new SpriteSheetExporter;
alert(exporter.format);
// "RGBA8888"
```

## **SpriteSheetExporter.image**

**Availability**

Flash Pro CS6.

**Usage**

```
SpriteSheetExporter.image
```

**Description**

Read-only property; A string value indicating the name of the sprite sheet image file. This property is provided for use by sprite sheet generator plugins.

**Example**

```
var exporter = new SpriteSheetExporter;
alert(exporter.image);
```

## **SpriteSheetExporter.layoutFormat**

**Availability**

Flash Pro CS6.

**Usage**

```
SpriteSheetExporter.layoutFormat
```

**Description**

Property; A string value indicating the format of the sprite sheet metadata. Valid values depend on the contents of the Sprite Sheet Plugin directory. "JSON" (JavaScript Object Notation) is one possible value. Your script passes the layout format's ID string to match one of the .jsfl files defined for the Sprite Sheet plugin (located in the *flashroot/Common/Configuration/Sprite Sheet Plugins* folder). These files have the naming convention *layoutformat.plugin.jsfl* and you pass a string set to *layoutformat*. For example, you would set *layoutFormat* equal to "Starling", "cocos2D v2", or "cocos2D v3".

**Example**

```
var exporter = new SpriteSheetExporter;  
exporter.layoutFormat = "JSON";
```

## SpriteSheetExporter.maxSheetHeight

**Availability**

Flash Pro CC.

**Usage**

```
SpriteSheetExporter.maxSheetHeight
```

**Description**

Controls the maximum height of the generated sprite sheet when *autoSize* = true, clipped to a maximum value of 8192.

**Example**

```
var exporter = new SpriteSheetExporter; exporter.autoSize to true; exporter.maxSheetHeight = 512;
```

## SpriteSheetExporter.maxSheetWidth

**Availability**

Flash Pro CC.

**Usage**

```
SpriteSheetExporter.maxSheetWidth
```

**Description**

Controls the maximum width of the generated sprite sheet when *autoSize* = true, clipped to a maximum value of 8192.

**Example**

```
var exporter = new SpriteSheetExporter; exporter.autoSize to true; exporter.maxSheetWidth = 512;
```

## SpriteSheetExporter.overflowed

### Availability

Flash Pro CS6.

### Usage

```
SpriteSheetExporter.overflowed
```

### Description

Read-only property; A boolean value indicating whether all the exported symbol frames cannot fit within the currently specified size of the sprite sheet.

### Example

```
var exporter = new SpriteSheetExporter;  
if (exporter.overflowed)  
{  
    exporter.sheetWidth = 1024;  
}
```

## SpriteSheetExporter.removeBitmap()

### Availability

Flash Pro CS6.

### Usage

```
SpriteSheetExporter.removeBitmap(bitmap)
```

### Parameters

**bitmap** The BitmapItem or Bitmap to remove from the sprite sheet.

### Returns

Nothing.

### Description

Method; Removes the specified bitmap or bitmapItem from the sprite sheet object.

## SpriteSheetExporter.removeSymbol()

### Availability

Flash Pro CS6.

### Usage

```
SpriteSheetExporter.removeSymbol(symbol)
```

**Parameters**

**symbol** Object; The SymbolItem or SymbolInstance to remove from the sprite sheet.

**Returns**

Boolean.

**Description**

Method; Removes the specified SymbolItem or SymbolInstance from the sprite sheet.

## SpriteSheetExporter.shapePadding

**Availability**

Flash Pro CS6.

**Usage**

```
SpriteSheetExporter.shapePadding
```

**Description**

Property; An integer value indicating how many pixels of padding should be added to each exported symbol frame when adding it to the sprite sheet.

**Example**

```
var exporter = new SpriteSheetExporter;  
exporter.shapePadding = 10;
```

## SpriteSheetExporter.sheetHeight

**Availability**

Flash Pro CS6.

**Usage**

```
SpriteSheetExporter.sheetHeight
```

**Description**

Property; An integer value specifying the height of the sprite sheet. This value is read-only if `SpriteSheetExporter.autoSize` is set to true.

**Example**

```
var exporter = new SpriteSheetExporter;  
exporter.sheetHeight = 512;
```

## SpriteSheetExporter.sheetWidth

### Availability

Flash Pro CS6.

### Usage

```
SpriteSheetExporter.sheetWidth
```

### Description

Property; An integer value specifying the width of the sprite sheet. This value is read-only if `SpriteSheetExporter.autoSize` is set to true.

### Example

```
var exporter = new SpriteSheetExporter;  
exporter.sheetwidth = 512;
```

## SpriteSheetExporter.stackDuplicateFrames

### Availability

Flash Pro CS6.

### Usage

```
SpriteSheetExporter.stackDuplicateFrames
```

### Description

Property; A boolean value specifying whether the sprite sheet can stack duplicate symbol frames when adding them to the sprite sheet. The default is true.

### Example

```
var exporter = new SpriteSheetExporter;  
exporter.stackDuplicateFrames = true;
```

## SpriteSheetExporter.version

### Availability

Flash Pro CS6.

### Usage

```
SpriteSheetExporter.version
```

**Description**

Read-only property; A string indicating the version number of the Flash Pro application. This property is provided for use by sprite sheet plugins.

**Example**

```
var exporter = new SpriteSheetExporter;  
alert(exporter.version);  
// "12.0.0.416"
```

# Chapter 39: Stroke object

## stroke summary

### Availability

Flash MX 2004.

### Description

The Stroke object contains all the settings for a stroke, including the custom settings. This object represents the information contained in the Property inspector. Using the Stroke object together with the `document.setCustomStroke()` method, you can change the stroke settings for the Tools panel, the Property inspector, and the current selection. You can also get the stroke settings of the Tools panel and Property inspector, or of the current selection, by using the `document.getCustomStroke()` method.

This object always has the following four properties: `style`, `thickness`, `color`, and `breakAtCorners`. (In Flash CS3, the `breakAtCorners` property was deprecated in favor of `stroke.joinType`.) Other properties can be set, depending on the value of the `stroke.style` property.

### Property summary

The following properties are available for the Stroke object:

Property	Description
<code>stroke.breakAtCorners</code>	A Boolean value, same as the Sharp Corners setting in the custom Stroke Style dialog box.
<code>stroke.capType</code>	A string that specifies the type of cap for the stroke.
<code>stroke.color</code>	A string, hexadecimal value, or integer that represents the stroke color.
<code>stroke.curve</code>	A string that specifies the type of hatching for the stroke.
<code>stroke.dash1</code>	An integer that specifies the lengths of the solid part of a dashed line.
<code>stroke.dash2</code>	An integer that specifies the lengths of the blank part of a dashed line.
<code>stroke.density</code>	A string that specifies the density of a stippled line.
<code>stroke.dotSize</code>	A string that specifies the dot size of a stippled line.
<code>stroke.dotSpace</code>	An integer that specifies the spacing between dots in a dotted line.
<code>stroke.hatchThickness</code>	A string that specifies the thickness of a hatch line.
<code>stroke.jiggle</code>	A string that specifies the jiggle property of a hatched line.
<code>stroke.joinType</code>	A string that specifies the type of join for the stroke.
<code>stroke.length</code>	A string that specifies the length of a hatch line.
<code>stroke.miterLimit</code>	A float value that specifies the angle above which the tip of the miter will be truncated by a segment.
<code>stroke.pattern</code>	A string that specifies the pattern of a ragged line.
<code>stroke.rotate</code>	A string that specifies the rotation of a hatch line.
<code>stroke.scaleType</code>	A string that specifies the type of scale to be applied to the stroke.

Property	Description
<code>stroke.shapeFill</code>	A <a href="#">Fill object</a> that represents the fill settings of the stroke.
<code>stroke.space</code>	A string that specifies the spacing of a hatched line.
<code>stroke.strokeHinting</code>	A Boolean value that specifies whether stroke hinting is set on the stroke.
<code>stroke.style</code>	A string that describes the stroke style.
<code>stroke.thickness</code>	An integer that specifies the stroke size.
<code>stroke.variation</code>	A string that specifies the variation of a stippled line.
<code>stroke.waveHeight</code>	A string that specifies the wave height of a ragged line.
<code>stroke.waveLength</code>	A string that specifies the wave length of a ragged line.

## stroke.breakAtCorners

### Availability

Flash MX 2004. Deprecated in Flash CS3 in favor of [stroke.joinType](#).

### Usage

```
stroke.breakAtCorners
```

### Description

Property; a Boolean value. This property is the same as the Sharp Corners setting in the custom Stroke Style dialog box.

### Example

The following example sets the `breakAtCorners` property to `true`:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.breakAtCorners = true;
fl.getDocumentDOM().setCustomStroke(myStroke);
```

## stroke.capType

### Availability

Flash 8.

### Usage

```
stroke.capType
```

### Description

Property; a string that specifies the type of cap for the stroke. Acceptable values are "none", "round", and "square".

### Example

The following example sets the stroke cap type to `round`:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.capType = "round";
fl.getDocumentDOM().setCustomStroke(myStroke);
```

## stroke.color

### Availability

Flash MX 2004. In Flash 8 and later, this property is deprecated in favor of `stroke.shapeFill.color`.

### Usage

```
stroke.color
```

### Description

Property; the color of the stroke, in one of the following formats:

- A string in the format "#RRGGBB" or "#RRGGBBAA"
- A hexadecimal number in the format 0xRRGGBB
- An integer that represents the decimal equivalent of a hexadecimal number

### Example

The following example sets the stroke color:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.color = "#000000";
fl.getDocumentDOM().setCustomStroke(myStroke);
```

### See also

[stroke.shapeFill](#)

## stroke.curve

### Availability

Flash MX 2004.

### Usage

```
stroke.curve
```

### Description

Property; a string that specifies the type of hatching for the stroke. This property can be set only if the `stroke.style` property is set to "hatched" (see [stroke.style](#)). Acceptable values are "straight", "slight curve", "medium curve", and "very curved".

### Example

The following example sets the curve property, as well as others, for a stroke having the `hatched` style:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.style = "hatched";
myStroke.curve = "straight";
myStroke.space = "close";
myStroke.jiggle = "wild";
myStroke.rotate = "free";
myStroke.length = "slight";
myStroke.hatchThickness = "thin";
fl.getDocumentDOM().setCustomStroke(myStroke);
```

## stroke.dash1

### Availability

Flash MX 2004.

### Usage

```
stroke.dash1
```

### Description

Property; an integer that specifies the lengths of the solid parts of a dashed line. This property is available only if the `stroke.style` property is set to dashed (see [stroke.style](#)).

### Example

The following example sets the `dash1` and `dash2` properties for a stroke style of dashed:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.style = "dashed";
myStroke.dash1 = 1;
myStroke.dash2 = 2;
fl.getDocumentDOM().setCustomStroke(myStroke);
```

## stroke.dash2

### Availability

Flash MX 2004.

### Usage

```
stroke.dash2
```

### Description

Property; an integer that specifies the lengths of the blank parts of a dashed line. This property is available only if the `stroke.style` property is set to dashed (see [stroke.style](#)).

### Example

See [stroke.dash1](#).

## stroke.density

### Availability

Flash MX 2004.

### Usage

```
stroke.density
```

### Description

Property; a string that specifies the density of a stippled line. This property is available only if the `stroke.style` property is set to `stipple` (see [stroke.style](#)). Acceptable values are "very dense", "dense", "sparse", and "very sparse".

### Example

The following example sets the density property to `sparse` for the stroke style of `stipple`:

```
var myStroke = fl.getDocumentDOM().getCustomStroke() ;
myStroke.style = "stipple";
myStroke.dotSpace= 3;
myStroke.variation = "random sizes";
myStroke.density = "sparse";
fl.getDocumentDOM().setCustomStroke(myStroke) ;
```

## stroke.dotSize

### Availability

Flash MX 2004.

### Usage

```
stroke.dotSize
```

### Description

Property; a string that specifies the dot size of a stippled line. This property is available only if the `stroke.style` property is set to `stipple` (see [stroke.style](#)). Acceptable values are "tiny", "small", "medium", and "large".

The following example sets the `dotSize` property to `tiny` for the stroke style of `stipple`:

```
var myStroke = fl.getDocumentDOM().getCustomStroke() ;
myStroke.style = "stipple";
myStroke.dotSpace= 3;
myStroke.dotsize = "tiny";
myStroke.variation = "random sizes";
myStroke.density = "sparse";
fl.getDocumentDOM().setCustomStroke(myStroke) ;
```

## stroke.dotSpace

### Availability

Flash MX 2004.

### Usage

```
stroke.dotSpace
```

### Description

Property; an integer that specifies the spacing between dots in a dotted line. This property is available only if the `stroke.style` property is set to dotted. See [stroke.style](#).

### Example

The following example sets the `dotSpace` property to 3 for a stroke style of dotted:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.style = "dotted";
myStroke.dotSpace= 3;
fl.getDocumentDOM().setCustomStroke(myStroke);
```

## stroke.hatchThickness

### Availability

Flash MX 2004.

### Usage

```
stroke.hatchThickness
```

### Description

Property; a string that specifies the thickness of a hatch line. This property is available only if the `stroke.style` property is set to hatched (see [stroke.style](#)). Acceptable values are "hairline", "thin", "medium", and "thick".

### Example

The following example sets the `hatchThickness` property to thin for a stroke style of hatched:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.style = "hatched";
myStroke.curve = "straight";
myStroke.space = "close";
myStroke.jiggle = "wild";
myStroke.rotate = "free";
myStroke.length = "slight";
myStroke.hatchThickness = "thin";
fl.getDocumentDOM().setCustomStroke(myStroke);
```

## stroke.jiggle

### Availability

Flash MX 2004.

### Usage

```
stroke.jiggle
```

### Description

Property; a string that specifies the jiggle property of a hatched line. This property is available only if the `stroke.style` property is set to `hatched` (see [stroke.style](#)). Acceptable values are "none", "bounce", "loose", and "wild".

### Example

The following example sets the `jiggle` property to `wild` for a stroke style of `hatched`:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.style = "hatched";
myStroke.curve = "straight";
myStroke.space = "close";
myStroke.jiggle = "wild";
myStroke.rotate = "free";
myStroke.length = "slight";
myStroke.hatchThickness = "thin";
fl.getDocumentDOM().setCustomStroke(myStroke);
```

## stroke.joinType

### Availability

Flash 8.

### Usage

```
stroke.joinType
```

### Description

Property; a string that specifies the type of join for the stroke. Acceptable values are "miter", "round", and "bevel".

### See also

[stroke.capType](#)

## stroke.length

### Availability

Flash MX 2004.

**Usage**

```
stroke.length
```

**Description**

Property; a string that specifies the length of a hatch line. This property is available only if the `stroke.style` property is set to `hatched` (see [stroke.style](#)). Acceptable values are "equal", "slight variation", "medium variation", and "random". (The value "random" actually maps to "medium variation".)

**Example**

The following example sets the `length` property to `slight` for a stroke style of `hatched`:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.style = "hatched";
myStroke.curve = "straight";
myStroke.space = "close";
myStroke.jiggle = "wild";
myStroke.rotate = "free";
myStroke.length = "slight variation";
myStroke.hatchThickness = "thin";
fl.getDocumentDOM().setCustomStroke(myStroke);
```

## stroke.miterLimit

**Availability**

Flash 8.

**Usage**

```
stroke.miterLimit
```

**Description**

Property; a float value that specifies the angle above which the tip of the miter will be truncated by a segment. That means the miter is truncated only if the miter angle is greater than the value of `miterLimit`.

**Example**

The following example changes the miter limit of the stroke setting to 3. If the miter angle is greater than 3, the miter is truncated.

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.miterLimit = 3;
var myStroke = fl.getDocumentDOM().setCustomStroke();
```

## stroke.pattern

**Availability**

Flash MX 2004.

**Usage**

```
stroke.pattern
```

**Description**

Property; a string that specifies the pattern of a ragged line. This property is available only if the `stroke.style` property is set to `ragged` (see [stroke.style](#)). Acceptable values are "solid", "simple", "random", "dotted", "random dotted", "triple dotted", and "random triple dotted".

**Example**

The following example sets the `pattern` property to `random` for a stroke style of `ragged`:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();  
myStroke.style = "ragged";  
myStroke.pattern = "random";  
fl.getDocumentDOM().setCustomStroke(myStroke);
```

## stroke.rotate

**Availability**

Flash MX 2004.

**Usage**

```
stroke.rotate
```

**Description**

Property; a string that specifies the rotation of a hatch line. This property is available only if the `stroke.style` property is set to `hatched` (see [stroke.style](#)). Acceptable values are "none", "slight", "medium", and "free".

**Example**

The following example sets the `rotate` property to `free` for a style stroke of `hatched`:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();  
myStroke.style = "hatched";  
myStroke.curve = "straight";  
myStroke.space = "close";  
myStroke.jiggle = "wild";  
myStroke.rotate = "free";  
myStroke.length = "slight";  
myStroke.hatchThickness = "thin";
```

## stroke.scaleType

**Availability**

Flash 8.

**Usage**

```
stroke.scaleType
```

**Description**

Property; a string that specifies the type of scale to be applied to the stroke. Acceptable values are "normal", "horizontal", "vertical", and "none".

**Example**

The following example sets the scale type of the stroke to horizontal:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.scaleType = "horizontal";
fl.getDocumentDOM().setCustomStroke(myStroke);
```

## stroke.shapeFill

**Availability**

Flash 8.

**Usage**

```
stroke.shapeFill
```

**Description**

Property; a [Fill object](#) that represents the fill settings of the stroke.

**Example**

The following example specifies fill settings and then applies them to the stroke:

```
var fill = fl.getDocumentDOM().getCustomFill();
fill.linearGradient = true;
fill.colorArray = [ 00ff00, ff0000, ffffff ];
var stroke = fl.getDocumentDOM().getCustomStroke();
stroke.shapeFill = fill;
fl.getDocumentDOM().setCustomStroke(stroke);
```

## stroke.space

**Availability**

Flash MX 2004.

**Usage**

```
stroke.space
```

**Description**

Property; a string that specifies the spacing of a hatched line. This property is available only if the `stroke.style` property is set to `hatched` (see [stroke.style](#)). Acceptable values are "very close", "close", "distant", and "very distant".

**Example**

The following example sets the `space` property to `close` for a stroke style of `hatched`:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.style = "hatched";
myStroke.curve = "straight";
myStroke.space = "close";
myStroke.jiggle = "wild";
myStroke.rotate = "free";
myStroke.length = "slight";
myStroke.hatchThickness = "thin";
fl.getDocumentDOM().setCustomStroke(myStroke);
```

## stroke.strokeHinting

### Availability

Flash 8.

### Usage

```
stroke.strokeHinting
```

### Description

Property; a Boolean value that specifies whether stroke hinting is set on the stroke.

### Example

The following example enables stroke hinting for the stroke:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.strokeHinting = true;
fl.getDocumentDOM().setCustomStroke(myStroke);
```

## stroke.style

### Availability

Flash MX 2004.

### Usage

```
stroke.style
```

### Description

Property; a string that describes the stroke style. Acceptable values are "noStroke", "solid", "dashed", "dotted", "ragged", "stipple", and "hatched". Some of these values require additional properties of the Stroke object to be set, as described in the following list:

- If value is "solid" or "noStroke", there are no other properties.
- If value is "dashed", there are two additional properties: dash1 and dash2.
- If value is "dotted", there is one additional property: dotSpace.
- If value is "ragged", there are three additional properties: pattern, waveHeight, and waveLength.
- If value is "stipple", there are three additional properties: dotSize, variation, and density.

- If value is "hatched", there are six additional properties: hatchThickness, space, jiggle, rotate, curve, and length.

**Example**

The following example sets the stroke style to ragged:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.style = "ragged";
fl.getDocumentDOM().setCustomStroke(myStroke);
```

## stroke.thickness

**Availability**

Flash MX 2004.

**Usage**

```
stroke.thickness
```

**Description**

Property; an integer that specifies the stroke size.

**Example**

The following example sets the thickness property of the stroke to a value of 2:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.thickness = 2;
fl.getDocumentDOM().setCustomStroke(myStroke);
```

## stroke.variation

**Availability**

Flash MX 2004.

**Usage**

```
stroke.variation
```

**Description**

Property; a string that specifies the variation of a stippled line. This property is available only if the `stroke.style` property is set to `stipple` (see [stroke.style](#)). Acceptable values are "one size", "small variation", "varied sizes", and "random sizes".

**Example**

The following example sets the variation property to `random sizes` for a stroke style of `stipple`:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.style = "stipple";
myStroke.dotSpace= 3;
myStroke.variation = "random sizes";
myStroke.density = "sparse";
fl.getDocumentDOM().setCustomStroke(myStroke);
```

## stroke.waveHeight

### Availability

Flash MX 2004.

### Usage

```
stroke.waveHeight
```

### Description

Property; a string that specifies the wave height of a ragged line. This property is available only if the `stroke.style` property is set to `ragged` (see [stroke.style](#)). Acceptable values are "flat", "wavy", "very wavy", and "wild".

### Example

The following example sets the `waveHeight` property to `flat` for a stroke style of `ragged`:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.style = "ragged";
myStroke.pattern = "random";
myStroke.waveHeight = "flat";
myStroke.waveLength = "short";
fl.getDocumentDOM().setCustomStroke(myStroke);
```

## stroke.waveLength

### Availability

Flash MX 2004.

### Usage

```
stroke.waveLength
```

### Description

Property; a string that specifies the wavelength of a ragged line. This property is available only if the `stroke.style` property is set to `ragged` (see [stroke.style](#)). Acceptable values are "very short", "short", "medium", and "long".

### Example

The following example sets the `waveLength` property to `short` for a stroke style of `ragged`:

```
var myStroke = fl.getDocumentDOM().getCustomStroke();
myStroke.style = "ragged";
myStroke.pattern = "random";
myStroke.waveHeight = 'flat';
myStroke.waveLength = "short";
fl.getDocumentDOM().setCustomStroke(myStroke);
```

# Chapter 40: swfPanel object

## swfPanel summary

### Availability

Flash CS4 Professional.

### Description

The swfPanel object represents a Window SWF panel. Window SWF panels are SWF files that implement applications you can run from the Flash authoring environment; they are available from the Window > Other Panels menu. By default, Window SWF panels are stored in a subfolder of the Configuration folder (see “[Saving JSFL files](#)” on page 2). For example, on Windows XP, the folder is in *boot drive\Documents and Settings\user\Local Settings\Application Data\Adobe\Flash CS4\language\Configuration\WindowSWF*. A sample Window SWF panel is available; see “[Sample Trace Bitmap panel](#)” on page 17. The array of registered Window SWF panels is stored in the `f1.swfPanels` property.

### Method summary

You can use the following method with the swfPanel object:

Method	Description
<code>swfPanel.call()</code>	Works in conjunction with the ActionScript <code>ExternalInterface.addCallback()</code> and <code>MMExecute()</code> methods to communicate with the SWF panel from the authoring environment.
<code>swfPanel.reload()</code>	Reloads content in the SWF panel.
<code>swfPanel.setFocus()</code>	Sets the keyboard focus to the specified SWF panel.

### Property summary

You can use the following properties with the swfPanel object:

Property	Description
<code>swfPanel.dpiScaleFactorX</code>	A string that contains the DPI scale factor (scaleX) for swfPanel.
<code>swfPanel.dpiScaleFactorY</code>	A string that contains the DPI scale factor (scaleY) for swfPanel.
<code>swfPanel.name</code>	Read-only; a string that represents the name of the specified Window SWF panel.
<code>swfPanel.path</code>	Read-only; a string that represents the path to the SWF file used in the specified Window SWF panel.

## swfPanel.call()

### Availability

Flash CS4 Professional.

**Usage**

```
swfPanel.call(request)
```

**Parameters**

**request** Parameters to pass to the function (see “Description” and “Example” below).

**Returns**

Either `null` or a string that is returned by the function call. The function result could be an empty string.

**Description**

Method; works in conjunction with the ActionScript `ExternalInterface.addCallback()` and `MMExecute()` methods to communicate with the SWF panel from the authoring environment.

**Example**

The following example illustrates how to use ActionScript and JavaScript code to create a Window SWF panel and communicate with it from the authoring environment.

1 Create an ActionScript 3.0 FLA file, set its color to a medium gray, and set its size to 400 pixels wide and 250 pixels high.

2 Place a dynamic text box in the center of the Stage, set its Instance name to `myTextField`, and type the word "Status" in the text box.

3 Set other text box properties similar to the following:

- Center aligned
- 355 pixels wide and 46 pixels high
- Times New Roman font, 28 points, red

4 Add the following ActionScript code:

```
// Here's the callback function to be called from JSAPI
function callMeFromJavascript(arg:String):void
{
    try {
        var name:String = String(arg);
        myTextField.text = name;
    } catch (e:Error) {
    }
}

// Expose the callback function as "callMySWF"
ExternalInterface.addCallback("callMySWF", callMeFromJavascript);

// run the JSAPI to wire up the callback
MMExecute("fl.runScript( fl.configURI + \\\"WindowSWF/fileOp.jsfl\\\" );");

MMExecute("fl.trace(\"AS3 File Status Panel Initialized\");")
```

5 Save the file as `fileStatus.fla`, and publish the SWF file with the default Publish settings.

6 Close Flash.

7 Copy the `fileStatus.swf` file to the `WindowSWF` folder, which is a subfolder of the Configuration folder (see “[Saving JSFL files](#)” on page 2). For example, on Windows XP, the folder is in `boot drive\Documents and Settings\user\Local Settings\Application Data\Adobe\Flash CS4\language\Configuration\WindowSWF`.

**8** Start Flash.

**9** Create a JSFL file with the following code:

```
function callMyPanel(panelName, arg)
{
    if(f1.swfPanels.length > 0){
        for(x = 0; x < f1.swfPanels.length; x++){
            // look for a SWF panel of the specified name, then call the specified AS3
            function
            // in this example, the panel is named "test" and the AS3 callback is "callMySWF"
            if(f1.swfPanels[x].name == panelName) // name busted?
            {
                f1.swfPanels[x].call("callMySWF",arg);
                break;
            }
        }
    }
    else
        fl.trace("no panels");
}

// define the various handlers for events
documentClosedHandler = function () { callMyPanel("fileStatus", "Document Closed");};
f1.addEventListener("documentClosed", documentClosedHandler );

var dater = "New Document";
documentNewHandler = function () { callMyPanel("fileStatus", dater );};
f1.addEventListener("documentNew", documentNewHandler );

documentOpenedHandler = function () { callMyPanel("fileStatus", "Document Opened");};
f1.addEventListener("documentOpened", documentOpenedHandler );
```

**10** Save the JSFL file in the same directory as the SWF file, with the name fileOp.jsfl.

**11** Choose Window > Other panels > fileStatus.

Now, as you create, open, and close FLA files, the fileStatus panel displays a message indicating the action you have taken.

## swfPanel.dpiScaleFactorX

### Availability

Flash Professional CC.

### Usage

`swfPanel.dpiScaleFactorX`

### Description

Read-only property: a string that contains the DPI scale factor (scaleX) for swfPanel. Depending on this scale-factor, SwfPanel can adjust its contents.

**Example**

The following code illustrates the use of this property:

```
f1.trace(f1.swfPanel[x].dpiScaleFactorX);
```

**See also**

[swfPanel.dpiScaleFactorY](#)

## swfPanel.dpiScaleFactorY

**Availability**

Flash Professional CC.

**Usage**

```
swfPanel.dpiScaleFactorY
```

**Description**

Read-only property: a string that contains the DPI scale factor (scaleY) for swfPanel. Depending on this scale-factor, SwfPanel can adjust its contents.

**Example**

The following code illustrates the use of this property:

```
f1.trace(f1.swfPanel[x].dpiScaleFactorY);
```

**See also**

[swfPanel.dpiScaleFactorX](#)

## swfPanel.name

**Availability**

Flash CS4 Professional.

**Usage**

```
swfPanel.name
```

**Description**

Read-only property: a string that represents the name of the specified Window SWF panel.

**Example**

The following code displays the name of the first registered Window SWF panel in the Output panel:

```
f1.trace(f1.swfPanels[0].name);
```

**See also**

[swfPanel.path](#), [f1.swfPanels](#)

## swfPanel.path

### Availability

Flash CS4 Professional.

### Usage

`swfPanel.path`

### Description

Read-only property: a string that represents the path to the SWF file used in the specified Window SWF panel.

### Example

The following code displays the path of the SWF file used in the first registered Window SWF panel in the Output panel:

```
f1.trace(f1.swfPanels[0].path);
```

### See also

[swfPanel.name](#), [f1.swfPanels](#)

## swfPanel.reload()

### Availability

Flash Professional CC.

### Usage

`swfPanel.reload()`

### Description

Method: Reloads content in the SWF panel.

### Example

The following example illustrates the use of this method:

```
f1.swfPanels[0].reload();
```

## swfPanel.setFocus()

### Availability

Flash CS5.5 Professional.

### Usage

`swfPanel.setFocus()`

**Description**

Method: Sets the keyboard focus to the specified SWF panel.

**Example**

The following code sets the focus to the SWF panel named “Project”:

Please do the followings steps before running this command:

- 1 Undock the Project panel, so it is a floating panel.
- 2 Open the Create File dialog box from the Project panel and then click the Stage.
- 3 Press the Tab key a few times to ensure the Project panel does not have focus.
- 4 Run the script below from the Commands menu (put a JSFL file containing the code below in the user/config/Commands directory):
- 5 Press the tab key. You should see an insertion cursor in one of the text fields in the Create File dialog box.

```
flash.getSwfPanel("Project").setFocus();
```

**See also**

[swfPanel.name](#), [fl.swfPanels](#)

# Chapter 41: SymbolInstance object

## symbolInstance summary

**Inheritance** [Element object](#) > [Instance object](#) > SymbolInstance object

### Availability

Flash MX 2004.

### Description

SymbolInstance is a subclass of the Instance object and represents a symbol in a frame (see [Instance object](#)).

### Property summary

In addition to the Instance object properties, the SymbolInstance object has the following properties:

Property	Description
<code>symbolInstance.accName</code>	A string that is equivalent to the Name field in the Accessibility panel.
<code>symbolInstance.actionScript - dropped</code>	Dropped in Flash Professional CC.
<code>symbolInstance.backgroundColor</code>	A string specifying the matte color when Opaque is selected.
<code>symbolInstance.bitmapRenderMode</code>	A string specifying the display type for a symbol instance.
<code>symbolInstance.blendMode</code>	A string that specifies the blending mode to be applied to a movie clip symbol.
<code>symbolInstance.brightness</code>	An integer that returns the value set in the color effect Property Inspector for brightness (percentage between -100 and 100) when <code>colorMode == 'brightness'</code> .
<code>symbolInstance.buttonTracking</code>	A string (button symbols only) that sets the same property as the pop-up menu for Track as Button or Track As Menu Item in the Property inspector.
<code>symbolInstance.cacheAsBitmap</code>	A Boolean value that specifies whether run-time bitmap caching is enabled.
<code>symbolInstance.colorAlphaAmount</code>	An integer that is part of the color transformation for the instance, specifying the Advanced Effect Alpha settings; equivalent to using the Color > Advanced setting in the Property inspector and adjusting the controls on the right of the dialog box.
<code>symbolInstance.colorAlphaPercent</code>	An integer that specifies part of the color transformation for the instance; equivalent to using the Color > Advanced setting in the instance Property inspector (the percentage controls on the left of the dialog box).
<code>symbolInstance.colorBlueAmount</code>	An integer that is part of the color transformation for the instance; equivalent to using the Color > Advanced setting in the instance Property inspector.
<code>symbolInstance.colorBluePercent</code>	An integer that is part of the color transformation for the instance; equivalent to using the Color > Advanced setting in the instance Property inspector (the percentage controls on the left of the dialog box).
<code>symbolInstance.colorGreenAmount</code>	An integer that is part of the color transformation for the instance; equivalent to using the Color > Advanced setting in the instance Property inspector. Allowable values are from -255 to 255.

Property	Description
<code>symbolInstance.colorGreenPercent</code>	Part of the color transformation for the instance; equivalent to using the Color > Advanced setting in the instance Property inspector (the percentage controls on the left of the dialog box).
<code>symbolInstance.colorMode</code>	A string that specifies the color mode as identified in the symbol Property inspector Color pop-up menu.
<code>symbolInstance.colorRedAmount</code>	An integer that is part of the color transformation for the instance, equivalent to using the Color > Advanced setting in the instance Property inspector.
<code>symbolInstance.colorRedPercent</code>	Part of the color transformation for the instance; equivalent to using the Color > Advanced setting in the instance Property inspector (the percentage controls on the left of the dialog box).
<code>symbolInstance.description</code>	A string that is equivalent to the Description field in the Accessibility panel.
<code>symbolInstance.filters</code>	An array of Filter objects (see <a href="#">Filter object</a> ).
<code>symbolInstance.firstFrame</code>	A zero-based integer that specifies the first frame to appear in the timeline of the graphic.
<code>symbolInstance.forceSimple</code>	A Boolean value that enables and disables the accessibility of the object's children; equivalent to the inverse logic of the Make Child Objects Accessible setting in the Accessibility panel.
<code>symbolInstance.is3D</code>	A Boolean value indicating whether the instance contains any 3D transforms.
<code>symbolInstance.loop</code>	A string that, for graphic symbols, sets the same property as the Loop pop-up menu in the Property inspector.
<code>symbolInstance.shortcut</code>	A string that is equivalent to the shortcut key associated with the symbol; equivalent to the Shortcut field in the Accessibility panel.
<code>symbolInstance.silent</code>	A Boolean value that enables or disables the accessibility of the object; equivalent to the inverse logic of the Make Object Accessible setting in the Accessibility panel.
<code>symbolInstance.symbolType</code>	A string that specifies the type of symbol; equivalent to the value for Behavior in the Create New Symbol and Convert To Symbol dialog boxes.
<code>symbolInstance.tabIndex</code>	An integer that is equivalent to the Tab index field in the Accessibility panel.
<code>symbolInstance.tintColor</code>	When the Color Effect Property Inspector is using style tint (colorMode == 'tint'), return the color applied to the tint.
<code>symbolInstance.tintPercent</code>	When the color effect Property Inspector is using style tint (colorMode == 'tint') then return the applied to the tint percentage from -100 to 100.
<code>symbolInstance.useBackgroundColor</code>	A boolean value that specifies whether to use 24 bit mode or 32 bit mode with alpha for the instance.
<code>symbolInstance.visible</code>	A boolean value specifying whether the instance is visible or not.

## symbolInstance.accName

### Availability

Flash MX 2004.

### Usage

`symbolInstance.accName`

**Description**

Property; a string that is equivalent to the Name field in the Accessibility panel. Screen readers identify objects by reading the name aloud. This property is not available for graphic symbols.

**Example**

The following example stores the value for the Accessibility panel name of the object in the `theName` variable:

```
var theName = fl.getDocumentDOM().selection[0].accName;
```

The following example sets the value for the Accessibility panel name of the object to Home Button:

```
fl.getDocumentDOM().selection[0].accName = "Home Button";
```

## **symbolInstance.actionScript - dropped**

**Availability**

Flash MX 2004. *Dropped in Flash Professional CC.*

**Usage**

```
symbolInstance.actionScript
```

**Description**

*Dropped in Flash Professional CC.*

Property; a string that specifies the actions assigned to the symbol. This applies only to movie clip and button instances. For a graphic symbol instance, the value returns `undefined`.

**Example**

The following example assigns an `onClipEvent` action to the first item in the first frame of the first layer in the timeline:

```
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].actionScript  
= "onClipEvent(enterFrame) {trace('movie clip enterFrame');};"
```

## **symbolInstance.backgroundColor**

**Availability**

Flash CS5.5 Professional.

**Usage**

```
symbolInstance.backgroundColor
```

**Description**

Property; a string that specifies the matte color when 24 bit mode is selected for the instance. This is a string in hexadecimal #rrggbba format or an integer containing the value.

**Example**

The following example assigns the symbol instance a background color of black:

```
var bitmapInstance = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];
bitmapInstance.backgroundColor = "#000000";
```

## symbolInstance.bitmapRenderMode

### Availability

Flash CS5.5 Professional.

### Usage

```
symbolInstance.bitmapRenderMode
```

### Description

Property; a string that sets the display type for the symbol.

Acceptable values include:

- “none”
- “cache” - sets the symbol to be cached as a bitmap by Flash Player at runtime.
- “export” - sets the symbol to be exported as a bitmap when the SWF is compiled.

The older “[symbolInstance.cacheAsBitmap](#)” on page 472 property is similar to this property, but it offers fewer choices since it’s a boolean. In the future, the cacheAsBitmap property may be deprecated, so users should switch to this new property. The true/false options in the boolean cacheAsBitmap property are the same as the “cache” / “none” values for this new property.

### Example

The following example assigns the symbol’s bitmapRenderMode to “export”:

```
var symbol = fl.getDocumentDOM().selection[0];
fl.trace(symbol.bitmapRenderMode);
symbol.bitmapRenderMode = "export";
```

## symbolInstance.blendMode

### Availability

Flash 8.

### Usage

```
symbolInstance.blendMode
```

### Description

Property; a string that specifies the blending mode to be applied to a movie clip symbol. Acceptable values are “normal”, “layer”, “multiply”, “screen”, “overlay”, “hardlight”, “lighten”, “darker”, “difference”, “add”, “subtract”, “invert”, “alpha”, and “erase”.

### Example

The following example sets the blending mode for the first movie clip symbol in the first frame on the first level to add:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].blendMode = "add";
```

**See also**

[document.setBlendMode\(\)](#)

## symbolInstance.brightness

**Availability**

Flash Professional CC.

**Usage**

```
symbolInstance.brightness
```

**Description**

Read-only property; returns the value set in the color effect Property Inspector for brightness (percentage between -100 and 100) when `colorMode == 'brightness'`. Error if `colorMode` is a different setting.

**Example**

The following illustrates use of the `brightness` property:

```
var elem = f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];
if (elem.colorMode == 'brightness') {
    f1.trace(elem.brightness);
}
```

## symbolInstance.buttonTracking

**Availability**

Flash MX 2004.

**Usage**

```
symbolInstance.buttonTracking
```

**Description**

Property; a string that, for button symbols only, sets the same property as the pop-up menu for Track As Button or Track As Menu Item in the Property inspector. For other types of symbols, this property is ignored. Acceptable values are "button" or "menu".

**Example**

The following example sets the first symbol in the first frame of the first layer in the timeline to a Track As Menu Item, as long as that symbol is a button:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].buttonTracking = "menu";
```

## symbolInstance.cacheAsBitmap

### Availability

Flash 8.

### Usage

```
symbolInstance.cacheAsBitmap
```

### Description

Property; a Boolean value that specifies whether run-time bitmap caching is enabled.

**Note:** Starting w/ Flash Professional CS5.5, users should switch to using the "["symbolInstance.bitmapRenderMode"](#)" on page 470 property instead of this property.

### Example

The following example enables run-time bitmap caching for the first element in the first frame on the first layer:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].cacheAsBitmap = true;
```

## symbolInstance.colorAlphaAmount

### Availability

Flash MX 2004.

### Usage

```
symbolInstance.colorAlphaAmount
```

### Description

Property; an integer that is part of the color transformation for the instance, specifying the Advanced Effect Alpha settings. This property is equivalent to using the Color > Advanced setting in the Property inspector and adjusting the controls on the right of the dialog box. This value either reduces or increases the tint and alpha values by a constant amount. This value is added to the current value. This property is most useful if used with ["symbolInstance.colorAlphaPercent"](#). Allowable values are from -255 to 255.

### Example

The following example subtracts 100 from the alpha setting of the selected symbol instance:

```
f1.getDocumentDOM().selection[0].colorAlphaAmount = -100;
```

## symbolInstance.colorAlphaPercent

### Availability

Flash MX 2004.

### Usage

```
symbolInstance.colorAlphaPercent
```

**Description**

Property; an integer that specifies part of the color transformation for the instance. This property is equivalent to using the Color > Advanced setting in the instance Property inspector (the percentage controls on the left of the dialog box). This value changes the tint and alpha values to a specified percentage. Allowable values are from -100 to 100. See also [symbolInstance.colorAlphaAmount](#).

**Example**

The following example sets the `colorAlphaPercent` of the selected symbol instance to 80:

```
f1.getDocumentDOM().selection[0].colorAlphaPercent = 80;
```

## **symbolInstance.colorBlueAmount**

**Availability**

Flash MX 2004.

**Usage**

```
symbolInstance.colorBlueAmount
```

**Description**

Property; an integer that is part of the color transformation for the instance. This property is equivalent to using the Color > Advanced setting in the instance Property inspector. Allowable values are from -255 to 255.

## **symbolInstance.colorBluePercent**

**Availability**

Flash MX 2004.

**Usage**

```
symbolInstance.colorBluePercent
```

**Description**

Property; an integer that is part of the color transformation for the instance. This property is equivalent to using the Color > Advanced setting in the instance Property inspector (the percentage controls on the left of the dialog box). This value sets the blue values to a specified percentage. Allowable values are from -100 to 100.

**Example**

The following example sets the `colorBluePercent` of the selected symbol instance to 80:

```
f1.getDocumentDOM().selection[0].colorBluePercent = 80;
```

## symbolInstance.colorGreenAmount

### Availability

Flash MX 2004.

### Usage

```
symbolInstance.colorGreenAmount
```

### Description

Property; an integer that is part of the color transformation for the instance. This property is equivalent to using the Color > Advanced setting in the instance Property inspector. Allowable values are from -255 to 255.

## symbolInstance.colorGreenPercent

### Availability

Flash MX 2004.

### Usage

```
symbolInstance.colorGreenPercent
```

### Description

Property; part of the color transformation for the instance. This property is equivalent to using the Color > Advanced setting in the instance Property inspector (the percentage controls on the left of the dialog box). This value sets the green values by a specified percentage. Allowable values are from -100 to 100.

### Example

The following example sets the colorGreenPercent of the selected symbol instance to 70:

```
f1.getDocumentDOM().selection[0].colorGreenPercent = 70;
```

## symbolInstance.colorMode

### Availability

Flash MX 2004.

### Usage

```
symbolInstance.colorMode
```

### Description

Property; a string that specifies the color mode as identified in the symbol Property inspector Color pop-up menu. Acceptable values are "none", "brightness", "tint", "alpha", and "advanced".

**Example**

The following example changes the `colorMode` property of the first element in the first frame of the first layer in the timeline to alpha:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].colorMode = "alpha";
```

## **symbolInstance.colorRedAmount**

**Availability**

Flash MX 2004.

**Usage**

```
symbolInstance.colorRedAmount
```

**Description**

Property; an integer that is part of the color transformation for the instance. This property is equivalent to using the Color > Advanced setting in the instance Property inspector. Allowable values are from -255 to 255.

**Example**

The following example sets the `colorRedAmount` of the selected symbol instance to 255:

```
f1.getDocumentDOM().selection[0].colorRedAmount = 255;
```

## **symbolInstance.colorRedPercent**

**Availability**

Flash MX 2004.

**Usage**

```
symbolInstance.colorRedPercent
```

**Description**

Property; part of the color transformation for the instance. This property is equivalent to using the Color > Advanced setting in the instance Property inspector (the percentage controls on the left of the dialog box). This value sets the red values to a specified percentage. Allowable values are from -100 to 100.

**Example**

The following example sets the `colorRedPercent` of the selected symbol instance to 10:

```
f1.getDocumentDOM().selection[0].colorRedPercent = 10;
```

## **symbolInstance.description**

**Availability**

Flash MX 2004.

**Usage**

```
symbolInstance.description
```

**Description**

Property; a string that is equivalent to the Description field in the Accessibility panel. The description is read by the screen reader. This property is not available for graphic symbols.

**Example**

The following example stores the value for the Accessibility panel description of the object in the theDescription variable:

```
var theDescription = fl.getDocumentDOM().selection[0].description;
```

The following example sets the value for the Accessibility panel description to Click the home button to go to home:

```
fl.getDocumentDOM().selection[0].description= "Click the home button to go to home";
```

## **symbolInstance.filters**

**Availability**

Flash 8.

**Usage**

```
symbolInstance.filters
```

**Description**

Property; an array of Filter objects (see [Filter object](#)). To modify filter properties, you don't write to this array directly. Instead, retrieve the array, set the individual properties, and then set the array to reflect the new properties.

**Example**

The following example traces the name of the filter at index 0. If it is a Glow filter, its blurX property is set to 100 and the new value is written to the filters array.

```
var filterName =
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].filters[0].name;
fl.trace(filterName);
var filterArray = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].filters;
if (filterName == 'glowFilter'){
    filterArray[0].blurX = 100;
}
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].filters = filterArray;
```

## **symbolInstance.firstFrame**

**Availability**

Flash MX 2004.

**Usage**

```
symbolInstance.firstFrame
```

**Description**

Property; a zero-based integer that specifies the first frame to appear in the timeline of the graphic. This property applies only to graphic symbols and sets the same property as the First field in the Property inspector. For other types of symbols, this property is undefined.

**Example**

The following example specifies that Frame 10 should be the first frame to appear in the timeline of the specified element:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].firstFrame = 10;
```

## **symbolInstance.forceSimple**

**Availability**

Flash MX 2004.

**Usage**

```
symbolInstance.forceSimple
```

**Description**

Property; a Boolean value that enables and disables the accessibility of the object's children. This property is equivalent to the inverse logic of the Make Child Objects Accessible setting in the Accessibility panel. For example, if `forceSimple` is `true`, it is the same as the Make Child Object Accessible option being unchecked. If `forceSimple` is `false`, it is the same as the Make Child Object Accessible option being checked.

This property is available only for MovieClip objects.

**Example**

The following example checks to see if the children of the object are accessible; a return value of `false` means the children are accessible:

```
var areChildrenAccessible = f1.getDocumentDOM().selection[0].forceSimple;
```

The following example allows the children of the object to be accessible:

```
f1.getDocumentDOM().selection[0].forceSimple = false;
```

## **symbolInstance.is3D**

**Availability**

Flash Pro CS6.

**Usage**

```
symbolInstance.is3D
```

**Description**

Read-only property; a boolean value that indicates whether the symbol instance contains a 3D matrix (transform).

**Example**

The following example returns the value of the is3D property for the currently selected symbol instance on the Stage:

```
f1.trace("the instance contains a 3D matrix: " + f1.getDocumentDOM().selection[0].is3D);
```

## **symbolInstance.loop**

**Availability**

Flash MX 2004.

**Usage**

```
symbolInstance.loop
```

**Description**

Property; a string that, for graphic symbols, sets the same property as the Loop pop-up menu in the Property inspector. For other types of symbols, this property is undefined. Acceptable values are "loop", "play once", and "single frame" to set the graphic's animation accordingly.

**Example**

The following example sets the first symbol in the first frame of the first layer in the timeline to `single frame` (display one specified frame of the graphic timeline), as long as that symbol is a graphic:

```
f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].loop = 'single frame';
```

## **symbolInstance.shortcut**

**Availability**

Flash MX 2004.

**Usage**

```
symbolInstance.shortcut
```

**Description**

Property; a string that is equivalent to the shortcut key associated with the symbol. This property is equivalent to the Shortcut field in the Accessibility panel. This key is read by the screen readers. This property is not available for graphic symbols.

**Example**

The following example stores the value for the shortcut key of the object in the `theShortcut` variable:

```
var theShortcut = f1.getDocumentDOM().selection[0].shortcut;
```

The following example sets the shortcut key of the object to `Ctrl+i`:

```
f1.getDocumentDOM().selection[0].shortcut = "Ctrl+i";
```

## symbolInstance.silent

### Availability

Flash MX 2004.

### Usage

```
symbolInstance.silent
```

### Description

Property; a Boolean value that enables or disables the accessibility of the object. This property is equivalent to the inverse logic of the Make Object Accessible setting in the Accessibility panel. For example, if `silent` is `true`, it is the same as the Make Object Accessible option being unchecked. If `silent` is `false`, it is the same as the Make Object Accessible option being checked.

This property is not available for graphic objects.

### Example

The following example checks to see if the object is accessible; a return value of `false` means the object is accessible:

```
var isSilent = fl.getDocumentDOM().selection[0].silent;
```

The following example sets the object to be accessible:

```
fl.getDocumentDOM().selection[0].silent = false;
```

## symbolInstance.symbolType

### Availability

Flash MX 2004.

### Usage

```
symbolInstance.symbolType
```

### Description

Property; a string that specifies the type of symbol. This property is equivalent to the value for Behavior in the Create New Symbol and Convert To Symbol dialog boxes. Acceptable values are "button", "movie clip", and "graphic".

### Example

The following example sets the first symbol in the first frame of the first layer in the timeline of the current document to behave as a graphic symbol:

```
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].symbolType = "graphic";
```

## symbolInstance.tabIndex

### Availability

Flash MX 2004.

**Usage**

```
symbolInstance.tabIndex
```

**Description**

Property; an integer that is equivalent to the Tab index field in the Accessibility panel. Creates a tab order in which objects are accessed when the user presses the Tab key. This property is not available for graphic symbols.

**Example**

The following example sets the `tabIndex` property of the `mySymbol` object to 3 and displays that value in the Output panel:

```
var mySymbol = fl.getDocumentDOM().selection[0];
mySymbol.tabIndex = 3;
fl.trace(mySymbol.tabIndex);
```

## **symbolInstance.tintColor**

**Availability**

Flash Professional CC.

**Usage**

```
symbolInstance.tintColor
```

**Description**

Read-only property; when the Color Effect Property Inspector is using style tint (`colorMode == 'tint'`), return the color applied to the tint. Otherwise using this property results in an error.

**Example**

The following illustrates use of the `tintColor` property:

```
var elem = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];
if (elem.colorMode == 'tint') {
    fl.trace(elem.tintColor);
    fl.trace(elem.tintColorPercent);
}
```

## **symbolInstance.tintColorPercent**

**Availability**

Flash Professional CC.

**Usage**

```
symbolInstance.tintColorPercent
```

**Description**

Read-only property; when the Color Effect Property Inspector is using style tint (`colorMode == 'tint'`), then return the tint percentage from -100 to 100. Otherwise using this property results in an error.

**Example**

The following illustrates use of the `tintPercent` property:

```
var elem = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0];
if (elem.colorMode = 'tint') {
    fl.trace(elem.tintColor);
    fl.trace(elem.tintPercent);
}
```

## **symbolInstance.useBackgroundColor**

**Availability**

Flash CS5.5 Professional.

**Usage**

```
symbolInstance.useBackgroundColor
```

**Description**

Property; a boolean value that indicates whether to use 24 bit mode (true) or 32 bit mode with alpha (false) for the instance. If true, the `backgroundColor` specified for the instance is used.

**Example**

The following example sets the `useBackgroundColor` property of an instance to true:

```
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].useBackgroundColor = true
```

## **symbolInstance.visible**

**Availability**

Flash CS5.5 Professional.

**Usage**

```
symbolInstance.visible
```

**Description**

Property; a boolean value that sets the `Visible` property of an object to on (true) or off (false).

**Example**

The following example sets the visibility of the first item in the first frame of the first layer to false:

```
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].visible = false;
```

# Chapter 42: SymbolItem object

## symbolItem summary

**Inheritance**    [Item object](#) > SymbolItem object

### Availability

Flash MX 2004.

### Description

The SymbolItem object is a subclass of the [Item object](#).

### Method summary

In addition to the Item object methods, you can use the following methods with the SymbolItem object:

Method	Description
<code>symbolItem.convertToCompiledClip()</code>	Converts a symbol item in the library to a compiled movie clip.
<code>symbolItem.exportSWC()</code>	Exports the symbol item to a SWC file.
<code>symbolItem.exportSWF()</code>	Exports the symbol item to a SWF file.
<code>symbolItem.exportToLibrary()</code>	Export an instance to a new bitmap in the Library.
<code>symbolItem.exportToPNGSequence()</code>	Export a symbol to a sequence of PNG files.

### Property summary

In addition to the Item object properties, the following properties are available for the SymbolItem object:

Property	Description
<code>symbolItem.lastModifiedDate</code>	A string hexadecimal value that indicates the modification date of the symbol.
<code>symbolItem.scalingGrid</code>	A Boolean value that specifies whether 9-slice scaling is enabled for the item.
<code>symbolItem.scalingGridRect</code>	A Rectangle object that specifies the locations of the four 9-slice guides.
<code>symbolItem.sourceAutoUpdate</code>	A Boolean value that specifies whether the item is updated when the FLA file is published.
<code>symbolItem.sourceFilePath</code>	A string that specifies the path for the source FLA file as a file:/// URI.
<code>symbolItem.sourceLibraryName</code>	A string that specifies the name of the item in the source file library.
<code>symbolItem.symbolType</code>	A string that specifies the type of symbol.
<code>symbolItem.timeline</code>	Read-only; a <a href="#">Timeline object</a> .

## symbolItem.convertToCompiledClip()

### Availability

Flash MX 2004.

### Usage

```
symbolItem.convertToCompiledClip()
```

### Parameters

None.

### Returns

Nothing.

### Description

Method; converts a symbol item in the library to a compiled movie clip.

### Example

The following example converts an item in the library to a compiled movie clip:

```
f1.getDocumentDOM().library.items[3].convertToCompiledClip();
```

## symbolItem.exportSWC()

### Availability

Flash MX 2004.

### Usage

```
symbolItem.exportSWC(outputURI)
```

### Parameters

**outputURI** A string, expressed as a file:/// URI, that specifies the SWC file to which the method will export the symbol.  
The *outputURI* must reference a local file. Flash does not create a folder if *outputURI* does not exist.

### Returns

Nothing.

### Description

Method; exports the symbol item to a SWC file.

### Example

The following example exports an item in the library to the SWC file named `mySymbol.swc` in the tests folder:

```
f1.getDocumentDOM.library.selectItem("mySymbol");
var currentSelection = f1.getDocumentDOM().library.getSelectedItems();
currentSelection[0].exportSWC("file:///Macintosh HD/SWCDirectory/mySymbol.swc");
```

## symbolItem.exportSWF()

### Availability

Flash MX 2004.

### Usage

```
symbolItem.exportSWF(outputURI)
```

### Parameters

**outputURI** A string, expressed as a file:/// URI, that specifies the SWF file to which the method will export the symbol. The *outputURI* must reference a local file. Flash does not create a folder if *outputURI* doesn't exist.

### Returns

Nothing.

### Description

Method; exports the symbol item to a SWF file.

### Example

The following example exports an item in the library to the `my.swf` file in the tests folder:

```
f1.getDocumentDOM().library.items[0].exportSWF("file:///c|/tests/my.swf");
```

## symbolItem.exportToLibrary()

### Availability

Flash Pro CS6.

### Usage

```
symbolItem.exportToLibrary(frameNumber, bitmapName)
```

### Parameters

**frameNumber** An integer indicating the frame within the symbol to be exported.

**bitmapName** A string indicating the name of the new bitmap to be added to the Library.

### Returns

Nothing.

### Description

Method; exports a frame from the selected instance of movie clip, graphic, or button symbol on the Stage to a bitmap in Library.

### Example

The following example exports the first frame of the currently selected symbol instance to a new bitmap in the library that will be called “mytestBitmap”:

```
f1.getDocumentDOM().library.item[0].exportToLibrary(1, "mytestBitmap");
```

## **symbolItem.exportToPNGSequence()**

### **Availability**

Flash Pro CS6.

### **Usage**

```
symbolItem.exportToPNGSequence(outputURI [, startFrameNum ] [, endFrameNum ] [, matrix])
```

### **Parameters**

**outputURI** The URI to export the PNG sequence files to. This URI must reference a local file. For example:  
file:///c|/tests/mytest.png.

**startFrameNum** An integer indicating the first frame within the symbol to be exported. If this parameter is omitted, all frames are exported.

**endFrameNum** An integer indicating the last frame within the symbol to be exported. If this parameter is omitted, all frames are exported.

**matrix** Optional. A matrix to be appended to the exported PNG sequence.

### **Returns**

Nothing.

### **Description**

Method; exports a movie clip, graphic, or button symbol to a sequence of PNG files on disk.

### **Example**

The following example exports the first symbol in the Library new sequence of numbered PNG files starting with the filename“myTest.png”:

```
f1.getDocumentDOM().library.items[0].exportToPNGSequence("file:///c|/tests/mytest.png");
```

## **symbolItem.lastModifiedDate**

### **Availability**

Flash Pro CS6.

### **Usage**

```
symbolItem.lastModifiedDate
```

### **Description**

Read-only property; a string that indicates the modification date of the symbol as a hexadecimal value, representing a date and time. This value is incremented every time a symbol's timeline is edited.

**Example**

The following example returns the hexidecimal modification date of the first symbol in the Library:

```
var item = fl.getDocumentDOM().library.items[0];
fl.trace("name: " + item.name + ", date: " + item.lastModifiedDate);
// name: Symbol 1, date: 4f273915
```

## **symbolItem.scalingGrid**

**Availability**

Flash 8.

**Usage**

```
symbolItem.scalingGrid
```

**Description**

Property; a Boolean value that specifies whether 9-slice scaling is enabled for the item.

**Example**

The following example enables 9-slice scaling for an item in the library:

```
fl.getDocumentDOM().library.items[0].scalingGrid = true;
```

**See also**

[symbolItem.scalingGridRect](#)

## **symbolItem.scalingGridRect**

**Availability**

Flash 8.

**Usage**

```
symbolItem.scalingGridRect
```

**Description**

Property; a Rectangle object that specifies the locations of the four 9-slice guides. For information on the format of the rectangle, see [document.addNewRectangle\(\)](#).

**Example**

The following example specifies the locations of the 9-slice guides:

```
fl.getDocumentDOM().library.items[0].scalingGridRect = {left:338, top:237, right:3859,
bottom:713};
```

**See also**

[symbolItem.scalingGrid](#)

## **symbolItem.sourceAutoUpdate**

### **Availability**

Flash MX 2004.

### **Usage**

```
symbolItem.sourceAutoUpdate
```

### **Description**

Property; a Boolean value that specifies whether the item is updated when the FLA file is published. The default value is `false`. Used for shared library symbols.

### **Example**

The following example sets the `sourceAutoUpdate` property for a library item:

```
f1.getDocumentDOM().library.items[0].sourceAutoUpdate = true;
```

## **symbolItem.sourceFilePath**

### **Availability**

Flash MX 2004.

### **Usage**

```
symbolItem.sourceFilePath
```

### **Description**

Property; a string that specifies the path for the source FLA file as a file:/// URI. The path must be an absolute path, not a relative path. This property is used for shared library symbols.

### **Example**

The following example shows the value of the `sourceFilePath` property in the Output panel:

```
f1.trace(f1.getDocumentDOM().library.items[0].sourceFilePath);
```

## **symbolItem.sourceLibraryName**

### **Availability**

Flash MX 2004.

### **Usage**

```
symbolItem.sourceLibraryName
```

### **Description**

Property; a string that specifies the name of the item in the source file library. It is used for shared library symbols.

**Example**

The following example shows the value of the `sourceLibraryName` property in the Output panel:

```
f1.trace(f1.getDocumentDOM().library.items[0].sourceLibraryName);
```

## symbolItem.symbolType

**Availability**

Flash MX 2004.

**Usage**

```
symbolItem.symbolType
```

**Description**

Property; a string that specifies the type of symbol. Acceptable values are "movie clip", "button", and "graphic".

**Example**

The following example shows the current value of the `symbolType` property, changes it to `button`, and shows it again:

```
alert(f1.getDocumentDOM().library.items[0].symbolType);
f1.getDocumentDOM().library.items[0].symbolType = "button";
alert(f1.getDocumentDOM().library.items[0].symbolType);
```

## symbolItem.timeline

**Availability**

Flash MX 2004.

**Usage**

```
symbolItem.timeline
```

**Description**

Read-only property; a [Timeline object](#).

**Example**

The following example obtains and shows the number of layers that the selected movie clip in the library contains:

```
var tl = f1.getDocumentDOM().library.getSelectedItems()[0].timeline;
alert(tl.layerCount);
```

# Chapter 43: Text object

## text summary

**Inheritance** [Element object](#) > Text object

### Availability

Flash MX 2004.

### Description

The Text object represents a single text item in a document. All properties of the text pertain to the entire text block.

To set properties of a text run within the text field, see the Property summary for the [TextAttrs object](#). To change properties of a selection within a text field, you can use `document.setElementTextAttr()` and specify a range of text, or use the current selection.

To set generic properties of the selected text field, use `document.setProperty()`. The following example sets the `x` value of the selected text field's registration point to 50:

```
f1.getDocumentDOM().setProperty("x", 50);
```

### Method summary

In addition to the Element object methods, the following methods are available for the Text object:

Method	Description
<code>text.getTextAttr()</code>	Retrieves the specified attribute for the text identified by the optional <code>startIndex</code> and <code>endIndex</code> parameters.
<code>text.getTextString()</code>	Retrieves the specified range of text.
<code>text.setTextAttr()</code>	Sets the specified attribute associated with the text identified by <code>startIndex</code> and <code>endIndex</code> .
<code>text.setTextString()</code>	Changes the text string within this Text object.

### Property summary

In addition to the Element object properties, the following properties are available for the Text object:

Property	Description
<code>text.accName</code>	A string that is equivalent to the Name field in the Accessibility panel.
<code>text.antiAliasSharpness</code>	A float value that specifies the anti-aliasing sharpness of the text.
<code>text.antiAliasThickness</code>	A float value that specifies the anti-aliasing thickness of the text.
<code>text.autoExpand</code>	A Boolean value that controls the expansion of the bounding width for static text fields or the bounding width and height for dynamic or input text.
<code>text.border</code>	A Boolean value that controls whether Flash shows ( <code>true</code> ) or hides ( <code>false</code> ) a border around dynamic or input text.
<code>text.description</code>	A string that is equivalent to the Description field in the Accessibility panel.

Property	Description
<code>text.embeddedCharacters</code>	A string that specifies characters to embed. This is equivalent to entering text in the Character Embedding dialog box.
<code>text.embedRanges</code>	A string that consists of delimited integers that correspond to the items that can be selected in the Character Embedding dialog box.
<code>text.embedVariantGlyphs</code>	A Boolean value that specifies whether to enable the embedding of variant glyphs.
<code>text.filters</code>	An array of filters applied to the text element
<code>text.fontRenderingMode</code>	A string that specifies the rendering mode for the text.
<code>text.length</code>	Read-only; an integer that represents the number of characters in the Text object.
<code>text.lineType</code>	A string that sets the line type to "single line", "multiline", "multiline no wrap", or "password".
<code>text.maxCharacters</code>	An integer that specifies the maximum characters the user can enter into this Text object.
<code>text.orientation</code>	A string that specifies the orientation of the text field.
<code>text.renderAsHTML</code>	A Boolean value that controls whether Flash draws the text as HTML and interprets embedded HTML tags.
<code>text.scrollable</code>	A Boolean value that controls whether the text can ( <code>true</code> ) or cannot ( <code>false</code> ) be scrolled.
<code>text.selectable</code>	A Boolean value that controls whether the text can ( <code>true</code> ) or cannot ( <code>false</code> ) be selected. Input text is always selectable.
<code>text.selectionEnd</code>	A zero-based integer that specifies the offset of the end of a text subselection.
<code>text.selectionStart</code>	A zero-based integer that specifies the offset of the beginning of a text subselection.
<code>text.shortcut</code>	A string that is equivalent to the Shortcut field in the Accessibility panel.
<code>text.silent</code>	A Boolean value that specifies whether the object is accessible.
<code>text.tabIndex</code>	An integer that is equivalent to the Tab Index field in the Accessibility panel.
<code>text.textRuns</code>	Read-only; an array of TextRun objects.
<code>text.textType</code>	A string that specifies the type of text field. Acceptable values are "static", "dynamic", and "input".
<code>text.useDeviceFonts</code>	A Boolean value. A value of <code>true</code> causes Flash to draw text using device fonts.
<code>text.variableName</code>	A string that contains the contents of the Text object.

## text.accName

### Availability

Flash MX 2004.

### Usage

`text.accName`

### Description

Property; a string that is equivalent to the Name field in the Accessibility panel. Screen readers identify objects by reading the name aloud. This property cannot be used with dynamic text.

**Example**

The following example retrieves the name of the object:

```
var doc = fl.getDocumentDOM();
var theName = doc.selection[0].accName;
```

The following example sets the name of the currently selected object:

```
fl.getDocumentDOM().selection[0].accName = "Home Button";
```

## text.antiAliasSharpness

**Availability**

Flash 8.

**Usage**

```
text.antiAliasSharpness
```

**Description**

Property; a float value that specifies the anti-aliasing sharpness of the text. This property controls how crisply the text is drawn; higher values specify sharper (or crisper) text. A value of 0 specifies normal sharpness. This property is available only if `text.fontRenderingMode` is set to `customThicknessSharpness`.

**Example**

See [text.fontRenderingMode](#).

**See also**

[text.antiAliasThickness](#), [text.fontRenderingMode](#)

## text.antiAliasThickness

**Availability**

Flash 8.

**Usage**

```
text.antiAliasThickness
```

**Description**

Property; a float value that specifies the anti-aliasing thickness of the text. This property controls how thickly the text is drawn, with higher values specifying thicker text. A value of 0 specifies normal thickness. This property is available only if `text.fontRenderingMode` is set to `customThicknessSharpness`.

**Example**

See [text.fontRenderingMode](#).

**See also**

[text.antiAliasSharpness](#), [text.fontRenderingMode](#)

## text.autoExpand

**Availability**

Flash MX 2004.

**Usage**

`text.autoExpand`

**Description**

Property; a Boolean value. For static text fields, a value of `true` causes the bounding width to expand to show all text. For dynamic or input text fields, a value of `true` causes the bounding width and height to expand to show all text.

**Example**

The following example sets the `autoExpand` property to a value of `true`:

```
f1.getDocumentDOM().selection[0].autoExpand = true;
```

## text.border

**Availability**

Flash MX 2004.

**Usage**

`text.border`

**Description**

Property; a Boolean value. A value of `true` causes Flash to show a border around text.

**Example**

The following example sets the `border` property to a value of `true`:

```
f1.getDocumentDOM().selection[0].border = true;
```

## text.description

**Availability**

Flash MX 2004.

**Usage**

`text.description`

**Description**

Property; a string that is equivalent to the Description field in the Accessibility panel. The description is read by the screen reader.

**Example**

The following example retrieves the description of the object:

```
var doc = fl.getDocumentDOM();
var desc = doc.selection[0].description;
```

The following example sets the description of the object:

```
var doc = fl.getDocumentDOM();
doc.selection[0].description= "Enter your name here";
```

## text.embeddedCharacters

**Availability**

Flash MX 2004.

**Usage**

```
text.embeddedCharacters
```

**Description**

Property; a string that specifies characters to embed. This is equivalent to entering text in the Character Embedding dialog box.

This property works only with dynamic or input text; it generates a warning if used with other text types.

**Note:** Beginning in Flash Professional CS5, font embedding is controlled at the document level instead of the text object level. Use the “[fontItem.embeddedCharacters](#)” on page 310 property instead of the text.embeddedCharacters property.

**Example**

The following example assumes that the first or only item in the current selection is a classic text object and sets the embeddedCharacters property to abc:

```
fl.getDocumentDOM().selection[0].embeddedCharacters = "abc";
```

## text.embedRanges

**Availability**

Flash MX 2004.

**Usage**

```
text.embedRanges
```

### Description

Property; a string that consists of delimited integers that correspond to the items that can be selected in the Character Embedding dialog box. This property works only with dynamic or input text; it is ignored if used with static text.

This property corresponds to the XML file in the Configuration/Font Embedding folder.

**Note:** Beginning in Flash Professional CS5, font embedding is controlled at the document level instead of the text object level. Use the “[fontItem.embedRanges](#)” on page 311 property instead of the text.embedRanges property.

### Example

The following example assumes that the first or only item in the current selection is a classic text object and sets the embedRanges property to "1|3|7":

```
var doc = fl.getDocumentDOM();
doc.selection[0].embedRanges = "1|3|7";
```

The following example resets the property:

```
var doc = fl.getDocumentDOM();
doc.selection[0].embedRanges = "";
```

## text.embedVariantGlyphs

### Availability

Flash CS4 Professional.

### Usage

```
text.embedVariantGlyphs
```

### Description

Property; a Boolean value that specifies whether to enable the embedding of variant glyphs (`true`) or not (`false`). This property works only with dynamic or input text; it is ignored if used with static text. The default value is `false`.

**Note:** Beginning in Flash Professional CS5, font embedding is controlled at the document level instead of the text object level. Use the “[fontItem.embedVariantGlyphs](#)” on page 311 property instead of the text.embedVariantGlyphs property. In Flash Professional CS5, the text.embedVariantGlyphs property no longer has any effect because Flash always embeds variant glyphs for TLF text and never embeds them for Classic text.

### Example

The following example enables variant glyphs to be embedded in the selected Text object:

```
fl.getDocumentDOM().selection[0].embedVariantGlyphs = true;
```

### See also

[fontItem.embedVariantGlyphs](#)

## text.filters

### Availability

Flash Professional CS6.

### Usage

```
text.filters
```

### Description

Property; an array of filters applied to the text element. To modify filter properties, you don't write to this array directly. Instead, retrieve the array, set the individual properties, and then set the array to reflect the new properties.

### Example

The following example traces the name of the filter at index 0. If it is a Glow filter, its blurX property is set to 100 and the new value is written to the filters array:

```
//trace the name of the filter at index 0, if it's glow filter, set its blurX to 100
var filterName =
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].filters[0].name;
fl.trace(filterName);
var filterArray = fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].filters;
if (filterName == 'glowFilter')
{
    filterArray[0].blurX = 100;
}
fl.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].filters = filterArray;
```

## text.fontRenderingMode

### Availability

Flash 8.

### Usage

```
text.fontRenderingMode
```

### Description

Property; a string that specifies the rendering mode for the text. This property affects how the text is displayed both on the Stage and in Flash Player. Acceptable values are described in the following table:

Property value	How text is rendered
device	Renders the text with device fonts.
bitmap	Renders aliased text as a bitmap, or as a pixel font would.

Property value	How text is rendered
standard	Renders text using the standard anti-aliasing method used by Flash MX 2004. This is the best setting to use for animated, very large, or skewed text.
advanced	Renders text using the advanced anti-aliasing font rendering technology implemented in Flash 8, which produces better anti-aliasing and improves readability, especially for small text.
customThicknessSharpness	Lets you specify custom settings for the sharpness and thickness of the text when using the advanced anti-aliasing font rendering technology implemented in Flash 8.

### Example

The following example shows how you can use the `customThicknessSharpness` value to specify the sharpness and thickness of the text:

```
f1.getDocumentDOM().setElementProperty("fontRenderingMode", "customThicknessSharpness");
f1.getDocumentDOM().setElementProperty("antiAliasSharpness", 400);
f1.getDocumentDOM().setElementProperty("antiAliasThickness", -200);
```

### See also

[text.antiAliasSharpness](#), [text.antiAliasThickness](#)

## text.getTextAttr()

### Availability

Flash MX 2004.

### Usage

```
text.getTextAttr(attrName [, startIndex [, endIndex]])
```

### Parameters

**attrName** A string that specifies the name of the `TextAttrs` object property to be returned. For a list of possible values for `attrName`, see the Property summary for the [TextAttrs object](#).

**startIndex** An integer that is the index of first character. This parameter is optional.

**endIndex** An integer that specifies the end of the range of text, which starts with `startIndex` and goes up to, but does not include, `endIndex`. This parameter is optional.

### Returns

The value of the attribute specified in the `attrName` parameter.

### Description

Method; retrieves the attribute specified by the `attrName` parameter for the text identified by the optional `startIndex` and `endIndex` parameters. If the attribute is not consistent for the specified range, Flash returns `undefined`. If you omit the optional parameters `startIndex` and `endIndex`, the method uses the entire text range. If you specify only `startIndex`, the range used is a single character at that position. If you specify both `startIndex` and `endIndex`, the range starts from `startIndex` and goes up to, but does not include, `endIndex`.

### Example

The following example gets the font size of the currently selected text field and shows it:

```
var TheTextSize = fl.getDocumentDOM().selection[0].getTextAttr("size");
fl.trace(TheTextSize);
```

The following example gets the text fill color of the selected text field:

```
var TheFill = fl.getDocumentDOM().selection[0].getTextAttr("fillColor");
fl.trace(TheFill);
```

The following example gets the size of the third character:

```
var Char3 = fl.getDocumentDOM().selection[0].getTextAttr("size", 2);
fl.trace(Char3);
```

The following example gets the color of the selected text field from the third through the eighth character:

```
fl.getDocumentDOM().selection[0].getTextAttr("fillColor", 2, 8);
```

## text.getTextString()

### Availability

Flash MX 2004.

### Usage

```
text.getTextString([startIndex [, endIndex]])
```

### Parameters

**startIndex** An integer that specifies the index (zero-based) of the first character. This parameter is optional.

**endIndex** An integer that specifies the end of the range of text, which starts from *startIndex* and goes up to, but does not include, *endIndex*. This parameter is optional.

### Returns

A string of the text in the specified range.

### Description

Method; retrieves the specified range of text. If you omit the optional parameters *startIndex* and *endIndex*, the whole text string is returned. If you specify only *startIndex*, the method returns the string starting at the index location and ending at the end of the field. If you specify both *startIndex* and *endIndex*, the method returns the string starting from *startIndex* and goes up to, but does not include, *endIndex*.

### Example

The following example gets the character(s) from the fifth character through the end of the selected text field:

```
var myText = fl.getDocumentDOM().selection[0].getTextString(4);
fl.trace(myText);
```

The following example gets the fourth through the ninth characters starting in the selected text field:

```
var myText = fl.getDocumentDOM().selection[0].getTextString(3, 9);
fl.trace(myText);
```

## text.length

### Availability

Flash MX 2004.

### Usage

```
text.length
```

### Description

Read-only property; an integer that represents the number of characters in the Text object.

### Example

The following example returns the number of characters in the selected text:

```
var textLength = fl.getDocumentDOM().selection[0].length;
```

## text.lineType

### Availability

Flash MX 2004.

### Usage

```
text.lineType
```

### Description

Property; a string that sets the line type. Acceptable values are "single line", "multiline", "multiline no wrap", and "password".

This property works only with dynamic or input text and generates a warning if used with static text. The "password" value works only for input text.

### Example

The following example sets the lineType property to the value multiline no wrap:

```
fl.getDocumentDOM().selection[0].lineType = "multiline no wrap";
```

## text.maxCharacters

### Availability

Flash MX 2004.

### Usage

```
text.maxCharacters
```

### Description

Property; an integer that specifies the maximum number of characters the user can enter in this Text object.

This property works only with input text; if used with other text types, the property generates a warning.

**Example**

The following example sets the value of the `maxCharacters` property to 30:

```
f1.getDocumentDOM().selection[0].maxCharacters = 30;
```

## text.orientation

**Availability**

Flash MX 2004.

**Usage**

```
text.orientation
```

**Description**

Property; a string that specifies the orientation of the text field. Acceptable values are "horizontal", "vertical left to right", and "vertical right to left".

This property works only with static text; it generates a warning if used with other text types.

**Example**

The following example sets the orientation property to vertical right to left:

```
f1.getDocumentDOM().selection[0].orientation = "vertical right to left";
```

## text.renderAsHTML

**Availability**

Flash MX 2004.

**Usage**

```
text.renderAsHTML
```

**Description**

Property; a Boolean value. If the value is `true`, Flash draws the text as HTML and interprets embedded HTML tags.

This property works only with dynamic or input text; it generates a warning if used with other text types.

**Example**

The following example sets the `renderAsHTML` property to `true`:

```
f1.getDocumentDOM().selection[0].renderAsHTML = true;
```

## text.scrollable

### Availability

Flash MX 2004.

### Usage

```
text.scrollable
```

### Description

Property; a Boolean value. If the value is `true`, the text can be scrolled.

This property works only with dynamic or input text; it generates a warning if used with static text.

### Example

The following example sets the `scrollable` property to `false`:

```
f1.getDocumentDOM().selection[0].scrollable = false;
```

## text.selectable

### Availability

Flash MX 2004.

### Usage

```
text.selectable
```

### Description

Property; a Boolean value. If the value is `true`, the text can be selected.

Input text is always selectable. Flash generates a warning when this property is set to `false` and used with input text.

### Example

The following example sets the `selectable` property to `true`:

```
f1.getDocumentDOM().selection[0].selectable = true;
```

## text.selectionEnd

### Availability

Flash MX 2004.

### Usage

```
text.selectionEnd
```

**Description**

Property; a zero-based integer that specifies the end of a text subselection. For more information, see [text.selectionStart](#).

## text.selectionStart

**Availability**

Flash MX 2004.

**Usage**

```
text.selectionStart
```

**Description**

Property; a zero-based integer that specifies the beginning of a text subselection. You can use this property with `text.selectionEnd` to select a range of characters. Characters up to, but not including, `text.selectionEnd` are selected. See [text.selectionEnd](#).

- If there is an insertion point or no selection, `text.selectionEnd` is equal to `text.selectionStart`.
- If `text.selectionStart` is set to a value greater than `text.selectionEnd`, `text.selectionEnd` is set to `text.selectionStart`, and no text is selected.

**Example**

The following example sets the start of the text subselection to the sixth character:

```
f1.getDocumentDOM().selection[0].selectionStart = 5;
```

The following example selects the characters Barbara from a text field that contains the text My name is Barbara and formats them as bold and green:

```
f1.getDocumentDOM().selection[0].selectionStart = 11;
f1.getDocumentDOM().selection[0].selectionEnd = 18;
var s = f1.getDocumentDOM().selection[0].selectionStart;
var e = f1.getDocumentDOM().selection[0].selectionEnd;
f1.getDocumentDOM().setElementTextAttr('bold', true, s, e);
f1.getDocumentDOM().setElementTextAttr("fillColor", "#00ff00", s, e);
```

## text.setTextAttr()

**Availability**

Flash MX 2004.

**Usage**

```
text.setTextAttr(attrName, attrValue [, startIndex [, endIndex]])
```

**Parameters**

**attrName** A string that specifies the name of the TextAttrs object property to change.

**attrValue** The value for the TextAttrs object property.

For a list of possible values for `attrName` and `attrValue`, see the Property summary for the [TextAttrs object](#).

**startIndex** An integer that is the index (zero-based) of the first character in the array. This parameter is optional.

**endIndex** An integer that specifies the index of the end point in the selected text string, which starts at `startIndex` and goes up to, but does not include, `endIndex`. This parameter is optional.

### Returns

Nothing.

### Description

Method; sets the attribute specified by the `attrName` parameter associated with the text identified by `startIndex` and `endIndex` to the value specified by `attrValue`. This method can be used to change attributes of text that might span TextRun elements (see [TextRun object](#)), or that are portions of existing TextRun elements. Using it may change the position and number of TextRun elements within this object's `text.textRuns` array (see [text.textRuns](#)).

If you omit the optional parameters, the method uses the entire Text object's character range. If you specify only `startIndex`, the range is a single character at that position. If you specify both `startIndex` and `endIndex`, the range starts from `startIndex` and goes up to, but does not include, the character located at `endIndex`.

### Example

The following example sets the selected text field to italic:

```
f1.getDocumentDOM().selection[0].setTextAttr("italic", true);
```

The following example sets the size of the third character to 10:

```
f1.getDocumentDOM().selection[0].setTextAttr("size", 10, 2);
```

The following example sets the color to red for the third through the eighth character of the selected text:

```
f1.getDocumentDOM().selection[0].setTextAttr("fillColor", 0xff0000, 2, 8);
```

## text.setTextString()

### Availability

Flash MX 2004.

### Usage

```
text.setTextString(text [, startIndex [, endIndex]])
```

### Parameters

**text** A string that consists of the characters to be inserted into this Text object.

**startIndex** An integer that specifies the index (zero-based) of the character in the string where the text will be inserted. This parameter is optional.

**endIndex** An integer that specifies the index of the end point in the selected text string. The new text overwrites the text from `startIndex` up to, but not including, `endIndex`. This parameter is optional.

### Returns

Nothing.

### Description

Property; changes the text string within this Text object. If you omit the optional parameters, the whole Text object is replaced. If you specify only *startIndex*, the specified string is inserted at the *startIndex* position. If you specify both *startIndex* and *endIndex*, the specified string replaces the segment of text starting from *startIndex* up to, but not including, *endIndex*.

### Example

The following example assigns the string `this is a string` to the selected text field:

```
fl.getDocumentDOM().selection[0].setTextString("this is a string");
```

The following example inserts the string `abc` beginning at the fifth character of the selected text field:

```
fl.getDocumentDOM().selection[0].setTextString("01234567890");
fl.getDocumentDOM().selection[0].setTextString("abc", 4);
// text field is now "0123abc4567890"
```

The following example replaces the text from the third through the eighth character of the selected text string with the string `abcdefghijklm`. Characters between *startIndex* and *endIndex* are overwritten. Characters beginning with *endIndex* follow the inserted string.

```
fl.getDocumentDOM().selection[0].setTextString("01234567890");
fl.getDocumentDOM().selection[0].setTextString("abcdefghijklm", 2, 8);
// text field is now "01abcdefghijklm890"
```

## text.shortcut

### Availability

Flash MX 2004.

### Usage

```
text.shortcut
```

### Description

Property; a string that is equivalent to the Shortcut field in the Accessibility panel. The shortcut is read by the screen reader. This property cannot be used with dynamic text.

### Example

The following example gets the shortcut key of the selected object and shows the value:

```
var theShortcut = fl.getDocumentDOM().selection[0].shortcut;
fl.trace(theShortcut);
```

The following example sets the shortcut key of the selected object:

```
fl.getDocumentDOM().selection[0].shortcut = "Ctrl+i";
```

## text.silent

### Availability

Flash MX 2004.

### Usage

```
text.silent
```

### Description

Property; a Boolean value that specifies whether the object is accessible. This is equivalent to the inverse logic of the Make Object Accessible setting in the Accessibility panel. That is, if `silent` is `true`, Make Object Accessible is deselected. If it is `false`, Make Object Accessible is selected.

### Example

The following example determines if the object is accessible (a value of `false` means that it is accessible):

```
var isSilent = fl.getDocumentDOM().selection[0].silent;
```

The following example sets the object to be accessible:

```
fl.getDocumentDOM().selection[0].silent = false;
```

## text.tabIndex

### Availability

Flash MX 2004.

### Usage

```
text.tabIndex
```

### Description

Property; an integer that is equivalent to the Tab Index field in the Accessibility panel. This value lets you determine the order in which objects are accessed when the user presses the Tab key.

### Example

The following example gets the `tabIndex` of the currently selected object:

```
var theTabIndex = fl.getDocumentDOM().selection[0].tabIndex;
```

The following example sets the `tabIndex` of the currently selected object:

```
fl.getDocumentDOM().selection[0].tabIndex = 1;
```

## text.textRuns

### Availability

Flash MX 2004.

**Usage**

```
text.textRuns
```

**Description**

Read-only property; an array of TextRun objects (see [TextRun object](#)).

**Example**

The following example stores the value of the `textRuns` property in the `myTextRuns` variable:

```
var myTextRuns = fl.getDocumentDOM().selection[0].textRuns;
```

## text.textType

**Availability**

Flash MX 2004.

**Usage**

```
text.textType
```

**Description**

Property; a string that specifies the type of text field. Acceptable values are "static", "dynamic", and "input".

**Example**

The following example sets the `textType` property to `input`:

```
fl.getDocumentDOM().selection[0].textType = "input";
```

## text.useDeviceFonts

**Availability**

Flash MX 2004.

**Usage**

```
text.useDeviceFonts
```

**Description**

Property; a Boolean value. A value of `true` causes Flash to draw text using device fonts.

**Example**

The following example causes Flash to use device fonts when drawing text:

```
fl.getDocumentDOM().selection[0].useDeviceFonts = true;
```

## text.variableName

### Availability

Flash MX 2004.

### Usage

```
text.variableName
```

### Description

Property; a string that contains the name of the variable associated with the Text object. This property works only with dynamic or input text; it generates a warning if used with other text types.

This property is supported only in ActionScript 1.0 and ActionScript 2.0.

### Example

The following example sets the variable name of the selected text box to `firstName`:

```
f1.getDocumentDOM().selection[0].variableName = "firstName";
```

# Chapter 44: TextAttrs object

## textAttrs summary

### Availability

Flash MX 2004.

### Description

The TextAttrs object contains all the properties of text that can be applied to a subselection. This object is a property of the TextRun object (`textRun.textAttrs`).

### Property summary

The following properties are available for the TextAttrs object:

Property	Description
<code>textAttrs.aliasText</code>	A Boolean value that specifies that Flash should draw the text using a method optimized for increasing the legibility of small text.
<code>textAttrs.alignment</code>	A string that specifies paragraph justification. Acceptable values are "left", "center", "right", and "justify".
<code>textAttrs.autoKern</code>	A Boolean value that determines whether Flash uses ( <code>true</code> ) or ignores ( <code>false</code> ) pair kerning information in the font(s) to kern the text.
<code>textAttrs.bold</code>	A Boolean value. A value of <code>true</code> causes text to appear with the bold version of the font.
<code>textAttrs.characterPosition</code>	A string that determines the baseline for the text.
<code>textAttrs.characterSpacing</code>	Deprecated in favor of <code>textAttrs.letterSpacing</code> . An integer that represents the space between characters.
<code>textAttrs.face</code>	A string that represents the name of the font, such as "Arial".
<code>textAttrs.fillColor</code>	A string, hexadecimal value, or integer that represents the fill color.
<code>textAttrs.indent</code>	An integer that specifies paragraph indentation.
<code>textAttrs.italic</code>	A Boolean value. A value of <code>true</code> causes text to appear with the italic version of the font.
<code>textAttrs.leftMargin</code>	An integer that specifies the paragraph's left margin.
<code>textAttrs.letterSpacing</code>	An integer that represents the space between characters.
<code>textAttrs.lineSpacing</code>	An integer that specifies the line spacing (leading) of the paragraph
<code>textAttrs.rightMargin</code>	An integer that specifies the paragraph's right margin.
<code>textAttrs.rotation</code>	A Boolean value. A value of <code>true</code> causes Flash to rotate the characters of the text 90°. The default value is <code>false</code> .
<code>textAttrs.size</code>	An integer that specifies the size of the font.
<code>textAttrs.target</code>	A string that represents the <code>target</code> property of the text field.
<code>textAttrs.url</code>	A string that represents the <code>URL</code> property of the text field.

## textAttrs.aliasText

### Availability

Flash MX 2004.

### Usage

```
textAttrs.aliasText
```

### Description

Property; a Boolean value that specifies that Flash should draw the text using a method optimized for increasing the legibility of small text.

### Example

The following example sets the `aliasText` property to `true` for all the text in the currently selected text field:

```
f1.getDocumentDOM().setElementTextAttr('aliasText', true);
```

## textAttrs.alignment

### Availability

Flash MX 2004.

### Usage

```
textAttrs.alignment
```

### Description

Property; a string that specifies paragraph justification. Acceptable values are "left", "center", "right", and "justify".

### Example

The following example sets the paragraphs that contain characters between index 0 up to, but not including, index 3 to justify. This can affect characters outside the specified range if they are in the same paragraph.

```
f1.getDocumentDOM().setTextSelection(0, 3);
f1.getDocumentDOM().setElementTextAttr("alignment", "justify");
```

## textAttrs.autoKern

### Availability

Flash MX 2004.

### Usage

```
textAttrs.autoKern
```

**Description**

Property; a Boolean value that determines whether Flash uses (`true`) or ignores (`false`) pair kerning information in the font(s) when it kerns the text.

**Example**

The following example selects the characters from index 2 up to, but not including, index 6 and sets the `autoKern` property to `true`:

```
f1.getDocumentDOM().setTextSelection(3, 6);
f1.getDocumentDOM().setElementTextAttr('autoKern', true);
```

## **textAttrs.bold**

**Availability**

Flash MX 2004.

**Usage**

```
textAttrs.bold
```

**Description**

Property; a Boolean value. A value of `true` causes text to appear with the bold version of the font.

**Example**

The following example selects the first character of the selected Text object and sets the `bold` property to `true`:

```
f1.getDocumentDOM().setTextSelection(0, 1);
f1.getDocumentDOM().setElementTextAttr('bold', true);
```

## **textAttrs.characterPosition**

**Availability**

Flash MX 2004.

**Usage**

```
textAttrs.characterPosition
```

**Description**

Property; a string that determines the baseline for the text. Acceptable values are `"normal"`, `"subscript"`, and `"superscript"`. This property applies only to static text.

**Example**

The following example selects the characters from index 2 up to, but not including, index 6 of the selected text field and sets the `characterPosition` property to `subscript`:

```
f1.getDocumentDOM().setTextSelection(2, 6);
f1.getDocumentDOM().setElementTextAttr("characterPosition", "subscript");
```

## textAttrs.characterSpacing

### Availability

Flash MX 2004. Deprecated in Flash 8 in favor of [textAttrs.letterSpacing](#).

### Usage

```
textAttrs.characterSpacing
```

### Description

Property; an integer that represents the space between characters. Acceptable values are -60 through 60.

This property applies only to static text; it generates a warning if used with other text types.

### Example

The following example sets the character spacing of the selected text field to 10:

```
f1.getDocumentDOM().setElementTextAttr("characterSpacing", 10);
```

## textAttrs.face

### Availability

Flash MX 2004.

### Usage

```
textAttrs.face
```

### Description

Property; a string that represents the name of the font, such as "Arial".

### Example

The following example sets the font of the selected text field from the character at index 2 up to, but not including, the character at index 8 to Arial:

```
f1.getDocumentDOM().selection[0].setTextAttr("face", "Arial", 2, 8);
```

## textAttrs.fillColor

### Availability

Flash MX 2004.

### Usage

```
textAttrs.fillColor
```

**Description**

Property; the color of the fill, in one of the following formats:

- A string in the format "#RRGGBB" or "#RRGGBAA"
- A hexadecimal number in the format 0xRRGGBB
- An integer that represents the decimal equivalent of a hexadecimal number

**Example**

The following example sets the color to red for the selected text field from the character at index 2 up to, but not including, the character at index 8:

```
f1.getDocumentDOM().selection[0].setTextAttr("fillColor", 0xff0000, 2, 8);
```

## **textAttrs.indent**

**Availability**

Flash MX 2004.

**Usage**

```
textAttrs.indent
```

**Description**

Property; an integer that specifies paragraph indentation. Acceptable values are -720 through 720.

**Example**

The following example sets the indentation of the selected text field from the character at index 2 up to, but not including, the character at index 8 to 100. This can affect characters outside the specified range if they are in the same paragraph.

```
f1.getDocumentDOM().selection[0].setTextAttr("indent", 100, 2, 8);
```

## **textAttrs.italic**

**Availability**

Flash MX 2004.

**Usage**

```
textAttrs.italic
```

**Description**

Property; a Boolean value. A value of `true` causes text to appear with the italic version of the font.

**Example**

The following example sets the selected text field to italic:

```
f1.getDocumentDOM().selection[0].setTextAttr("italic", true);
```

## textAttrs.leftMargin

### Availability

Flash MX 2004.

### Usage

```
textAttrs.leftMargin
```

### Description

Property; an integer that specifies the paragraph's left margin. Acceptable values are 0 through 720.

### Example

The following example sets the `leftMargin` property of the selected text field from the character at index 2 up to, but not including, the character at index 8 to 100. This can affect characters outside the specified range if they are in the same paragraph.

```
f1.getDocumentDOM().selection[0].setTextAttr("leftMargin", 100, 2, 8);
```

## textAttrs.letterSpacing

### Availability

Flash 8.

### Usage

```
textAttrs.letterSpacing
```

### Description

Property; an integer that represents the space between characters. Acceptable values are -60 through 60.

This property applies only to static text; it generates a warning if used with other text types.

### Example

The following code selects the characters from index 0 up to but not including index 10 and sets the character spacing to 60:

```
f1.getDocumentDOM().setTextSelection(0, 10);
f1.getDocumentDOM().setElementTextAttr("letterSpacing", 60);
```

## textAttrs.lineSpacing

### Availability

Flash MX 2004.

### Usage

```
textAttrs.lineSpacing
```

**Description**

Property; an integer that specifies the line spacing (*leading*) of the paragraph. Acceptable values are -360 through 720.

**Example**

The following example sets the selected text field's `lineSpacing` property to 100:

```
f1.getDocumentDOM().selection[0].setTextAttr("lineSpacing", 100);
```

## **textAttrs.rightMargin**

**Availability**

Flash MX 2004.

**Usage**

```
textAttrs.rightMargin
```

**Description**

Property; an integer that specifies the paragraph's right margin. Acceptable values are 0 through 720.

**Example**

The following example sets the `rightMargin` property of the selected text field from the character at index 2 up to, but not including, the character at index 8 to 100. This can affect characters outside the specified range if they are in the same paragraph.

```
f1.getDocumentDOM().selection[0].setTextAttr("rightMargin", 100, 2, 8);
```

## **textAttrs.rotation**

**Availability**

Flash MX 2004.

**Usage**

```
textAttrs.rotation
```

**Description**

Property; a Boolean value. A value of `true` causes Flash to rotate the characters of the text 90°. The default value is `false`. This property applies only to static text with a vertical orientation; it generates a warning if used with other text types.

**Example**

The following example sets the rotation of the selected text field to `true`:

```
f1.getDocumentDOM().setElementTextAttr("rotation", true);
```

## **textAttrs.size**

### **Availability**

Flash MX 2004.

### **Usage**

```
textAttrs.size
```

### **Description**

Property; an integer that specifies the size of the font.

### **Example**

The following example retrieves the size of the character at index 2 and shows the result in the Output panel:

```
f1.outputPanel.trace(f1.getDocumentDOM().selection[0].getTextAttr("size", 2));
```

## **textAttrs.target**

### **Availability**

Flash MX 2004.

### **Usage**

```
textAttrs.target
```

### **Description**

Property; a string that represents the `target` property of the text field. This property works only with static text.

### **Example**

The following example gets the `target` property of the text field in the first frame of the top layer of the current scene and shows it in the Output panel:

```
f1.outputPanel.trace(f1.getDocumentDOM().getTimeline().layers[0].frames[0].elements[0].getTextAttr("target"));
```

## **textAttrs.url**

### **Availability**

Flash MX 2004.

### **Usage**

```
textAttrs.url
```

**Description**

Property; a string that represents the URL property of the text field. This property works only with static text.

**Example**

The following example sets the URL of the selected text field to `http://www.adobe.com`:

```
f1.getDocumentDOM().setElementTextAttr("url", "http://www.adobe.com");
```

# Chapter 45: TextRun object

## textRun summary

### Availability

Flash MX 2004.

### Description

The TextRun object represents a run of characters that have attributes that match all of the properties in the [TextAttrs object](#). This object is a property of the Text object ([text.textRuns](#)).

### Property summary

In addition to the properties available for use with the Text object, the TextRun object provides the following properties:

Property	Description
<a href="#">textRun.characters</a>	A string that represents the text contained in the TextRun object.
<a href="#">textRun.textAttrs</a>	The TextAttrs object containing the attributes of the run of text.

## textRun.textAttrs

### Availability

Flash MX 2004.

### Usage

```
textRun.textAttrs
```

### Description

Property; the [TextAttrs object](#) containing the attributes of the run of text.

### Example

The following example displays the properties of the first run of characters in the selected text field in the Output panel.

```
var curTextAttrs = fl.getDocumentDOM().selection[0].textRuns[0].textAttrs;
for (var prop in curTextAttrs) {
    fl.trace(prop + " = " + curTextAttrs[prop]);
}
```

## textRun.characters

### Availability

Flash MX 2004.

**Usage**

```
textRun.characters
```

**Description**

Property; the text contained in the TextRun object.

**Example**

The following example displays the characters that make up the first run of characters in the selected text field in the Output panel:

```
fl.trace(fl.getDocumentDOM().selection[0].textRuns[0].characters);
```

# Chapter 46: Timeline object

## timeline summary

### Availability

Flash MX 2004.

### Description

The Timeline object represents the Flash timeline, which can be accessed for the current document by using `f1.getDocumentDOM().getTimeline()`. This method returns the timeline of the current scene or symbol that is being edited.

When you work with scenes, each scene's timeline has an index value, and can be accessed for the current document by `f1.getDocumentDOM().timelines[i]`. (In this example, `i` is the index of the value of the timeline.)

When you work with frames by using the methods and properties of the Timeline object, remember that the frame value is a zero-based index (not the actual frame number in the sequence of frames in the timeline). That is, the first frame has a frame index of 0.

### Method summary

The following methods are available for the Timeline object:

Method	Description
<code>timeline.addMotionGuide()</code>	Adds a motion guide layer above the current layer and attaches the current layer to the newly added guide layer.
<code>timeline.addNewLayer()</code>	Adds a new layer to the document and makes it the current layer.
<code>timeline.clearFrames()</code>	Deletes all the contents from a frame or range of frames on the current layer.
<code>timeline.clearKeyframes()</code>	Converts a keyframe to a regular frame and deletes its contents on the current layer.
<code>timeline.convertToBlankKeyframes()</code>	Converts frames to blank keyframes on the current layer.
<code>timeline.convertToKeyframes()</code>	Converts a range of frames to keyframes (or converts the selection if no frames are specified) on the current layer.
<code>timeline.copyFrames()</code>	Copies a range of frames on the current layer to the clipboard.
<code>timeline.copyLayers()</code>	Copies a range of Timeline layers to the clipboard.
<code>timeline.copyMotion()</code>	Copies motion on selected frames, either from a motion tween or from frame-by-frame animation, so it can be applied to other frames.
<code>timeline.copyMotionAsAS3()</code>	Copies motion on selected frames, either from a motion tween or from frame-by-frame animation, to the clipboard as ActionScript 3.0 code.
<code>timeline.createMotionObject()</code>	Creates a new motion object at a designated start and end frame.
<code>timeline.createMotionTween()</code>	Sets the <code>frame.TweenType</code> property to <code>motion</code> for each selected keyframe on the current layer, and converts each frame's contents to a single symbol instance if necessary.

Method	Description
<code>timeline.cutFrames()</code>	Cuts a range of frames on the current layer from the timeline and saves them to the clipboard.
<code>timeline.cutLayers()</code>	Cuts a range of Timeline layers and saves them to the clipboard.
<code>timeline.deleteLayer()</code>	Deletes a layer.
<code>timeline.duplicateLayers()</code>	Duplicates the selected layers or specified layers.
<code>timeline.expandFolder()</code>	Expands or collapses the specified folder or folders.
<code>timeline.findLayerIndex()</code>	Finds an array of indexes for the layers with the given name.
<code>timeline.getBounds()</code>	Returns the bounding rectangle for all elements on all layers on the Timeline, for a given frame.
<code>timeline.getFrameProperty()</code>	Retrieves the specified property's value for the selected frames.
<code>timeline.getGuidelines()</code>	Returns an XML string that represents the current positions of the horizontal and vertical guide lines for a timeline (View > Guides > Show Guides).
<code>timeline.getLayerProperty()</code>	Retrieves the specified property's value for the selected layers.
<code>timeline.getSelectedFrames()</code>	Retrieves the currently selected frames in an array.
<code>timeline.getSelectedLayers()</code>	Retrieves the zero-based index values of the currently selected layers.
<code>timeline.insertBlankKeyframe()</code>	Inserts a blank keyframe at the specified frame index; if the index is not specified, inserts the blank keyframe by using the playhead/selection.
<code>timeline.insertFrames()</code>	Inserts the specified number of frames at the given frame number.
<code>timeline.insertKeyframe()</code>	Inserts a keyframe at the specified frame.
<code>timeline.pasteFrames()</code>	Pastes the range of frames from the clipboard into the specified frames.
<code>timeline.pasteLayers()</code>	Pastes copied layers to the Timeline above the specified layer index.
<code>timeline.pasteMotion()</code>	Pastes the range of motion frames retrieved by <code>timeline.copyMotion()</code> to the Timeline.
<code>timeline.pasteMotionSpecial()</code>	Pastes motion on selected frames, displaying a dialog box that lets the user choose which parts of a classic tween to paste.
<code>timeline.removeFrames()</code>	Deletes the frame.
<code>timeline.removeMotionObject()</code>	Removes the motion object created with <code>timeline.createMotionObject()</code> , and converts the frame(s) to static frames.
<code>timeline.reorderLayer()</code>	Moves the first specified layer before or after the second specified layer.
<code>timeline.reverseFrames()</code>	Reverses a range of frames.
<code>timeline.selectAllFrames()</code>	Selects all the frames in the current timeline.
<code>timeline setFrameProperty()</code>	Sets the property of the Frame object for the selected frames.
<code>timeline.setGuidelines()</code>	Replaces the guide lines for the timeline with the information specified.
<code>timeline.setLayerProperty()</code>	Sets the specified property on all the selected layers to a specified value.
<code>timeline.setSelectedFrames()</code>	Selects a range of frames in the current layer or sets the selected frames to the selection array passed into this method.

Method	Description
<code>timeline.setSelectedLayers()</code>	Sets the layer to be selected; also makes the specified layer the current layer.
<code>timeline.showLayerMasking()</code>	Shows the layer masking during authoring by locking the mask and masked layers.
<code>timeline.startPlayback()</code>	Starts automatic playback of the timeline if it is not currently playing.
<code>timeline.stopPlayback()</code>	Stops automatic playback of the timeline if it is currently playing.

### Property summary

The following properties are available for the Timeline object:

Property	Description
<code>timeline.currentFrame</code>	A zero-based index for the frame at the current playhead location.
<code>timeline.currentLayer</code>	A zero-based index for the currently active layer.
<code>timeline.frameCount</code>	Read-only; an integer that represents the number of frames in this timeline's longest layer.
<code>timeline.layerCount</code>	Read-only; an integer that represents the number of layers in the specified timeline.
<code>timeline.layers</code>	Read-only; an array of layer objects.
<code>timeline.libraryItem</code>	Read-only property; indicates whether the timeline belongs to a scene.
<code>timeline.name</code>	A string that represents the name of the current timeline.

## timeline.addMotionGuide()

### Availability

Flash MX 2004.

### Usage

```
timeline.addMotionGuide()
```

### Parameters

None.

### Returns

An integer that represents the zero-based index of the newly added guide layer. If the current layer type is not of type "Normal", Flash returns -1.

### Description

Method; adds a motion guide layer above the current layer and attaches the current layer to the newly added guide layer, converting the current layer to a layer of type "Guided".

This method functions only on a layer of type "Normal". It has no effect on a layer whose type is "Folder", "Mask", "Masked", "Guide", or "Guided".

**Example**

The following example adds a motion guide layer above the current layer, and converts the current layer to Guided:

```
f1.getDocumentDOM().getTimeline().addMotionGuide();
```

## timeline.addNewLayer()

**Availability**

Flash MX 2004.

**Usage**

```
timeline.addNewLayer([name] [, layerType [, bAddAbove]])
```

**Parameters**

**name** A string that specifies the name for the new layer. If you omit this parameter, a new default layer name is assigned to the new layer ("Layer n," where *n* is the total number of layers created and deleted for that particular instance of the file). This parameter is optional.

**layerType** A string that specifies the type of layer to add. If you omit this parameter, a "Normal" type layer is created. This parameter is optional. Acceptable values are "normal", "guide", "guided", "mask", "masked", and "folder".

**bAddAbove** A Boolean value that, if set to `true` (the default), causes Flash to add the new layer above the current layer; `false` causes Flash to add the layer below the current layer. This parameter is optional.

**Returns**

An integer value of the zero-based index of the newly added layer.

**Description**

Method; adds a new layer to the document and makes it the current layer.

**Example**

The following example adds a new layer to the timeline with a default name generated by Flash:

```
f1.getDocumentDOM().getTimeline().addNewLayer();
```

The following example adds a new folder layer on top of the current layer and names it `Folder1`:

```
f1.getDocumentDOM().getTimeline().addNewLayer("Folder1", "folder", true);
```

## timeline.clearFrames()

**Availability**

Flash MX 2004.

**Usage**

```
timeline.clearFrames([startFrameIndex [, endFrameIndex]])
```

**Parameters**

**startFrameIndex** A zero-based index that defines the beginning of the range of frames to clear. If you omit *startFrameIndex*, the method uses the current selection. This parameter is optional.

**endFrameIndex** A zero-based index that defines the end of the range of frames to clear. The range goes up to, but does not include, *endFrameIndex*. If you specify only *startFrameIndex*, *endFrameIndex* defaults to the value of *startFrameIndex*. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; deletes all the contents from a frame or range of frames on the current layer.

**Example**

The following example clears the frames from Frame 6 up to, but not including, Frame 11 (remember that index values are different from frame number values):

```
f1.getDocumentDOM().getTimeline().clearFrames(5, 10);
```

The following example clears Frame 15:

```
f1.getDocumentDOM().getTimeline().clearFrames(14);
```

## timeline.clearKeyframes()

**Availability**

Flash MX 2004.

**Usage**

```
timeline.clearKeyframes([startFrameIndex [, endFrameIndex]])
```

**Parameters**

**startFrameIndex** A zero-based index that defines the beginning of the range of frames to clear. If you omit *startFrameIndex*, the method uses the current selection. This parameter is optional.

**endFrameIndex** A zero-based index that defines the end of the range of frames to clear. The range goes up to, but does not include, *endFrameIndex*. If you specify only *startFrameIndex*, *endFrameIndex* defaults to the value of *startFrameIndex*. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; converts a keyframe to a regular frame and deletes its contents on the current layer.

**Example**

The following example clears the keyframes from Frame 5 up to, but not including, Frame 10 (remember that index values are different from frame number values):

```
f1.getDocumentDOM().getTimeline().clearKeyframes(4, 9);
```

The following example clears the keyframe at Frame 15 and converts it to a regular frame:

```
f1.getDocumentDOM().getTimeline().clearKeyframes(14);
```

## timeline.convertToBlankKeyframes()

### Availability

Flash MX 2004.

### Usage

```
timeline.convertToBlankKeyframes([startFrameIndex [, endFrameIndex]])
```

### Parameters

**startFrameIndex** A zero-based index that specifies the starting frame to convert to keyframes. If you omit *startFrameIndex*, the method converts the currently selected frames. This parameter is optional.

**endFrameIndex** A zero-based index that specifies the frame at which the conversion to keyframes will stop. The range of frames to convert goes up to, but does not include, *endFrameIndex*. If you specify only *startFrameIndex*, *endFrameIndex* defaults to the value of *startFrameIndex*. This parameter is optional.

### Returns

Nothing.

### Description

Method; converts frames to blank keyframes on the current layer.

### Example

The following example converts Frame 2 up to, but not including, Frame 10 to blank keyframes (remember that index values are different from frame number values):

```
f1.getDocumentDOM().getTimeline().convertToBlankKeyframes(1, 9);
```

The following example converts Frame 5 to a blank keyframe:

```
f1.getDocumentDOM().getTimeline().convertToBlankKeyframes(4);
```

## timeline.convertToKeyframes()

### Availability

Flash MX 2004.

### Usage

```
timeline.convertToKeyframes([startFrameIndex [, endFrameIndex]])
```

**Parameters**

**startFrameIndex** A zero-based index that specifies the first frame to convert to keyframes. If you omit *startFrameIndex*, the method converts the currently selected frames. This parameter is optional.

**endFrameIndex** A zero-based index that specifies the frame at which conversion to keyframes will stop. The range of frames to convert goes up to, but does not include, *endFrameIndex*. If you specify only *startFrameIndex*, *endFrameIndex* defaults to the value of *startFrameIndex*. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; converts a range of frames to keyframes (or converts the selection if no frames are specified) on the current layer.

**Example**

The following example converts the selected frames to keyframes:

```
f1.getDocumentDOM().getTimeline().convertToKeyframes();
```

The following example converts to keyframes the frames from Frame 2 up to, but not including, Frame 10 (remember that index values are different from frame number values):

```
f1.getDocumentDOM().getTimeline().convertToKeyframes(1, 9);
```

The following example converts Frame 5 to a keyframe:

```
f1.getDocumentDOM().getTimeline().convertToKeyframes(4);
```

## timeline.copyFrames()

**Availability**

Flash MX 2004.

**Usage**

```
timeline.copyFrames([startFrameIndex [, endFrameIndex]])
```

**Parameters**

**startFrameIndex** A zero-based index that specifies the beginning of the range of frames to copy. If you omit *startFrameIndex*, the method uses the current selection. This parameter is optional.

**endFrameIndex** A zero-based index that specifies the frame at which to stop copying. The range of frames to copy goes up to, but does not include, *endFrameIndex*. If you specify only *startFrameIndex*, *endFrameIndex* defaults to the value of *startFrameIndex*. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; copies a range of frames on the current layer to the clipboard.

**Example**

The following example copies the selected frames to the clipboard:

```
f1.getDocumentDOM().getTimeline().copyFrames();
```

The following example copies Frame 2 up to, but not including, Frame 10, to the clipboard (remember that index values are different from frame number values):

```
f1.getDocumentDOM().getTimeline().copyFrames(1, 9);
```

The following example copies Frame 5 to the clipboard:

```
f1.getDocumentDOM().getTimeline().copyFrames(4);
```

## timeline.copyLayers()

**Availability**

Flash CS5 Professional.

**Usage**

```
timeline.copyLayers([startLayerIndex [, endLayerIndex]])
```

**Parameters**

**startLayerIndex** Optional. A zero-based index that specifies the beginning of the range of layers to copy. If you omit startLayerIndex, the method uses the current selection.

**endLayerIndex** Optional. A zero-based index that specifies the layer at which to stop copying. The range of layers to copy goes up to and including endLayerIndex. If you specify only startLayerIndex, then endLayerIndex defaults to the value of startLayerIndex.

**Returns**

Nothing.

**Description**

Method; Copies the layers that are currently selected in the Timeline, or the layers in the specified range. Optional arguments can be provided in order to specify a layer or range of layers to copy.

**Example**

The following example copies the layers from index 2 to index 7 in the Timeline:

```
f1.getDocumentDOM().getTimeline().copyLayers(2, 7);
```

**See also**

[timeline.cutLayers\(\)](#), [timeline.pasteLayers\(\)](#), [timeline.duplicateLayers\(\)](#)

## timeline.copyMotion()

**Availability**

Flash CS3 Professional.

**Usage**

```
timeline.copyMotion()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; copies motion on selected frames, either from a motion tween or from frame-by-frame animation. You can then use [timeline.pasteMotion\(\)](#) to apply the motion to other frames.

To copy motion as text (code) that you can paste into a script, see [timeline.copyMotionAsAS3\(\)](#).

**Example**

The following example copies the motion from the selected frame or frames:

```
f1.getDocumentDOM().getTimeline().copyMotion();
```

**See also**

[timeline.copyMotionAsAS3\(\)](#), [timeline.pasteMotion\(\)](#)

## timeline.copyMotionAsAS3()

**Availability**

Flash CS3 Professional.

**Usage**

```
timeline.copyMotionAsAS3()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; copies motion on selected frames, either from a motion tween or from frame-by-frame animation, to the clipboard as ActionScript 3.0 code. You can then paste this code into a script.

To copy motion in a format that you can apply to other frames, see [timeline.copyMotion\(\)](#).

**Example**

The following example copies the motion from the selected frame or frames to the clipboard as ActionScript 3.0 code:

```
f1.getDocumentDOM().getTimeline().copyMotionAsAS3();
```

**See also**`timeline.copyMotion()`

## timeline.createMotionObject()

**Availability**

Flash Professional CS5.

**Usage**`timeline.createMotionObject([startFrame [, endFrame]])`**Parameters**

**startFrame** Specifies the first frame at which to create motion objects. If you omit *startFrame*, the method uses the current selection; if there is no selection, all frames at the current playhead on all layers are removed. This parameter is optional.

**endFrame** Specifies the frame at which to stop creating motion objects; the range of frames goes up to, but does not include, *endFrame*. If you specify only *startFrame*, *endFrame* defaults to the *startFrame* value. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; creates a new motion object. The parameters are optional, and if specified set the timeline selection to the indicated frames prior to creating the motion object.

**Example**

The following example creates a motion objects at the current playhead position on the top layer:

```
f1.getDocumentDOM().getTimeline().currentLayer = 0;  
f1.getDocumentDOM().getTimeline().createMotionObject();
```

The following example creates a motion object starting at Frame 5, and extending up to, but not including, Frame 15 of the top layer in the current scene:

```
f1.getDocumentDOM().getTimeline().currentLayer = 0;  
f1.getDocumentDOM().getTimeline().createMotionObject(5, 15);
```

## timeline.createMotionTween()

**Availability**

Flash MX 2004.

**Usage**`timeline.createMotionTween([startFrameIndex [, endFrameIndex]])`

**Parameters**

**startFrameIndex** A zero-based index that specifies the beginning frame at which to create a motion tween. If you omit *startFrameIndex*, the method uses the current selection. This parameter is optional.

**endFrameIndex** A zero-based index that specifies the frame at which to stop the motion tween. The range of frames goes up to, but does not include, *endFrameIndex*. If you specify only *startFrameIndex*, *endFrameIndex* defaults to the *startFrameIndex* value. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; sets the `frame.tweenType` property to `motion` for each selected keyframe on the current layer, and converts each frame's contents to a single symbol instance if necessary. This property is the equivalent to the Create Motion Tween menu item in the Flash authoring tool.

**Example**

The following example converts the shape in the first frame up to, but not including, Frame 10 to a graphic symbol instance and sets the `frame.tweenType` to `motion` (remember that index values are different from frame number values):

```
f1.getDocumentDOM().getTimeline().createMotionTween(0, 9);
```

## timeline.currentFrame

**Availability**

Flash MX 2004.

**Usage**

```
timeline.currentFrame
```

**Description**

Property; the zero-based index for the frame at the current playhead location.

**Example**

The following example sets the playhead of the current timeline to Frame 10 (remember that index values are different from frame number values):

```
f1.getDocumentDOM().getTimeline().currentFrame = 9;
```

The following example stores the value of the current playhead location in the `curFrame` variable:

```
var curFrame = f1.getDocumentDOM().getTimeline().currentFrame;
```

## timeline.currentLayer

**Availability**

Flash MX 2004.

**Usage**

```
timeline.currentLayer
```

**Description**

Property; the zero-based index for the currently active layer. A value of 0 specifies the top layer, a value of 1 specifies the layer below it, and so on.

**Example**

The following example makes the top layer active:

```
f1.getDocumentDOM().getTimeline().currentLayer = 0;
```

The following example stores the index of the currently active layer in the `curLayer` variable:

```
var curLayer = f1.getDocumentDOM().getTimeline().currentLayer;
```

## timeline.cutFrames()

**Availability**

Flash MX 2004.

**Usage**

```
timeline.cutFrames([startFrameIndex [, endFrameIndex]])
```

**Parameters**

`startFrameIndex` A zero-based index that specifies the beginning of a range of frames to cut. If you omit `startFrameIndex`, the method uses the current selection. This parameter is optional.

`endFrameIndex` A zero-based index that specifies the frame at which to stop cutting. The range of frames goes up to, but does not include, `endFrameIndex`. If you specify only `startFrameIndex`, `endFrameIndex` defaults to the `startFrameIndex` value. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; cuts a range of frames on the current layer from the timeline and saves them to the clipboard.

**Example**

The following example cuts the selected frames from the timeline and saves them to the clipboard:

```
f1.getDocumentDOM().getTimeline().cutFrames();
```

The following example cuts Frame 2 up to, but not including, Frame 10 from the timeline and saves them to the clipboard (remember that index values are different from frame number values):

```
f1.getDocumentDOM().getTimeline().cutFrames(1, 9);
```

The following example cuts Frame 5 from the timeline and saves it to the clipboard:

```
f1.getDocumentDOM().getTimeline().cutFrames(4);
```

## timeline.cutLayers()

### Availability

Flash CS5.5 Professional.

### Usage

```
timeline.cutLayers([startLayerIndex [, endLayerIndex]])
```

### Parameters

**startLayerIndex** Optional. A zero-based index that specifies the beginning of the range of layers to cut. If you omit startLayerIndex, the method uses the current selection.

**endLayerIndex** Optional. A zero-based index that specifies the layer at which to stop cutting. The range of layers to cut goes up to and including endLayerIndex. If you specify only startLayerIndex, then endLayerIndex defaults to the value of startLayerIndex.

### Returns

Nothing.

### Description

Method; Cuts the layers that are currently selected in the Timeline, or the layers in the specified range. Optional arguments can be provided in order to specify a layer or range of layers to cut.

### Example

The following example cuts the layers from index 2 to index 7 in the Timeline:

```
f1.getDocumentDOM().getTimeline().cutLayers(2, 7);
```

### See also

[timeline.copyLayers\(\)](#), [timeline.pasteLayers\(\)](#), [timeline.duplicateLayers\(\)](#)

## timeline.deleteLayer()

### Availability

Flash MX 2004.

### Usage

```
timeline.deleteLayer([index])
```

### Parameters

**index** A zero-based index that specifies the layer to be deleted. If there is only one layer in the timeline, this method has no effect. This parameter is optional.

### Returns

Nothing.

### Description

Method; deletes a layer. If the layer is a folder, all layers within the folder are deleted. If you do not specify the layer index, Flash deletes the currently selected layers.

### Example

The following example deletes the second layer from the top:

```
f1.getDocumentDOM().getTimeline().deleteLayer(1);
```

The following example deletes the currently selected layers:

```
f1.getDocumentDOM().getTimeline().deleteLayer();
```

## timeline.duplicateLayers()

### Availability

Flash CS5.5 Professional.

### Usage

```
timeline.duplicateLayers([startLayerIndex [, endLayerIndex]])
```

### Parameters

**startLayerIndex** Optional. A zero-based index that specifies the beginning of the range of layers to copy. It also specifies the layer above which the layers on the clipboard are pasted. If you omit `startLayerIndex`, the method uses the current layer selection.

**endLayerIndex** Optional. A zero-based index that specifies the layer at which to stop copying. The range of layers to copy goes up to and including `endLayerIndex`. If you specify only `startLayerIndex`, then `endLayerIndex` defaults to the value of `startLayerIndex`.

### Returns

Nothing.

### Description

Method; Duplicates the layers that are currently selected in the Timeline, or the layers in the specified range. Optional arguments can be provided in order to specify a layer or range of layers to duplicate.

### Example

The following example duplicates the layer currently selected in the Timeline:

```
f1.getDocumentDOM().getTimeline().duplicateLayers();
```

The following example duplicates the layers from index 2 to index 7 above layer index 2:

```
f1.getDocumentDOM().getTimeline().duplicatedLayers(2, 7);
```

### See also

[timeline.copyLayers\(\)](#), [timeline.cutLayers\(\)](#), [timeline.pasteLayers\(\)](#)

## timeline.expandFolder()

### Availability

Flash MX 2004.

### Usage

```
timeline.expandFolder(bExpand [, bRecurseNestedParents [, index]])
```

### Parameters

**bExpand** A Boolean value that, if set to `true`, causes the method to expand the folder; `false` causes the method to collapse the folder.

**bRecurseNestedParents** A Boolean value that, if set to `true`, causes all the layers within the specified folder to be opened or closed, based on the *bExpand* parameter. This parameter is optional.

**index** A zero-based index of the folder to expand or collapse. Use `-1` to apply to all layers (you also must set *bRecurseNestedParents* to `true`). This property is equivalent to the Expand All/Collapse All menu items in the Flash authoring tool. This parameter is optional.

### Returns

Nothing.

### Description

Method; expands or collapses the specified folder or folders. If you do not specify a layer, this method operates on the current layer.

### Example

The following examples use this folder structure:

```
Folder 1 ***
--layer 7
--Folder 2 ****
----Layer 5
```

The following example expands Folder 1 only:

```
f1.getDocumentDOM().getTimeline().currentLayer = 1;
f1.getDocumentDOM().getTimeline().expandFolder(true);
```

The following example expands Folder 1 only (assuming that Folder 2 collapsed when Folder 1 last collapsed; otherwise, Folder 2 appears expanded):

```
f1.getDocumentDOM().getTimeline().expandFolder(true, false, 0);
```

The following example collapses all folders in the current timeline:

```
f1.getDocumentDOM().getTimeline().expandFolder(false, true, -1);
```

## timeline.findLayerIndex()

### Availability

Flash MX 2004.

**Usage**

```
timeline.findLayerIndex(name)
```

**Parameters**

**name** A string that specifies the name of the layer to find.

**Returns**

An array of index values for the specified layer. If the specified layer is not found, Flash returns `undefined`.

**Description**

Method; finds an array of indexes for the layers with the given name. The layer index is flat, so folders are considered part of the main index.

**Example**

The following example shows the index values of all layers named `Layer 7` in the Output panel:

```
var layerIndex = fl.getDocumentDOM().getTimeline().findLayerIndex("Layer 7");
fl.trace(layerIndex);
```

The following example illustrates how to pass the values returned from this method back to `timeline.setSelectedLayers()`:

```
var layerIndex = fl.getDocumentDOM().getTimeline().findLayerIndex("Layer 1");
fl.getDocumentDOM().getTimeline().setSelectedLayers(layerIndex[0], true);
```

## timeline.frameCount

**Availability**

Flash MX 2004.

**Usage**

```
timeline.frameCount
```

**Description**

Read-only property; an integer that represents the number of frames in this timeline's longest layer.

**Example**

The following example uses a `countNum` variable to store the number of frames in the current document's longest layer:

```
var countNum = fl.getDocumentDOM().getTimeline().frameCount;
```

## timeline.getBounds()

**Availability**

Flash Professional CC.

**Usage**

```
timeline.getBounds([frame [, includeHiddenLayers]])
```

**Parameters**

**frame** The number of the frame for which you want the bounds. Defaults to 1, which is the first frame. This parameter is optional.

**includeHiddenLayers** Indicates whether to include element bounds from hidden layers. Defaults to the SWF publish setting value for "include hidden layers". This parameter is optional.

**Returns**

The bounding rectangle for all elements on all layers on the Timeline, for the specified frame.

**Description**

Method; returns the bounding rectangle for all elements on all layers on the Timeline, for a given frame.

**Example**

The following example illustrates the use of this method:

```
var doc = fl.getDocumentDOM();
var tl = doc.getTimeline();
for (var f = 1; f <= 20; f++) {
    var rect = tl.getBounds(f, true);
    if (rect != 0) {
        var width = rect.right - rect.left;
        var height = rect.bottom - rect.top;
        fl.trace("") + rect.left + "," + rect.top + "," + width + "," + height);
    }
}
```

## timeline.getFrameProperty()

**Availability**

Flash MX 2004.

**Usage**

```
timeline.getFrameProperty(property [, startframeIndex [, endFrameIndex]])
```

**Parameters**

**property** A string that specifies the name of the property for which to get the value. See the Property summary for the [Frame object](#) for a complete list of properties.

**startFrameIndex** A zero-based index that specifies the starting frame number for which to get the value. If you omit *startFrameIndex*, the method uses the current selection. This parameter is optional.

**endFrameIndex** A zero-based index that specifies the end of the range of frames to select. The range goes up to, but does not include, *endFrameIndex*. If you specify only *startFrameIndex*, *endFrameIndex* defaults to the value of *startFrameIndex*. This parameter is optional.

**Returns**

A value for the specified property, or `undefined` if all the selected frames do not have the same property value.

**Description**

Method; retrieves the specified property's value for the selected frames.

**Example**

The following example retrieves the name of the first frame in the current document's top layer and displays the name in the Output panel:

```
f1.getDocumentDOM().getTimeline().currentLayer = 0;  
f1.getDocumentDOM().getTimeline().setSelectedFrames(0, 0, true);  
var frameName = f1.getDocumentDOM().getTimeline().getFrameProperty("name");  
fl.trace(frameName);
```

## timeline.getGuidelines()

**Availability**

Flash CS4 Professional.

**Usage**

```
timeline.getGuidelines()
```

**Parameters**

None.

**Returns**

An XML string.

**Description**

Method: returns an XML string that represents the current positions of the horizontal and vertical guide lines for a timeline (View > Guides >Show Guides). To apply these guide lines to a timeline, use [timeline.setGuidelines\(\)](#).

**Example**

Assuming that you have some guide lines on the first timeline, the following example displays them as an XML string in the Output panel:

```
var currentTimeline = fl.getDocumentDOM().timelines[0];  
fl.trace(currentTimeline.getGuidelines());
```

## timeline.getLayerProperty()

**Availability**

Flash MX 2004.

**Usage**

```
timeline.getLayerProperty(property)
```

**Parameters**

**property** A string that specifies the name of the property whose value you want to retrieve. For a list of properties, see the Property summary for the [Frame object](#).

**Returns**

The value of the specified property. Flash looks at the layer's properties to determine the type. If all the specified layers don't have the same property value, Flash returns `undefined`.

**Description**

Method; retrieves the specified property's value for the selected layers.

**Example**

The following example retrieves the name of the top layer in the current document and displays it in the Output panel:

```
f1.getDocumentDOM().getTimeline().currentLayer = 0;  
var layerName = f1.getDocumentDOM().getTimeline().getLayerProperty("name");  
fl.trace(layerName);
```

## timeline.getSelectedFrames()

**Availability**

Flash MX 2004.

**Parameters**

None.

**Returns**

An array containing  $3n$  integers, where  $n$  is the number of selected regions. The first integer in each group is the layer index, the second integer is the start frame of the beginning of the selection, and the third integer specifies the ending frame of that selection range. The ending frame is not included in the selection.

**Description**

Method; retrieves the currently selected frames in an array.

**Example**

With the top layer being the current layer, the following example displays `0,5,10,0,20,25` in the Output panel:

```
var timeline = f1.getDocumentDOM().getTimeline();  
timeline.setSelectedFrames(5,10);  
timeline.setSelectedFrames(20,25,false);  
var theSelectedFrames = timeline.getSelectedFrames();  
fl.trace(theSelectedFrames);
```

**See also**

[timeline.setSelectedFrames\(\)](#)

## timeline.getSelectedLayers()

### Availability

Flash MX 2004.

### Parameters

None.

### Returns

An array of the zero-based index values of the selected layers.

### Description

Method; gets the zero-based index values of the currently selected layers.

### Example

The following example displays 1, 0 in the Output panel:

```
f1.getDocumentDOM().getTimeline().setSelectedLayers(0);
f1.getDocumentDOM().getTimeline().setSelectedLayers(1, false);
var layerArray = f1.getDocumentDOM().getTimeline().getSelectedLayers();
fl.trace(layerArray);
```

### See also

[timeline.setSelectedLayers\(\)](#)

## timeline.insertBlankKeyframe()

### Availability

Flash MX 2004.

### Usage

```
timeline.insertBlankKeyframe([frameNumIndex])
```

### Parameters

**frameNumIndex** A zero-based index that specifies the frame at which to insert the keyframe. If you omit *frameNumIndex*, the method uses the current playhead frame number. This parameter is optional.

If the specified or selected frame is a regular frame, the keyframe is inserted at the frame. For example, if you have a span of 10 frames numbered 1-10 and you select Frame 5, this method makes Frame 5 a blank keyframe, and the length of the frame span is still 10 frames. If Frame 5 is selected and is a keyframe with a regular frame next to it, this method inserts a blank keyframe at Frame 6. If Frame 5 is a keyframe and the frame next to it is already a keyframe, no keyframe is inserted but the playhead moves to Frame 6.

### Returns

Nothing.

### Description

Method; inserts a blank keyframe at the specified frame index; if the index is not specified, the method inserts the blank keyframe by using the playhead/selection. See also [timeline.insertKeyframe\(\)](#).

### Example

The following example inserts a blank keyframe at Frame 20 (remember that index values are different from frame number values):

```
f1.getDocumentDOM().getTimeline().insertBlankKeyframe(19);
```

The following example inserts a blank keyframe at the currently selected frame (or playhead location if no frame is selected):

```
f1.getDocumentDOM().getTimeline().insertBlankKeyframe();
```

## timeline.insertFrames()

### Availability

Flash MX 2004.

### Usage

```
timeline.insertFrames([numFrames [, bAllLayers [, frameNumIndex]]])
```

### Parameters

**numFrames** An integer that specifies the number of frames to insert. If you omit this parameter, the method inserts frames at the current selection in the current layer. This parameter is optional.

**bAllLayers** A Boolean value that, if set to `true`, causes the method to insert the specified number of frames in the `numFrames` parameter into all layers; if set to `false` (the default), the method inserts frames into the current layer. This parameter is optional.

**frameNumIndex** A zero-based index that specifies the frame at which to insert a new frame. This parameter is optional.

### Returns

Nothing.

### Description

Method; inserts the specified number of frames at the specified index.

If no parameters are specified, this method works as follows:

- If one or more frames are selected, the method inserts the selected number of frames at the location of the first selected frame in the current layer. That is, if frames 6 through 10 are selected (a total of five frames), the method adds five frames at Frame 6 in the layer containing the selected frames.
- If no frames are selected, the method inserts one frame at the current frame on all layers.

If parameters are specified, the method works as follows:

- If only `numFrames` is specified, inserts the specified number of frames at the current frame on the current layer.

- If *numFrames* is specified and *bAllLayers* is `true`, inserts the specified number of frames at the current frame on all layers.
- If all three parameters are specified, inserts the specified number of frames at the specified index (*frameIndex*); the value passed for *bAllLayers* determines if the frames are added only to the current layer or to all layers.

If the specified or selected frame is a regular frame, the frame is inserted at that frame. For example, if you have a span of 10 frames numbered 1-10 and you select Frame 5 (or pass a value of 4 for *frameIndex*), this method adds a frame at Frame 5, and the length of the frame span becomes 11 frames. If Frame 5 is selected and it is a keyframe, this method inserts a frame at Frame 6 regardless of whether the frame next to it is also a keyframe.

### Example

The following example inserts a frame (or frames, depending on the selection) at the current selection in the current layer:

```
f1.getDocumentDOM().getTimeline().insertFrames();
```

The following example inserts five frames at the current frame in all layers:

```
f1.getDocumentDOM().getTimeline().insertFrames(5);
```

**Note:** If you have multiple layers with frames in them, and you select a frame in one layer when using the previous command, Flash inserts the frames in the selected layer only. If you have multiple layers with no frames selected in them, Flash inserts the frames in all layers.

The following example inserts three frames in the current layer only:

```
f1.getDocumentDOM().getTimeline().insertFrames(3, false);
```

The following example inserts four frames in all layers, starting from the first frame:

```
f1.getDocumentDOM().getTimeline().insertFrames(4, true, 0);
```

## timeline.insertKeyframe()

### Availability

Flash MX 2004.

### Usage

```
timeline.insertKeyframe([frameNumIndex])
```

### Parameters

**frameNumIndex** A zero-based index that specifies the frame index at which to insert the keyframe in the current layer. If you omit *frameNumIndex*, the method uses the frame number of the current playhead or selected frame. This parameter is optional.

### Returns

Nothing.

### Description

Method; inserts a keyframe at the specified frame. If you omit the parameter, the method inserts a keyframe using the playhead or selection location.

This method works the same as `timeline.insertBlankKeyframe()` except that the inserted keyframe contains the contents of the frame it converted (that is, it's not blank).

**Example**

The following example inserts a keyframe at the playhead or selected location:

```
f1.getDocumentDOM().getTimeline().insertKeyframe();
```

The following example inserts a keyframe at Frame 10 of the second layer (remember that index values are different from frame or layer number values):

```
f1.getDocumentDOM().getTimeline().currentLayer = 1;  
f1.getDocumentDOM().getTimeline().insertKeyframe(9);
```

## timeline.layerCount

**Availability**

Flash MX 2004.

**Usage**

```
timeline.layerCount
```

**Description**

Read-only property; an integer that represents the number of layers in the specified timeline.

**Example**

The following example uses the `NumLayer` variable to store the number of layers in the current scene:

```
var NumLayer = f1.getDocumentDOM().getTimeline().layerCount;
```

## timeline.layers

**Availability**

Flash MX 2004.

**Usage**

```
timeline.layers
```

**Description**

Read-only property; an array of layer objects.

**Example**

The following example uses the `currentLayers` variable to store the array of layer objects in the current document:

```
var currentLayers = f1.getDocumentDOM().getTimeline().layers;
```

## timeline.libraryItem

### Availability

Flash Professional CS5.

### Usage

```
timeline.libraryItem
```

### Description

Read-only property; If the timeline's `libraryItem` property is null, the timeline belongs to a scene. If it's not null, you can treat it like a `LibraryItem` object.

### Example

The following example outputs the name of the `libraryItem` if the value of `libraryItem` is not null, and the name of the scene if `libraryItem` is null:

```
var item = fl.getDocumentDOM().getTimeline().libraryItem;
if (item)
    fl.trace("libraryItem name: " + item.name);
else
    fl.trace("scene name: " + fl.getDocumentDOM().getTimeline().name);
```

## timeline.name

### Availability

Flash MX 2004.

### Usage

```
timeline.name
```

### Description

Property; a string that specifies the name of the current timeline. This name is the name of the current scene, screen (slide or form), or symbol that is being edited.

### Example

The following example retrieves the first scene name:

```
var sceneName = fl.getDocumentDOM().timelines[0].name;
```

The following example sets the first scene name to `FirstScene`:

```
fl.getDocumentDOM().timelines[0].name = "FirstScene";
```

## timeline.pasteFrames()

### Availability

Flash MX 2004.

**Usage**

```
timeline.pasteFrames( [startFrameIndex [, endFrameIndex] ] )
```

**Parameters**

**startFrameIndex** A zero-based index that specifies the beginning of a range of frames to paste. If you omit *startFrameIndex*, the method uses the current selection. This parameter is optional.

**endFrameIndex** A zero-based index that specifies the frame at which to stop pasting frames. The method pastes up to, but not including, *endFrameIndex*. If you specify only *startFrameIndex*, *endFrameIndex* defaults to the *startFrameIndex* value. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; pastes the range of frames from the clipboard into the specified frames.

**Example**

The following example pastes the frames on the clipboard to the currently selected frame or playhead location:

```
f1.getDocumentDOM().getTimeline().pasteFrames();
```

The following example pastes the frames on the clipboard at Frame 2 up to, but not including, Frame 10 (remember that index values are different from frame number values):

```
f1.getDocumentDOM().getTimeline().pasteFrames(1, 9);
```

The following example pastes the frames on the clipboard starting at Frame 5:

```
f1.getDocumentDOM().getTimeline().pasteFrames(4);
```

## timeline.pasteLayers()

**Availability**

Flash CS5.5 Professional.

**Usage**

```
timeline.pasteLayers([layerIndex])
```

**Parameters**

**layerIndex** Optional. A zero-based index that specifies the layer above which the layers on the clipboard are pasted. If you omit *layerIndex*, the method uses the current selection.

**Returns**

Integer indicating the lowest layer index of the layers that were pasted.

**Description**

Method; Paste layers that have been previously cut or copied above the currently selected layer, or above the specified layer index. If the specified layer is a folder layer, the layers are pasted into the folder. Returns the lowest layer index of the layers that were pasted. This action does not affect the system clipboard.

**Example**

The following example pastes the layers from the layer clipboard above the currently selected layer in the Timeline:

```
f1.getDocumentDOM().getTimeline().pasteLayers();
```

The following example pastes the layers from the layer clipboard above layer index 2:

```
f1.getDocumentDOM().getTimeline().pasteLayers(2);
```

**See also**

[timeline.cutLayers\(\)](#), [timeline.copyLayers\(\)](#), [timeline.duplicateLayers\(\)](#)

## timeline.pasteMotion()

**Availability**

Flash CS3 Professional.

**Usage**

```
timeline.pasteMotion()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; pastes the range of motion frames retrieved by [timeline.copyMotion\(\)](#) to the Timeline. If necessary, existing frames are displaced (moved to the right) to make room for the frames being pasted.

**Example**

The following example pastes the motion on the clipboard to the currently selected frame or playhead location, displacing that frame to the right of the pasted frames:

```
f1.getDocumentDOM().getTimeline().pasteMotion();
```

**See also**

[timeline.copyMotion\(\)](#)

## timeline.pasteMotionSpecial()

**Availability**

Flash CS3 Professional.

**Usage**

```
timeline.pasteMotionSpecial()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; Pastes motion on selected frames. Applies only to a copied classic tween, not a motion tween. Displays a dialog box whose options let the user choose which parts of a classic tween to apply when pasting: X position, Y position, Horizontal scale, Vertical scale, Rotation and skew, Color, Filters, Blend mode.

**Example**

The following example displays the dialog, then pastes the classic tween to the selected frames:

```
f1.getDocumentDOM().getTimeline().pasteMotionSpecial();
```

**See also**

[timeline.pasteMotion\(\)](#)

## timeline.removeFrames()

**Availability**

Flash MX 2004.

**Usage**

```
timeline.removeFrames([startFrameIndex [,endFrameIndex]])
```

**Parameters**

**startFrameIndex** A zero-based index that specifies the first frame at which to start removing frames. If you omit *startFrameIndex*, the method uses the current selection; if there is no selection, all frames at the current playhead on all layers are removed. This parameter is optional.

**endFrameIndex** A zero-based index that specifies the frame at which to stop removing frames; the range of frames goes up to, but does not include, *endFrameIndex*. If you specify only *startFrameIndex*, *endFrameIndex* defaults to the *startFrameIndex* value. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; deletes the frame.

**Example**

The following example deletes Frame 5 up to, but not including, Frame 10 of the top layer in the current scene (remember that index values are different from frame number values):

```
f1.getDocumentDOM().getTimeline().currentLayer = 0;  
f1.getDocumentDOM().getTimeline().removeFrames(4, 9);
```

The following example deletes Frame 8 on the top layer in the current scene:

```
f1.getDocumentDOM().getTimeline().currentLayer = 0;  
f1.getDocumentDOM().getTimeline().removeFrames(7);
```

## timeline.removeMotionObject()

### Availability

Flash Professional CS5.

### Usage

```
timeline.removeMotionObject([startFrame [,endFrame]])
```

### Parameters

**startFrame** Specifies the first frame at which to start removing motion objects. If you omit *startFrame*, the method uses the current selection; if there is no selection, all frames at the current playhead on all layers are removed. This parameter is optional.

**endFrame** Specifies the frame at which to stop removing motion objects; the range of frames goes up to, but does not include, *endFrame*. If you specify only *startFrame*, *endFrame* defaults to the *startFrame* value. This parameter is optional.

### Returns

Nothing.

### Description

Method; removes the motion object and converts the frame(s) back to static frames. The parameters are optional, and if specified set the timeline selection to the indicated frames prior to removing the motion object.

### Example

The following example deletes all motion objects and converts the frames back to static frames at the current playhead position on the top layer:

```
f1.getDocumentDOM().getTimeline().currentLayer = 0;  
f1.getDocumentDOM().getTimeline().removeMotionObject();
```

The following example deletes motion objects from Frame 5 up to, but not including, Frame 15 of the top layer in the current scene:

```
f1.getDocumentDOM().getTimeline().currentLayer = 0;  
f1.getDocumentDOM().getTimeline().removeMotionObject(5, 15);
```

### See also

[timeline.createMotionObject\(\)](#)

## timeline.reorderLayer()

### Availability

Flash MX 2004.

### Usage

```
timeline.reorderLayer(layerToMove, layerToPutItBy [, bAddBefore])
```

### Parameters

**layerToMove** A zero-based index that specifies which layer to move.

**layerToPutItBy** A zero-based index that specifies which layer you want to move the layer next to. For example, if you specify 1 for *layerToMove* and 0 for *layerToPutItBy*, the second layer is placed next to the first layer.

**bAddBefore** Specifies whether to move the layer before or after *layerToPutItBy*. If you specify `false`, the layer is moved after *layerToPutItBy*. The default value is `true`. This parameter is optional.

### Returns

Nothing.

### Description

Method; moves the first specified layer before or after the second specified layer.

### Example

The following example moves the layer at index 2 to the top (on top of the layer at index 0):

```
f1.getDocumentDOM().getTimeline().reorderLayer(2, 0);
```

The following example places the layer at index 3 after the layer at index 5:

```
f1.getDocumentDOM().getTimeline().reorderLayer(3, 5, false);
```

## timeline.reverseFrames()

### Availability

Flash MX 2004.

### Usage

```
timeline.reverseFrames([startFrameIndex [, endFrameIndex]])
```

### Parameters

**startFrameIndex** A zero-based index that specifies the first frame at which to start reversing frames. If you omit *startFrameIndex*, the method uses the current selection. This parameter is optional.

**endFrameIndex** A zero-based index that specifies the first frame at which to stop reversing frames; the range of frames goes up to, but does not include, *endFrameIndex*. If you specify only *startFrameIndex*, *endFrameIndex* defaults to the value of *startFrameIndex*. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; reverses a range of frames.

**Example**

The following example reverses the positions of the currently selected frames:

```
f1.getDocumentDOM().getTimeline().reverseFrames();
```

The following example reverses frames from Frame 10 up to, but not including, Frame 15 (remember that index values are different from frame number values):

```
f1.getDocumentDOM().getTimeline().reverseFrames(9, 14);
```

## **timeline.selectAllFrames()**

**Availability**

Flash MX 2004.

**Usage**

```
timeline.selectAllFrames()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; selects all the frames in the current timeline.

**Example**

The following example selects all the frames in the current timeline.

```
f1.getDocumentDOM().getTimeline().selectAllFrames();
```

## **timeline.setFrameProperty()**

**Availability**

Flash MX 2004.

**Usage**

```
timeline.setFrameProperty(property, value [, startFrameIndex [, endFrameIndex]])
```

**Parameters**

**property** A string that specifies the name of the property to be modified. For a complete list of properties and values, see the Property summary for the [Frame object](#).

You can't use this method to set values for read-only properties such as `frame.duration` and `frame.elements`.

**value** Specifies the value to which you want to set the property. To determine the appropriate values and type, see the Property summary for the [Frame object](#).

**startFrameIndex** A zero-based index that specifies the starting frame number to modify. If you omit `startFrameIndex`, the method uses the current selection. This parameter is optional.

**endFrameIndex** A zero-based index that specifies the first frame at which to stop. The range of frames goes up to, but does not include, `endFrameIndex`. If you specify `startFrameIndex` but omit `endFrameIndex`, `endFrameIndex` defaults to the value of `startFrameIndex`. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; sets the property of the Frame object for the selected frames.

**Example**

The following example assigns the ActionScript `stop()` command to the first frame of the top layer in the current document:

```
f1.getDocumentDOM().getTimeline().currentLayer = 0;
f1.getDocumentDOM().getTimeline().setSelectedFrames(0,0,true);
f1.getDocumentDOM().getTimeline().setFrameProperty("actionScript", "stop();");
```

The following example sets a motion tween from Frame 2 up to, but not including, Frame 5, of the current layer (remember that index values are different from frame number values):

```
var doc = f1.getDocumentDOM();
doc.getTimeline().setFrameProperty("tweenType", "motion", 1, 4);
```

## timeline.setGuidelines()

**Availability**

Flash CS4 Professional.

**Usage**

```
timeline.setGuidelines(xmlString)
```

**Parameters**

**xmlString** An XML string that contains information on the guidelines to apply.

**Returns**

A Boolean value of `true` if the guidelines are successfully applied; `false` otherwise.

**Description**

Method: replaces the guide lines for the timeline (View > Guides > Show Guides) with the information specified in *xmlString*. To retrieve an XML string that can be passed to this method, use `timeline.getGuidelines()`.

To view the newly set guide lines, you may have to hide them and then view them.

**Example**

The following example applies the guide lines from one FLA file to another FLA file:

```
var doc0 = fl.documents[0];
var guides0 = doc0.timelines[0].getGuidelines();
var doc1 = fl.documents[1];
doc1.timelines[0].setGuidelines(guides0);
```

## timeline.setLayerProperty()

**Availability**

Flash MX 2004.

**Usage**

```
timeline.setLayerProperty(property, value [, layersToChange])
```

**Parameters**

**property** A string that specifies the property to set. For a list of properties, see “[Layer object](#)” on page 355.

**value** The value to which you want to set the property. Use the same type of value you would use when setting the property in the layer object.

**layersToChange** A string that identifies which layers should be modified. Acceptable values are “selected”, “all”, and “others”. The default value is “selected” if you omit this parameter. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; sets the specified property on all the selected layers to a specified value.

**Example**

The following example makes the selected layer(s) invisible:

```
fl.getDocumentDOM().getTimeline().setLayerProperty("visible", false);
```

The following example sets the name of the selected layer(s) to `selLayer`:

```
fl.getDocumentDOM().getTimeline().setLayerProperty("name", "selLayer");
```

# timeline.setSelectedFrames()

## Availability

Flash MX 2004.

## Usage

```
timeline.setSelectedFrames(startFrameIndex, endFrameIndex [, bReplaceCurrentSelection])
timeline.setSelectedFrames(selectionList [, bReplaceCurrentSelection])
```

## Parameters

**startFrameIndex** A zero-based index that specifies the beginning frame to set.

**endFrameIndex** A zero-based index that specifies the end of the selection; *endFrameIndex* is the frame after the last frame in the range to select.

**bReplaceCurrentSelection** A Boolean value that, if it is set to `true`, causes the currently selected frames to be deselected before the specified frames are selected. The default value is `true`.

**selectionList** An array of three integers, as returned by `timeline.getSelectedFrames()`.

## Returns

Nothing.

## Description

Method; selects a range of frames in the current layer or sets the selected frames to the selection array passed into this method.

## Example

The following examples show two ways to select the top layer, Frame 1, up to but not including Frame 10, and then to add Frame 12 up to but not including Frame 15 on the same layer to the current selection (remember that index values are different from frame number values):

```
f1.getDocumentDOM().getTimeline().setSelectedFrames(0, 9);
f1.getDocumentDOM().getTimeline().setSelectedFrames(11, 14, false);
f1.getDocumentDOM().getTimeline().setSelectedFrames([0, 0, 9]);
f1.getDocumentDOM().getTimeline().setSelectedFrames([0, 11, 14], false);
```

The following example first stores the array of selected frames in the `savedSelectionList` variable and then uses the array later in the code to reselect those frames after a command or user interaction has changed the selection:

```
var savedSelectionList = f1.getDocumentDOM().getTimeline().getSelectedFrames();
// Do something that changes the selection.
f1.getDocumentDOM().getTimeline().setSelectedFrames(savedSelectionList);
```

## See also

[timeline.getSelectedFrames\(\)](#)

## timeline.setSelectedLayers()

### Availability

Flash MX 2004.

### Usage

```
timeline.setSelectedLayers(index [, bReplaceCurrentSelection])
```

### Parameters

**index** A zero-based index for the layer to select.

**bReplaceCurrentSelection** A Boolean value that, if it is set to `true`, causes the method to replace the current selection; `false` causes the method to extend the current selection. The default value is `true`. This parameter is optional.

### Returns

Nothing.

### Description

Method; sets the layer to be selected, and also makes the specified layer the current layer. Selecting a layer also means that all the frames in the layer are selected.

### Example

The following example selects the top layer:

```
f1.getDocumentDOM().getTimeline().setSelectedLayers(0);
```

The following example adds the next layer to the selection:

```
f1.getDocumentDOM().getTimeline().setSelectedLayers(1, false);
```

### See also

[timeline.getSelectedLayers\(\)](#)

## timeline.showLayerMasking()

### Availability

Flash MX 2004.

### Usage

```
timeline.showLayerMasking([layer])
```

### Parameters

**layer** A zero-based index of a mask or masked layer to show masking during authoring. This parameter is optional.

**Returns**

Nothing.

**Description**

Method; shows the layer masking during authoring by locking the mask and masked layers. This method uses the current layer if no layer is specified. If you use this method on a layer that is not of type Mask or Masked, Flash displays an error in the Output panel.

**Example**

The following example specifies that the layer masking of the first layer should show during authoring.

```
f1.getDocumentDOM().getTimeline().showLayerMasking(0);
```

## timeline.startPlayback()

**Availability**

Flash Professional CS5.

**Usage**

```
timeline.startPlayback()
```

**Returns**

Nothing.

**Description**

Method; starts automatic playback of the timeline if it is currently playing. This method can be used with SWF panels to control timeline playback in the authoring environment.

**Example**

The following example starts playback of the timeline.

```
f1.getDocumentDOM().getTimeline().startPlayback();
```

## timeline.stopPlayback()

**Availability**

Flash Professional CS5.

**Usage**

```
timeline.stopPlayback()
```

**Returns**

Nothing.

**Description**

Method; stops automatic playback of the timeline if it is currently playing. This method can be used with SWF panels to control timeline playback in the authoring environment.

**Example**

The following example stops playback of the timeline.

```
f1.getDocumentDOM().getTimeline().stopPlayback();
```

# Chapter 47: ToolObj object

## toolObj summary

### Availability

Flash MX 2004.

### Description

A ToolObj object represents an individual tool in the Tools panel. To access a ToolObj object, use properties of the **Tools object**: either the `tools.toolObjs` array or `tools.activeTool`.

### Method summary

The following methods are available for the ToolObj object.

**Note:** *The following methods are used only when creating extensible tools.*

Method	Description
<code>toolObj.enablePIControl()</code>	Enables or disables the specified control in a Property inspector. Used only when creating extensible tools.
<code>toolObj.setIcon()</code>	Identifies a PNG file to use as a tool icon in the Flash Tools panel.
<code>toolObj.setMenuString()</code>	Sets the string that appears in the pop-up menu as the name for the tool.
<code>toolObj.setOptionsFile()</code>	Associates an XML file with the tool.
<code>toolObj.setPI()</code>	Sets a particular Property inspector to be used when the tool is activated.
<code>toolObj.setToolName()</code>	Assigns a name to the tool for the configuration of the Tools panel.
<code>toolObj.setToolTip()</code>	Sets the tooltip that appears when the mouse is held over the tool icon.
<code>toolObj.showPIControl()</code>	Shows or hides a control in the Property inspector.
<code>toolObj.showTransformHandles()</code>	Called in the <code>configureTool()</code> method of an extensible tool's JavaScript file to indicate that the free transform handles should appear when the tool is active.

### Property summary

The following properties are available for the ToolObj object:

Property	Description
<code>toolObj.depth</code>	An integer that specifies the depth of the tool in the pop-up menu in the Tools panel.
<code>toolObj.iconID</code>	An integer that specifies the resource ID of the tool.
<code>toolObj.position</code>	Read-only; an integer specifying the position of the tool in the Tools panel.

## toolObj.depth

### Availability

Flash MX 2004.

### Usage

```
toolObj.depth
```

### Description

Read-only property; an integer that specifies the depth of the tool in the pop-up menu in the Tools panel. This property is used only when creating extensible tools.

### Example

The following example specifies that the tool has a depth of 1, which means one level under a tool in the Tools panel:

```
f1.tools.activeTool.depth = 1;
```

## toolObj.enablePIControl()

### Availability

Flash MX 2004.

### Usage

```
toolObj.enablePIControl(control, bEnable)
```

### Parameters

**control** A string that specifies the name of the control to enable or disable. Legal values depend on the Property inspector invoked by this tool; see [toolObj.setPI\(\)](#).

A shape Property inspector has the following controls:

stroke	fill
--------	------

A text Property inspector has the following controls:

type	font	pointsize
color	bold	italic
direction	alignLeft	alignCenter
alignRight	alignJustify	spacing
position	autoKern	small
rotation	format	lineType
selectable	html	border
deviceFonts	varEdit	options
link	maxChars	target

A movie Property inspector has the following controls:

size	publish	background
framerate	player	profile

**bEnable** A Boolean value that determines whether to enable (`true`) or disable (`false`) the control.

#### Returns

Nothing.

#### Description

Method; enables or disables the specified control in a Property inspector. Used only when creating extensible tools.

#### Example

The following command in an extensible tool's JavaScript file sets Flash to not show the stroke options in the Property inspector for that tool:

```
theTool.enablePIControl("stroke",false);
```

## toolObj.iconID

#### Availability

Flash MX 2004.

#### Usage

```
toolObj.iconID
```

#### Description

Read-only property; an integer with a value of -1. This property is used only when you create extensible tools. An `iconID` value of -1 means that Flash will not try find an icon for the tool. Instead, the script for the tool should specify the icon to display in the Tools panel; see `toolObj.setIcon()`.

#### Example

The following example assigns a value of -1 (the icon ID of the current tool) to the `toolIconID` variable:

```
var toolIconID = fl.tools.activeTool.iconID
```

## toolObj.position

#### Availability

Flash MX 2004.

#### Usage

```
toolObj.position
```

**Description**

Read-only property; an integer that specifies the position of the tool in the Tools panel. This property is used only when you create extensible tools.

**Example**

The following commands in the `mouseDown()` method of a tool's JavaScript file will show that tool's position in the Tools panel as an integer in the Output panel:

```
myToolPos = fl.tools.activeTool.position;  
fl.trace(myToolPos);
```

## toolObj.setIcon()

**Availability**

Flash MX 2004.

**Usage**

```
toolObj.setIcon(file)
```

**Parameters**

**file** A string that specifies the name of the PNG file to use as the icon. The PNG file must be placed in the same folder as the JSFL file.

**Returns**

Nothing.

**Description**

Method; identifies a PNG file to use as a tool icon in the Tools panel. This method is used only when you create extensible tools.

**Example**

The following example specifies that the image in the PolyStar.png file should be used as the icon for the tool named `PolyStar`. This code is taken from the sample `PolyStar.jsfl` file (see “[Sample PolyStar tool](#)” on page 17):

```
theTool = fl.tools.activeTool;  
theTool.setIcon("PolyStar.png");
```

## toolObj.setMenuString()

**Availability**

Flash MX 2004.

**Usage**

```
toolObj.setMenuString(menuStr)
```

**Parameters**

**menuStr** A string that specifies the name that appears in the pop-up menu as the name for the tool.

**Returns**

Nothing.

**Description**

Method; sets the string that appears in the pop-up menu as the name for the tool. This method is used only when you create extensible tools.

**Example**

The following example specifies that the tool named `theTool` should display the name “PolyStar Tool” in its pop-up menu. This code is taken from the sample PolyStar.jsfl file (see “[Sample PolyStar tool](#)” on page 17):

```
theTool = fl.tools.activeTool;
theTool.setMenuString("PolyStar Tool");
```

## toolObj.setOptionsFile()

**Availability**

Flash MX 2004.

**Usage**

```
toolObj.setOptionsFile(xmlFile)
```

**Parameters**

**xmlFile** A string that specifies the name of the XML file that has the description of the tool’s options. The XML file must be placed in the same folder as the JSFL file.

**Returns**

Nothing.

**Description**

Method; associates an XML file with the tool. The file specifies the options to appear in a modal panel that is invoked by an Options button in the Property inspector. You would usually use this method in the `configureTool()` function inside your JSFL file. See [configureTool\(\)](#).

For example, the PolyStar.xml file specifies three options associated with the Polygon tool:

```
<properties>
    <property name="Style"
        variable="style"
        list="polygon,star"
        defaultValue="0"
        type="Strings"/>

    <property name="Number of Sides"
        variable="nsides"
        min="3"
        max="32"
        defaultValue="5"
        type="Number" />

    <property name="Star point size"
        variable="pointParam"
        min="0"
        max="1"
        defaultValue=".5"
        type="Double" />

</properties>
```

**Example**

The following example specifies that the file named PolyStar.xml is associated with the currently active tool. This code is taken from the sample PolyStar.jsfl file (see “[Sample PolyStar tool](#)” on page 17):

```
theTool = fl.tools.activeTool;
theTool.setOptionsFile("PolyStar.xml");
```

## toolObj.setPI()

**Availability**

Flash MX 2004.

**Usage**

```
toolObj.setPI(pi)
```

**Parameters**

**pi** A string that specifies the Property inspector to invoke for this tool.

**Returns**

Nothing.

**Description**

Method; specifies which Property inspector should be used when the tool is activated. This method is used only when you create extensible tools. Acceptable values are "shape" (the default), "text", and "movie".

**Example**

The following example specifies that the shape Property inspector should be used when the tool is activated. This code is taken from the sample PolyStar.jsfl file (see “[Sample PolyStar tool](#)” on page 17):

```
theTool = fl.tools.activeTool;
theTool.setPI("shape");
```

## **toolObj.setToolName()**

**Availability**

Flash MX 2004.

**Usage**

```
toolObj.setToolName(name)
```

**Parameters**

**name** A string that specifies the name of the tool.

**Returns**

Nothing.

**Description**

Method; assigns a name to the tool for the configuration of the Tools panel. This method is used only when you create extensible tools. The name is used only by the XML layout file that Flash reads to construct the Tools panel. The name does not appear in the Flash user interface.

**Example**

The following example assigns the name polystar to the tool named theTool. This code is taken from the sample PolyStar.jsfl file (see “[Sample PolyStar tool](#)” on page 17):

```
theTool = fl.tools.activeTool;
theTool.setToolName("polystar");
```

## **toolObj.setToolTip()**

**Availability**

Flash MX 2004.

**Usage**

```
toolObj.setToolTip(toolTip)
```

**Parameters**

**toolTip** A string that specifies the tooltip to use for the tool.

**Returns**

Nothing.

**Description**

Method; sets the tooltip that appears when the mouse is held over the tool icon. This method is used only when you create extensible tools.

**Example**

The following example specifies that the tooltip for the tool should be PolyStar Tool. This code is taken from the sample PolyStar.jsfl file (see “[Sample PolyStar tool](#)” on page 17):

```
theTool = fl.tools.activeTool;  
theTool.setToolTip("PolyStar Tool");
```

## toolObj.showPIControl()

**Availability**

Flash MX 2004.

**Usage**

```
toolObj.showPIControl(control, bShow)
```

**Parameters**

**control** A string that specifies the name of the control to show or hide. This method is used only when you create extensible tools. Valid values depend on the Property inspector invoked by this tool (see [toolObj.setPI\(\)](#)).

A shape Property inspector has the following controls:

stroke	fill
--------	------

A text Property inspector has the following controls:

type	font	pointsize
color	bold	italic
direction	alignLeft	alignCenter
alignRight	alignJustify	spacing
position	autoKern	small
rotation	format	lineType
selectable	html	border
deviceFonts	varEdit	options
link	maxChars	target

The movie Property inspector has the following controls:

size	publish	background
framerate	player	profile

**bShow** A Boolean value that determines whether to show or hide the specified control (`true` shows the control; `false` hides the control).

**Returns**

Nothing.

**Description**

Method; shows or hides a control in the Property inspector. This method is used only when you create extensible tools.

**Example**

The following command in an extensible tool's JavaScript file will set Flash to not show the fill options in the Property inspector for that tool:

```
fl.tools.activeTool.showPIControl("fill", false);
```

## toolObj.showTransformHandles()

**Availability**

Flash MX 2004.

**Usage**

```
toolObj.showTransformHandles(bShow)
```

**Parameters**

**bShow** A Boolean value that determines whether to show or hide the free transform handles for the current tool (`true` shows the handles; `false` hides them).

**Returns**

Nothing.

**Description**

Method; called in the `configureTool()` method of an extensible tool's JavaScript file to indicate that the free transform handles should appear when the tool is active. This method is used only when you create extensible tools.

**Example**

See [configureTool\(\)](#).

# Chapter 48: Tools object

## tools summary

### Availability

Flash MX 2004.

### Description

The Tools object is accessible from the flash object (`f1.tools`). The `tools.toolObjs` property contains an array of ToolObj objects, and the `tools.activeTool` property returns the ToolObj object for the currently active tool. (See also [ToolObj object](#) and the list of Extensible tools in “[Top-Level Functions and Methods](#)” on page 18.)

**Note:** *The following methods and properties are used only when creating extensible tools.*

### Method summary

The following methods are available for the Tools object:

Method	Description
<code>tools.constrainPoint()</code>	Takes two points and returns a new adjusted or <i>constrained</i> point.
<code>tools.getKeyDown()</code>	Returns the most recently pressed key.
<code>tools.setCursor()</code>	Sets the pointer to a specified appearance.
<code>tools.snapPoint()</code>	Takes a point as input and returns a new point that may be adjusted or <i>snapped</i> to the nearest geometric object.

### Property summary

The following properties are available for the Tools object:

Property	Description
<code>tools.activeTool</code>	Read-only; returns the <a href="#">ToolObj object</a> for the currently active tool.
<code>tools.altIsDown</code>	Read-only; a Boolean value that identifies if the Alt key is being pressed.
<code>tools.ctlIsDown</code>	Read-only; a Boolean value that identifies if the Control key is being pressed.
<code>tools.mouseIsDown</code>	Read-only; a Boolean value that identifies if the left mouse button is currently pressed.
<code>tools.penDownLoc</code>	Read-only; a point that represents the position of the last mouse-down event on the Stage.
<code>tools.penLoc</code>	Read-only; a point that represents the current location of the mouse.
<code>tools.shiftIsDown</code>	Read-only; a Boolean value that identifies if the Shift key is being pressed.
<code>tools.toolObjs</code>	Read-only; an array of ToolObj objects.

## tools.activeTool

### Availability

Flash MX 2004.

### Usage

```
tools.activeTool
```

### Description

Read-only property; returns the [ToolObj object](#) for the currently active tool.

### Example

The following example saves an object that represents the currently active tool in the `theTool` variable:

```
var theTool = fl.tools.activeTool;
```

## tools.altIsDown

### Availability

Flash MX 2004.

### Usage

```
tools.altIsDown
```

### Description

Read-only property; a Boolean value that identifies if the Alt key is being pressed. The value is `true` if the Alt key is pressed, and `false` otherwise.

### Example

The following example determines whether the Alt key is being pressed:

```
var isAltDown = fl.tools.altIsDown;
```

## tools.constrainPoint()

### Availability

Flash MX 2004.

### Usage

```
tools.constrainPoint(pt1, pt2)
```

### Parameters

`pt1, pt2` Points that specify the starting-click point and the drag-to point.

**Returns**

A new adjusted or constrained point.

**Description**

Method; takes two points and returns a new adjusted or constrained point. If the Shift key is pressed when the command is run, the returned point is constrained to follow either a 45° constrain (useful for something such as a line with an arrowhead) or to constrain an object to maintain its aspect ratio (such as pulling out a perfect square with the Rectangle tool).

**Example**

The following example returns a constrained point:

```
pt2 = fl.tools.constrainPoint(pt1, tempPt);
```

## tools.ctrlIsDown

**Availability**

Flash MX 2004.

**Usage**

```
tools.ctrlIsDown
```

**Description**

Read-only property; a Boolean value that is `true` if the Control key is pressed; `false` otherwise.

**Example**

The following example determines whether the Control key is being pressed:

```
var isCtrldown = fl.tools.ctrlIsDown;
```

## tools.getKeyDown()

**Availability**

Flash MX 2004.

**Usage**

```
tools.getKeyDown()
```

**Parameters**

None.

**Returns**

The integer value of the key.

**Description**

Method; returns the most recently pressed key.

**Example**

The following example displays the integer value of the most recently pressed key:

```
var theKey = fl.tools.getKeyDown();
fl.trace(theKey);
```

## tools.mouseIsDown

**Availability**

Flash MX 2004.

**Usage**

```
tools.mouseIsDown
```

**Description**

Read-only property; a Boolean value that is `true` if the left mouse button is currently down; `false` otherwise.

**Example**

The following example determines whether the left mouse button is pressed.

```
var isMouseDown = fl.tools.mouseIsDown;
```

## tools.penDownLoc

**Availability**

Flash MX 2004.

**Usage**

```
tools.penDownLoc
```

**Description**

Read-only property; a point that represents the position of the last mouse-down event on the Stage. The `tools.penDownLoc` property comprises two properties, `x` and `y`, corresponding to the `x,y` location of the mouse pointer.

**Example**

The following example determines the position of the last mouse-down event on the Stage and displays the `x` and `y` values in the Output panel:

```
var pt1 = fl.tools.penDownLoc;
fl.trace("x,y location of last mouseDown event was " + pt1.x + ", " + pt1.y)
```

**See also**[tools.penLoc](#)

## tools.penLoc

**Availability**

Flash MX 2004.

**Usage**[tools.penLoc](#)**Description**

Read-only property; a point that represents the current location of the mouse pointer. The `tools.penLoc` property comprises two properties, *x* and *y*, corresponding to the *x,y* location of the mouse pointer.

**Example**

The following example determines the current location of the mouse:

```
var tempPt = fl.tools.penLoc;
```

**See also**[tools.penDownLoc](#)

## tools.setCursor()

**Availability**

Flash MX 2004.

**Usage**

```
tools.setCursor(cursor)
```

**Parameters**

**cursor** An integer that defines the pointer appearance, as described in the following list:

- 0 = Plus cursor (+)
- 1 = black arrow
- 2 = white arrow
- 3 = four-way arrow
- 4 = two-way horizontal arrow
- 5 = two-way vertical arrow
- 6 = X
- 7 = hand cursor

**Returns**

Nothing.

**Description**

Method; sets the pointer to a specified appearance.

**Example**

The following example sets the pointer to a black arrow.

```
fl.tools.setCursor(1);
```

## tools.shiftIsDown

**Availability**

Flash MX 2004.

**Usage**

```
tools.shiftIsDown
```

**Description**

Read-only property; a Boolean value that is `true` if the Shift key is pressed; `false` otherwise.

**Example**

The following example determines whether the Shift key is being pressed.

```
var isShiftDown = fl.tools.shiftIsDown;
```

## tools.snapPoint()

**Availability**

Flash MX 2004.

**Usage**

```
tools.snapPoint(pt)
```

**Parameters**

**pt** Specifies the location of the point for which you want to return a snap point.

**Returns**

A new point that may be adjusted or snapped to the nearest geometric object.

**Description**

Method; takes a point as input and returns a new point that may be adjusted or *snapped* to the nearest geometric object. If snapping is disabled in the View menu in the Flash user interface, the point returned is the original point.

**Example**

The following example returns a new point that may be snapped to the nearest geometric object.

```
var theSnapPoint = fl.tools.snapPoint(pt1);
```

## tools.toolObjs

**Availability**

Flash MX 2004.

**Usage**

```
tools.toolObjs
```

**Description**

Read-only property; an array of ToolObj objects (see [ToolObj object](#)).

# Chapter 49: Tween Object

## Tween object Summary

### Availability

Flash Pro CC.

### Description

The Tween object can be used to access interpolated properties of tweens. Accessing properties for a non-tween frame throws errors.

### Method Summary

You can use the following methods with the Tween object class:

Method	Description
<a href="#">"Tween.getCol orTransform()" on page 571</a>	Returns color transformation data between frames.
<a href="#">"Tween.getFilte rs()" on page 572</a>	Returns filters data of a selected frame from a tween span.
<a href="#">"Tween. getGeometricT ransform()" on page 572</a>	Returns Matrix object that represents geometric transformation of a tween within a user-defined range (from offset to a selected frame).
<a href="#">"Tween. getShape()" on page 573</a>	Returns interpolated shape of a selected frame within a tween-span.

### Properties Summary

You can use the following properties within methods of Tween object class:

Property	Description
<a href="#">"Tween. duration" on page 573</a>	Duration of a tween span that is equal to number of frames in a tween.
<a href="#">"Tween. startFrame" on page 574</a>	Start frame of a tween.
<a href="#">"Tween. tweenType" on page 574</a>	Specifies the type of tween. For example, Motion or Shape.

**Usage**

```
var mat;
var frame = fl.getDocumentDOM().getTimeline().layers[0].frames[0];
var tweenObj = frame.tweenObj;
var frame1 = fl.getDocumentDOM().getTimeline().layers[1].frames[0];
fl.trace(" Tween duration = " + tweenObj.duration);
for(var i = 1; i < tweenObj.duration; i++) {
mat = tweenObj.getGeometricTransform(i);
var colors = tweenObj.getColorTransform(i);
fl.trace(" Frame " + i + " Matrix = a = " + mat.a + " b = " + mat.b + " c = " + mat.c + " d =
" + mat.d + " tx = " + mat.tx + " ty = " + mat.ty );
fl.trace(" color transform :");
fl.trace(" alpha : amount = "+colors.colorAlphaAmount+" percent = "+colors.colorAlphaPercent);
fl.trace(" red : amount = "+colors.colorRedAmount+" percent = "+colors.colorRedPercent);
fl.trace(" green : amount = "+colors.colorGreenAmount+" percent = "+colors.colorGreenPercent);
fl.trace(" blue : amount = "+colors.colorBlueAmount+" percent = "+colors.colorBluePercent); }
```

## Tween.getColorTransform( )

**Availability**

Flash Pro CC.

**Usage**`Tween.getColorTransform(frameIndex)`**Parameters****frameIndex** Offset index of interpolated frame.**Returns**

Value object {"colorAlphaAmount", "colorAlphaPercent", "colorRedAmount", "colorRedPercent", "colorGreenAmount", "colorGreenPercent", "colorBlueAmount", "colorBluePercent"}.

**Description**

Method; Gets color transformation data between frames.

**Usage**

```
var mat;
var frame = fl.getDocumentDOM().getTimeline().layers[0].frames[0];
var tweenObj = frame.tweenObj;
var frame1 = fl.getDocumentDOM().getTimeline().layers[1].frames[0];
fl.trace(" Tween duration = " + tweenObj.duration);
for(var i = 1; i < tweenObj.duration; i++) {
mat = tweenObj.getGeometricTransform(i);
var colors = tweenObj.getColorTransform(i);
fl.trace(" Frame " + i + " Matrix = a = " + mat.a + " b = " + mat.b + " c = " + mat.c + " d =
" + mat.d + " tx = " + mat.tx + " ty = " + mat.ty );
fl.trace(" color transform :");
```

## Tween.getFilters( )

### Availability

Flash Pro CC.

### Usage

```
Tween.getFilters(frameIndex);
```

### Parameters

**FrameIndex** Index of the frame from which filter data is to be retrieved.

### Returns

Returns array of Filter objects.

### Description

Method; Returns filters data of a selected frame from a tween span.

### Usage

```
var tweenObj = fl.getDocumentDOM().getTimeline().layers[0].frames[0].tweenObj;
for( var i = 0; i < tweenObj.duration; i++ ) {
    var filterList = tweenObj.getFilters(i);
    for( var j = 0; j< filterList.length; j++) {
        var filter = filterList[j];
        fl.trace(filter.name);
        fl.trace("Blur x = " + filter.blurX + " y = " + filter.blurY); } }
```

## Tween.getGeometricTransform()

### Availability

Flash Pro CC.

### Usage

```
Tween.getGeometricTransform(frameIndex)
```

### Parameters

**FrameIndex** Offset index of the frame from which geometric transformations have to be retrieved.

### Returns

Matrix object that represents geometric transformations at the frame offset.

### Description

Method; Returns Matrix object that represents geometric transformation of a tween within a user-defined range (from offset to a selected frame).

**Usage**

```
var mat;
var frame = fl.getDocumentDOM().getTimeline().layers[0].frames[0];
var tweenObj = frame.tweenObj;
var frame1 = fl.getDocumentDOM().getTimeline().layers[1].frames[0];
fl.trace(" Tween duration = " + tweenObj.duration);
for(var i = 1; i < tweenObj.duration; i++) {
mat = tweenObj.getGeometricTransform(i);
var colors = tweenObj.getColorTransform(i);
fl.trace(" Frame " + i + " Matrix = a = " + mat.a + " b = " + mat.b + " c = " + mat.c + " d =
" + mat.d + " tx = " + mat.tx + " ty = " + mat.ty );
fl.trace(" color transform :");
fl.trace(" alpha : amount = "+colors.colorAlphaAmount+" percent = "+colors.colorAlphaPercent);
fl.trace(" red : amount = "+colors.colorRedAmount+" percent = "+colors.colorRedPercent);
fl.trace(" green : amount = "+colors.colorGreenAmount+" percent = "+colors.colorGreenPercent);
fl.trace(" blue : amount = "+colors.colorBlueAmount+" percent = "+colors.colorBluePercent); }
```

## Tween.getShape( )

**Availability**

Flash Pro CC.

**Usage**`Tween.getShape( )`**Parameters****FrameIndex** Offset index of the frame from which shape data has to be retrieved.**Returns**

Returns shape coordinates at the frame offset.

**Description**

Method; Returns interpolated shape of a selected frame within a tween-span.

**Usage**

```
var tweenObj = fl.getDocumentDOM().getTimeline().layers[0].frames[0].tweenObj;if( tweenObj.tweenType ==
"shape") {for(var i = 1; i < tweenObj.duration; i++) {var shape = tweenObj.getShape(i); //// code to perform some
operation on returned shape } }
```

## Tween.duration

**Availability**

Flash Pro CC

**Usage**

```
Tween.duration
```

**Description**

Duration of a tween span that is equal to number of frames in a tween.

**Example**

```
var tweenObj = fl.getDocumentDOM().getTimeline().layers[0].frames[0].tweenObj;  
if( tweenObj.tweenType == "shape") {  
for(var i = 1; i < tweenObj.duration; i++) {  
var shape = tweenObj.getShape(i); ////// code to perform some operation on returned shape } }
```

## Tween.startFrame

**Availability**

Flash Pro CC

**Usage**

```
Tween.startFrame
```

**Description**

Start frame of a tween.

**Example**

```
var tweenObj = fl.getDocumentDOM().getTimeline().layers[0].frames[0].tweenObj;  
fl.trace(tweenObj.startFrame);
```

## Tween.tweenType

**Availability**

Flash Pro CC

**Usage**

```
Tween.tweenType
```

**Description**

Specifies the type of tween. For example, Motion or Shape.

**Example**

```
var tweenObj = fl.getDocumentDOM().getTimeline().layers[0].frames[0].tweenObj;  
if( tweenObj.tweenType == "shape") {  
for(var i = 1; i < tweenObj.duration; i++) {  
var shape = tweenObj.getShape(i); ////// code to perform some operation on returned shape } }
```

# Chapter 50: Vertex object

## vertex summary

### Availability

Flash MX 2004.

### Description

The Vertex object is the part of the shape data structure that holds the coordinate data.

### Method summary

You can use the following methods with the Vertex object:

Method	Description
<code>vertex.getHalfEdge()</code>	Gets a <a href="#">HalfEdge object</a> that shares this vertex.
<code>vertex.setLocation()</code>	Sets the location of the vertex.

### Property summary

The following properties are available for the Vertex object:

Property	Description
<code>vertex.x</code>	Read-only; the x location of the vertex in pixels.
<code>vertex.y</code>	Read-only; the y location of the vertex in pixels.

## vertex.getHalfEdge()

### Availability

Flash MX 2004.

### Usage

```
vertex.getHalfEdge()
```

### Parameters

None.

### Returns

A [HalfEdge object](#).

### Description

Method; gets a [HalfEdge object](#) that shares this vertex.

**Example**

The following example shows how to get other half edges that share the same vertex:

```
var shape = fl.getDocumentDOM().selection[0];
var hEdge = shape.edges[0].getHalfEdge(0);
var theVertex = hEdge.getVertex();
var someHEdge = theVertex.getHalfEdge(); // Not necessarily the same half edge
var theSameVertex = someHEdge.getVertex();
fl.trace('the same vertex: ' + theSameVertex);
```

## **vertex.setLocation()**

**Availability**

Flash MX 2004.

**Usage**

```
vertex.setLocation(x, y)
```

**Parameters**

- x** A floating-point value that specifies the *x* coordinate of where the vertex should be positioned, in pixels.
- y** A floating-point value that specifies the *y* coordinate of where the vertex should be positioned, in pixels.

**Returns**

Nothing.

**Description**

Method; sets the location of the vertex. You must call [shape.beginEdit\(\)](#) before using this method.

**Example**

The following example sets the vertex to the origin point:

```
var shape = fl.getDocumentDOM().selection[0];
shape.beginEdit();
var hEdge = shape.edges[0].getHalfEdge(0);
var vertex = hEdge.getVertex();
var someHEdge = vertex.getHalfEdge();
var vertex = someHEdge.getVertex();
// Move the vertex to the origin.
vertex.setLocation(0.0, 0.0);
shape.endEdit();
```

## **vertex.x**

**Availability**

Flash MX 2004.

**Usage**`vertex.x`**Description**

Read-only property; the *x* location of the vertex, in pixels.

**Example**

The following example displays the location of the *x* and *y* values of the vertex in the Output panel:

```
var shape = fl.getDocumentDOM().selection[0];
var hEdge = shape.edges[0].getHalfEdge(0);
var vertex = hEdge.getVertex();

fl.trace('x location of vertex is: ' + vertex.x);
fl.trace('y location of vertex is: ' + vertex.y);
```

## **vertex.y**

**Availability**

Flash MX 2004.

**Usage**`vertex.y`**Description**

Read-only property; the *y* location of the vertex, in pixels.

**Example**

See [vertex.x](#).

# Chapter 51: VideoItem object

## videoltem summary

**Inheritance** [Item object](#) > VideoItem object

### Availability

Flash MX 2004.

### Description

The VideoItem object is a subclass of the [Item object](#).

### Method summary

In addition to the Item object methods, the VideoItem object has the following method:

Property	Description
<a href="#">videoItem.exportToFLV()</a>	Exports the specified item to an FLV file.

### Property summary

In addition to the Item object properties, you can use the following properties with the VideoItem object:

Property	Description
<a href="#">videoItem.fileLastModifiedDate</a>	Read-only; a string containing a hexadecimal number that represents the number of seconds that have elapsed between January 1, 1970, and the modification date of the original file (on disk) at the time the file was imported to the library.
<a href="#">videoItem.lastModifiedDate</a>	Read-only; the modification date of the video item in the Library.
<a href="#">videoItem.sourceFileExists</a>	Read-only; a Boolean value that specifies whether the file that was imported to the Library still exists in the location from where it was imported.
<a href="#">videoItem.sourceFileIsCurrent</a>	Read-only; a Boolean value that specifies whether the file modification date of the Library item is the same as the modification date (on disk) of the file that was imported.
<a href="#">videoItem.sourceFilePath</a>	Read-only; a string that specifies the path to the video item.
<a href="#">videoItem.videoType</a>	Read-only; a string that specifies the type of video the item represents.

## videoltem.exportToFLV()

### Availability

Flash CS4 Professional.

### Usage

```
videoItem.exportToFLV(fileURI)
```

**Parameters**

**fileURI** A string, expressed as a file:/// URI, that specifies the path and name of the exported file.

**Returns**

A Boolean value of `true` if the file is exported successfully; `false` otherwise.

**Description**

Method; exports the specified item to an FLV file.

**Example**

Assuming that the first item in the Library is a video item, the following code exports it as an FLV file:

```
var videoFileURL = "file:///C|/out.flv";
var libItem = fl.getDocumentDOM().library.items[0];
libItem.exportToFLV(videoFileURL);
```

## **videoItem.fileLastModifiedDate**

**Availability**

Flash CS4 Professional.

**Usage**

```
videoItem.fileLastModifiedDate
```

**Description**

Read-only property: a string containing a hexadecimal number that represents the number of seconds that have elapsed between January 1, 1970, and the modification date of the original file (on disk) at the time the file was imported to the library. If the file no longer exists, this value is "00000000".

**Example**

Assuming that the first item in the Library is a video item, the following code displays a hexadecimal number as described above.

```
var libItem = fl.getDocumentDOM().library.items[0];
fl.trace("Mod date when imported = " + libItem.fileLastModifiedDate);
```

**See also**

[videoItem.sourceFileExists](#), [videoItem.sourceFileIsCurrent](#), [videoItem.sourceFilePath](#),  
[FLfile.getModificationDate\(\)](#)

## **videoItem.lastModifiedDate**

**Availability**

Flash Pro CS6.

**Usage**

```
videoItem.lastModifiedDate
```

**Description**

Read-only property; a hexadecimal value indicating the modification date and time of the video item. This value is incremented every time the video item is imported.

**Example**

Assuming the first item in the Library is a video item, the following code displays a hex number as described above.

```
var libItem = fl.getDocumentDOM().library.items[0];
fl.trace("Mod date when imported = " + libItem.lastModifiedDate);
```

## **videoItem.sourceFileExists**

**Availability**

Flash CS4 Professional.

**Usage**

```
videoItem.sourceFileExists
```

**Description**

Read-only property: a Boolean value of `true` if the file that was imported to the Library still exists in the location from where it was imported; `false` otherwise.

**Example**

Assuming that the first item in the Library is a video item, the following code displays "true" if the file that was imported into the Library still exists.

```
var libItem = fl.getDocumentDOM().library.items[0];
fl.trace("sourceFileExists = "+ libItem.sourceFileExists);
```

**See also**

[videoItem.sourceFileIsCurrent](#), [videoItem.sourceFilePath](#)

## **videoItem.sourceFileIsCurrent**

**Availability**

Flash CS4 Professional.

**Usage**

```
videoItem.sourceFileIsCurrent
```

**Description**

Read-only property: a Boolean value of `true` if the file modification date of the Library item is the same as the modification date (on disk) of the file that was imported; `false` otherwise.

**Example**

Assuming that the first item in the Library is a video item, the following code displays "true" if the file that was imported has not been modified on disk since it was imported.

```
var libItem = fl.getDocumentDOM().library.items[0];
fl.trace("fileIsCurrent = "+ libItem.sourceFileIsCurrent);
```

**See also**

[videoItem.fileLastModifiedDate](#), [videoItem.sourceFilePath](#)

## videolitem.sourceFilePath

**Availability**

Flash 8.

**Usage**

```
videoItem.sourceFilePath
```

**Description**

Read-only property; a string, expressed as a file:/// URI that specifies the path to the video item.

**Example**

The following example displays the name and source file path of any items in the library that are of type video:

```
for (idx in fl.getDocumentDOM().library.items) {
  if (fl.getDocumentDOM().library.items[idx].itemType == "video") {
    var myItem = fl.getDocumentDOM().library.items[idx];
    fl.trace(myItem.name + " source is " + myItem.sourceFilePath);
  }
}
```

**See also**

[videoItem.sourceFileExists](#)

## videolitem.videoType

**Availability**

Flash 8.

**Usage**

```
videoItem.videoType
```

**Description**

Read-only property; a string that specifies the type of video the item represents. Possible values are "embedded video" and "video".

**Example**

The following example displays the name and type of any items in the library that are of type `video`:

```
for (idx in fl.getDocumentDOM().library.items) {  
    if (fl.getDocumentDOM().library.items[idx].itemType == "video") {  
        var myItem = fl.getDocumentDOM().library.items[idx];  
        fl.trace(myItem.name + " is " + myItem.videoType);  
    }  
}
```

# Chapter 52: XMLUI object

## xmlui summary

### Availability

Flash MX 2004.

### Description

Flash 8 supports custom dialog boxes written in a subset of the XML User Interface Language (XUL). An XML User Interface (XMLUI) dialog box can be used by several Flash features, such as commands and behaviors, to provide a user interface for features that you build using extensibility. The XMLUI object provides the ability to get and set properties of an XMLUI dialog box, and accept or cancel out of one. The XMLUI methods can be used in callbacks, such as oncommand handlers in buttons.

You can write a dialog.xml file and invoke it from the JavaScript API using the `document.xmlPanel()` method. To retrieve an object representing the current XMLUI dialog box, use `f1.xmlui`.

### Method summary

The following methods are available for the XMLUI object:

Method	Description
<code>xmlui.accept()</code>	Closes the current XMLUI dialog box with an accept state.
<code>xmlui.cancel()</code>	Closes the current XMLUI dialog box with a cancel state.
<code>xmlui.get()</code>	Retrieves the value of the specified property of the current XMLUI dialog box.
<code>xmlui.getControlItemElement()</code>	Returns the current control item for the specified control.
<code>xmlui.getEnabled()</code>	Returns a Boolean value that specifies whether the control is enabled or disabled (dimmed).
<code>xmlui.getVisible()</code>	Returns a Boolean value that specifies whether the control is visible or hidden.
<code>xmlui.set()</code>	Modifies the value of the specified property of the current XMLUI dialog box.
<code>xmlui.setControlItemElement()</code>	Sets the label and value for the current item.
<code>xmlui.setControlItemElements()</code>	Sets the label, value pairs of the current item.
<code>xmlui.setEnabled()</code>	Enables or disables (dims) a control.
<code>xmlui.setVisible()</code>	Shows or hides a control.

## xmlui.accept()

### Availability

Flash MX 2004.

**Usage**

```
xmlui.accept()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; closes the current XMLUI dialog box with an accept state, which is equivalent to the user clicking the OK button.

**See also**

[fl.xmlui](#), [document.xmlPanel\(\)](#), [xmlui.cancel\(\)](#)

## xmlui.cancel()

**Availability**

Flash MX 2004.

**Usage**

```
xmlui.cancel()
```

**Parameters**

None.

**Returns**

Nothing.

**Description**

Method; closes the current XMLUI dialog box with a cancel state, which is equivalent to the user clicking the Cancel button.

**See also**

[fl.xmlui](#), [document.xmlPanel\(\)](#), [xmlui.accept\(\)](#)

## xmlui.get()

**Availability**

Flash MX 2004.

**Usage**

```
xmlui.get(controlPropertyName)
```

**Parameters**

**controlPropertyName** A string that specifies the name of the XMLUI property whose value you want to retrieve.

**Returns**

A string that represents the value of the specified property. In cases where you might expect a Boolean value of `true` or `false`, it returns the string "true" or "false".

**Description**

Method; retrieves the value of the specified property of the current XMLUI dialog box.

**Example**

The following example returns the value of a property named `URL`:

```
fl.xmlui.get("URL");
```

**See also**

[fl.xmlui](#), [document.xmlPanel\(\)](#), [xmlui.getControlItemElement\(\)](#), [xmlui.set\(\)](#)

## xmlui.getControlItemElement()

**Availability**

Flash 8.

**Usage**

```
xmlui.getControlItemElement(controlPropertyName)
```

**Parameters**

**controlPropertyName** A string that specifies the property whose control item element you want to retrieve.

**Returns**

An object that represents the current control item for the control specified by `controlPropertyName`.

**Description**

Method; returns the label and value of the line selected in a ListBox or ComboBox control for the control specified by `controlPropertyName`.

**Example**

The following example returns the label and value of the currently selected line for the `myListBox` control:

```
var elem = new Object();
elem = fl.xmlui.getControlItemElement("myListBox");
fl.trace("label = " + elem.label + " value = " + elem.value);
```

**See also**

[fl.xmlui](#), [document.xmlPanel\(\)](#), [xmlui.get\(\)](#), [xmlui.setControlItemElement\(\)](#),  
[xmlui.setControlItemElements\(\)](#)

## xmlui.getEnabled()

### Availability

Flash 8.

### Usage

```
xmlui.getEnabled(controlID)
```

### Parameters

**controlID** A string that specifies the ID attribute of the control whose status you want to retrieve.

### Returns

A Boolean value of `true` if the control is enabled; `false` otherwise.

### Description

Method; returns a Boolean value that specifies whether the control is enabled or disabled (dimmed).

### Example

The following example returns a value that indicates whether the control with the ID attribute `myListBox` is enabled:

```
var isEnabled = fl.xmlui.getEnabled("myListBox");
fl.trace(isEnabled);
```

### See also

[fl.xmlui](#), [document.xmlPanel\(\)](#), [xmlui.setEnabled\(\)](#)

## xmlui.getVisible()

### Availability

Flash 8.

### Usage

```
xmlui.getVisible(controlID)
```

### Parameters

**controlID** A string that specifies the ID attribute of the control whose visibility status you want to retrieve.

### Returns

A Boolean value of `true` if the control is visible, or `false` if it is invisible (hidden).

### Description

Method; returns a Boolean value that specifies whether the control is visible or hidden.

### Example

The following example returns a value that indicates whether the control with the ID attribute `myListBox` is visible:

```
var isVisible = fl.xmlui.getVisible("myListBox");
fl.trace(isVisible);
```

**See also**[xmlui.setVisible\(\)](#)

## xmlui.set()

**Availability**

Flash MX 2004.

**Usage**

```
xmlui.set(controlPropertyName, value)
```

**Parameters****controlPropertyName** A string that specifies the name of XMLUI property to modify.**value** A string that specifies the value to which you want to set the XMLUI property.**Returns**

Nothing.

**Description**

Method; modifies the value of the specified property of the current XMLUI dialog box.

**Example**

The following example sets the value of a property named URL to www.adobe.com:

```
fl.xmlui.set("URL", "www.adobe.com");
```

**See also**[fl.xmlui](#), [document.xmlPanel\(\)](#), [xmlui.get\(\)](#), [xmlui.setControlItemElement\(\)](#),  
[xmlui.setControlItemElements\(\)](#)

## xmlui.setControlItemElement()

**Availability**

Flash 8.

**Usage**

```
xmlui.setControlItemElement(controlPropertyName, elementItem)
```

**Parameters****controlPropertyName** A string that specifies the control item element to set.

**elementItem** A JavaScript object with a string property named `label` and an optional string property named `value`. If the `value` property does not exist, then it is created and assigned the same value as `label`.

### Returns

Nothing.

### Description

Method; sets the label and value of the currently selected line in the ListBox or ComboBox control specified by `controlPropertyName`.

### Example

The following example sets the label and value for the current item of the control property named `PhoneNumber`:

```
var elem = new Object();
elem.label = "Fax";
elem.value = "707-555-5555";
fl.xmlui.setControlItemElement("PhoneNumber", elem);
```

### See also

[fl.xmlui](#), [document.xmlPanel\(\)](#), [xmlui.getControlItemElement\(\)](#), [xmlui.set\(\)](#),  
[xmlui.setControlItemElements\(\)](#)

## xmlui.setControlItemElements()

### Availability

Flash 8.

### Usage

```
xmlui.setControlItemElements(controlID, elementItemArray)
```

### Parameters

**controlID** A string that specifies the ID attribute of the control you want to set.

**elementItemArray** An array of JavaScript objects, where each object has a string property named `label` and an optional string property named `value`. If the `value` property does not exist, then it is created and assigned the same value as `label`.

### Returns

Nothing.

### Description

Method; clears the values of the ListBox or ComboBox control specified by `controlID` and replaces the list or menu items with the `label`, `value` pairs specified by `elementItemArray`.

### Example

The following example sets the label and value of items in the control with the ID attribute `myControlID` to the `label`, `value` pairs specified:

```
var nameArray = new Array("January", "February", "March");
var monthArray = new Array();
for (i=0;i<nameArray.length;i++) {
    elem = new Object();
    elem.label = nameArray[i];
    elem.value = i;
    monthArray[i] = elem;
}
fl.xmlui.setControlItemElements("myControlID", monthArray);
```

**See also**

[xmlui.getControlItemElement\(\)](#), [xmlui.set\(\)](#), [xmlui.setControlItemElement\(\)](#)

## xmlui.setEnabled()

**Availability**

Flash 8.

**Usage**

```
xmlui.setEnabled(controlID, enable)
```

**Parameters**

**controlID** A string that specifies the ID attribute of the control you want to enable or disable.

**enable** A Boolean value of `true` if you want to enable the control, or `false` if you want to disable (dim) it.

**Returns**

Nothing.

**Description**

Method; enables or disables (dims) a control.

**Example**

The following example dims the control with the ID attribute `myControl`:

```
fl.xmlui.setEnabled("myControl", false);
```

**See also**

[xmlui.getEnabled\(\)](#)

## xmlui.setVisible()

**Availability**

Flash 8.

**Usage**

```
xmlui.setVisible(controlID, visible)
```

**Parameters**

**controlID** A string that specifies the ID attribute of the control you want to show or hide.

**visible** A Boolean value of `true` if you want to show the control; `false` if you want to hide it.

**Returns**

Nothing.

**Description**

Method; shows or hides a control.

**Example**

The following example hides the control with the ID attribute `myControl`:

```
fl.xmlui.setVisible("myControl", false);
```

**See also**

[xmlui.getVisible\(\)](#)

# Chapter 53: C-Level Extensibility

## About extensibility

This chapter describes the C-level extensibility mechanism, which lets you implement Adobe Flash Professional extensibility files using a combination of JavaScript and custom C code.

**Note:** *Flash Professional CC runs on 64-bit operating systems only. All C extensions for this release must be built (or rebuilt) for 64 bit support.*

To implement extensibility, you define functions using C, bundle them in a dynamic linked library (DLL) or a shared library, save the library in the appropriate directory, and then call the functions from JavaScript using the Adobe Flash JavaScript API.

For example, you might want to define a function that performs intense calculations more efficiently than JavaScript does, which improves performance, or when you want to create more advanced tools or effects.

This extensibility mechanism is a subset of the Adobe Dreamweaver CS3 API. If you are familiar with that API, you might recognize the functions in the C-level extensibility mechanism API. However, this API differs from the Dreamweaver API in the following ways:

- This API does not contain all the commands in the Dreamweaver API.
- All declarations of type `wchar_t` and `char` in the Dreamweaver API are implemented as `unsigned short` declarations in this API, to support Unicode when strings are passed.
- The `JSVal JS_BytesToValue()` function in this API is not part of the Dreamweaver API.
- The location in which the DLL or shared library files must be stored is different (see “[Integrating C functions](#)” on page 591).

## Integrating C functions

The C-level extensibility mechanism lets you implement Flash extensibility files using a combination of JavaScript and C code.

**Note:** *Flash Professional CC runs on 64-bit operating systems only. All C extensions for this release must be built (or rebuilt) for 64 bit support.*

The process for implementing this capability is summarized in the following steps:

- 1 Define functions using the C or C++ language.
- 2 Bundle them in a DLL file (Windows) or a shared library (Macintosh).
- 3 Save the DLL file or library in the appropriate location:
  - Windows 7 and 8:  
`boot drive\Users\username\AppData\Local\Adobe\Flash CC\language\Configuration\External Libraries`
  - Mac OS X:  
`Macintosh HD/Users/username/Library/Application Support/Adobe/Flash CC/language/Configuration/External Libraries`

- 4 Create a JSFL file that calls the functions.
- 5 Run the JSFL file from the Commands menu in the Flash authoring environment.

For more information, see “[Sample DLL implementation](#)” on page 595.

## C-level extensibility and the JavaScript interpreter

The C code in the DLL or shared library interacts with the Flash JavaScript API at three different times:

- At startup, to register the library’s functions
- When the C function is called, to unpack the arguments that are being passed from JavaScript to C
- Before the C function returns, to package the return value

To accomplish these tasks, the interpreter defines several data types and exposes an API. Definitions for the data types and functions that are listed in this section appear in the mm\_jsapi.h file. For your library to work properly, you must include the mm\_jsapi.h file at the top of each file in your library, with the following line:

```
#include "mm_jsapi.h"
```

Including the mm\_jsapi.h file includes the mm\_jsapi\_environment.h file, which defines the MM\_Environment structure.

To get a copy of the mm\_jsapi.h file, extract it from the sample ZIP or SIT file (see “[Sample DLL implementation](#)” on page 595), or copy the following code into a file that you name mm\_jsapi.h:

```
#ifndef _MM_JSAPI_H_
#define _MM_JSAPI_H_

/********************* Public data types ********************/
typedef struct JSContext JSContext;
typedef struct JSObject JSObject;
typedef long jsval;
#ifndef JSBool
typedef long JSBool;
#endif

typedef JSBool (*JSNative) (JSContext *cx, JSObject *obj, unsigned int argc,
jsval *argv, jsval *rval);

/* Possible values for JSBool */
#define JS_TRUE 1
#define JS_FALSE 0

/********************* Public functions ********************/
/* JSBool JS_DefineFunction(unsigned short *name, JSNative call, unsigned int nargs) */
#define JS_DefineFunction(n, c, a) \
(mmEnv.defineFunction ? (*(mmEnv.defineFunction))(mmEnv.libObj, n, c, a) \
: JS_FALSE)
```

```
/* unsigned short *JS_ValueToString(JSContext *cx, jsval v, unsigned int *pLength) */
#define JS_ValueToString(c, v, l) \
(mmEnv.valueToString? (*(mmEnv.valueToString))(c, v, l) : (char *)0)

/* unsigned char *JS_ValueToBytes(JSContext *cx, jsval v, unsigned int *pLength) */
#define JS_ValueToBytes(c, v, l) \
(mmEnv.valueToBytes? (*(mmEnv.valueToBytes))(c, v, l) : (unsigned char *)0)

/* JSBool JS_ValueToInteger(JSContext *cx, jsval v, long *lp); */
#define JS_ValueToInteger(c, v, l) \
(mmEnv.valueToInteger ? (*(mmEnv.valueToInteger))(c, v, l) : JS_FALSE)

/* JSBool JS_ValueToDouble(JSContext *cx, jsval v, double *dp); */
#define JS_ValueToDouble(c, v, d) \
(mmEnv.valueToDouble? (*(mmEnv.valueToDouble))(c, v, d) : JS_FALSE)

/* JSBool JS_ValueToBoolean(JSContext *cx, jsval v, JSBool *bp); */
#define JS_ValueToBoolean(c, v, b) \
(mmEnv.valueToBoolean ? (*(mmEnv.valueToBoolean))(c, v, b) : JS_FALSE)

/* JSBool JS_ValueToObject(JSContext *cx, jsval v, JSObject **op); */
#define JS_ValueToObject(c, v, o) \
(mmEnv.valueToObject? (*(mmEnv.valueToObject))(c, v, o) : JS_FALSE)

/* JSBool JS_StringToValue(JSContext *cx, unsigned short *bytes, uint sz, jsval *vp); */
#define JS_StringToValue(c, b, s, v) \
(mmEnv.stringToValue? (*(mmEnv.stringToValue))(c, b, s, v) : JS_FALSE)

/* JSBool JS_BytesToValue(JSContext *cx, unsigned char *bytes, uint sz, jsval *vp); */
#define JS_BytesToValue(c, b, s, v) \
(mmEnv.bytesToValue? (*(mmEnv.bytesToValue))(c, b, s, v) : JS_FALSE)

/* JSBool JS_DoubleToValue(JSContext *cx, double dv, jsval *vp); */
#define JS_DoubleToValue(c, d, v) \
(mmEnv.doubleToValue? (*(mmEnv.doubleToValue))(c, d, v) : JS_FALSE)

/* jsval JS_IntegerToValue(long lv); */
#define JS_IntegerToValue(lv) (((jsval)(lv) << 1) | 0x1)

/* jsval JS_BooleanToValue(JSBool bv); */
#define JS_BooleanToValue(bv) (((jsval)(bv) << 3) | 0x6)

/* jsval JS_ObjectToValue(JSObject *obj); */
#define JS_ObjectToValue(ov) ((jsval)(ov))

/* unsigned short *JS_ObjectType(JSObject *obj); */
#define JS_ObjectType(o) \
(mmEnv.objectType ? (*(mmEnv.objectType))(o) : (char *)0)

/* JSObject *JS_NewArrayObject(JSContext *cx, unsigned int length, jsval *v) */
#define JS_NewArrayObject(c, l, v) \
(mmEnv.newArrayObject ? (*(mmEnv.newArrayObject))(c, l, v) : (JSObject *)0)

/* long JS_GetArrayLength(JSContext *cx, JSObject *obj) */
#define JS_GetArrayLength(c, o) \
(mmEnv.getArrayLength ? (*(mmEnv.getArrayLength))(c, o) : -1)
```

```

/* JSBool JS_GetElement(JSContext *cx, JSObject *obj, jsint idx, jsval *vp) */
#define JS_GetElement(c, o, i, v) \
(mmEnv.getElement ? (*(mmEnv.getElement))(c, o, i, v) : JS_FALSE)

/* JSBool JS_SetElement(JSContext *cx, JSObject *obj, jsint idx, jsval *vp) */
#define JS_SetElement(c, o, i, v) \
(mmEnv.setElement ? (*(mmEnv.createElement))(c, o, i, v) : JS_FALSE)

/* JSBool JS_ExecuteScript(JSContext *cx, JSObject *obj, unsigned short *script,
 * unsigned int sz, jsval *rval) */
#define JS_ExecuteScript(c, o, s, z, r) \
(mmEnv.executeScript? (*(mmEnv.executeScript))(c, o, s, z, (LPCTSTR)_FILE_, \
__LINE__, r) : JS_FALSE)

/* JSBool JS_ReportError(JSContext *cx, unsigned short *error, unsigned int sz) */
#define JS_ReportError(c, e, s) \
(mmEnv.reportError? (*(mmEnv.reportError))(c, e, s) : JS_FALSE)

/********************* Private data types, macros, and globals *****************/
/* Private data types, macros, and globals
 ****
 */

typedef struct {
JSObject *libObj;
JSBool (*defineFunction)(JSObject *libObj, unsigned short *name, JSNative call,
unsigned int nargs);
unsigned short *(*valueToString)(JSContext *cx, jsval v, unsigned int *pLength);
unsigned char *(*valueToBytes)(JSContext *cx, jsval v, unsigned int *pLength);
JSBool (*valueToInteger)(JSContext *cx, jsval v, long *lp);
JSBool (*value.ToDouble)(JSContext *cx, jsval v, double *dp);
JSBool (*value.ToBoolean)(JSContext *cx, jsval v, JSBool *bp);
JSBool (*valueToObject)(JSContext *cx, jsval v, JSObject **op);
JSBool (*stringToValue)(JSContext *cx, unsigned short *b, unsigned int sz, jsval *vp);
JSBool (*bytesToValue)(JSContext *cx, unsigned char *b, unsigned int sz, jsval *vp);
JSBool (*doubleToValue)(JSContext *cx, double dv, jsval *vp);
unsigned short *(*objectType)(JSObject *obj);
JSObject *(*newArrayObject)(JSContext *cx, unsigned int length, jsval *vp);
long (*getArrayLength)(JSContext *cx, JSObject *obj);
JSBool (*getElement)(JSContext *cx, JSObject *obj, unsigned int idx,
jsval *vp);
JSBool (*setElement)(JSContext *cx, JSObject *obj, unsigned int idx,
jsval *vp);
JSBool (*executeScript)(JSContext *cx, JSObject *obj, unsigned short *script,
unsigned int sz, unsigned short *file, unsigned int lineNumber, jsval *rval);
JSBool (*reportError)(JSContext *cx, unsigned short *error, unsigned int sz);
} MM_Environment;

extern MM_Environment mmEnv;

// Declare the external entry point and linkage
#endif _WIN32
#ifndef _MAC
// Windows
__declspec( dllexport ) void MM_InitWrapper( MM_Environment *env, unsigned int envSize );
#endif

```

```
#else
extern void MM_InitWrapper( MM_Environment *env, unsigned int envSize );
#endif

#define MM_STATE\
/* Definitions of global variables */ \
MM_Environment mmEnv; \
\
void\
MM_InitWrapper(MM_Environment *env, unsigned int envSize) \
{ \
extern void MM_Init();\
\
char **envPtr = (char **)env; \
char **mmPtr = (char **)(&mmEnv); \
char **envEnd = (char **)((char *)envPtr + envSize); \
char **mmEnd = (char **)((char *)mmPtr + sizeof(MM_Environment)); \
\
/* Copy fields from env to mmEnv, one pointer at a time */\
while (mmPtr < mmEnd && envPtr < envEnd)\
*mmPtr++ = *envPtr++; \
\
/* If env doesn't define all of mmEnv's fields, set extras to NULL */ \
while (mmPtr < mmEnd) \
*mmPtr++ = (char *)0; \
\
/* Call user's MM_Init function */\
MM_Init();\
} \
}

#endif /* _MM_JSAPI_H_ */
```

## Sample DLL implementation

This section illustrates how to build a simple DLL implementation. If you want to see how the process works without actually building the DLL yourself, you can install the sample DLL files that are provided in the Samples.zip file; the files are located in the ExtendingFlash/dllSampleComputeSum folder. (For information on downloading the Samples.zip file, see “[Sample implementations](#)” on page 16.) Extract the sample files from the dllSampleComputeSum.dmg or dllSampleComputeSum.zip file, and then do the following:

- Store the Sample.jsfl file in the Configuration/Commands directory (see “[Saving JSFL files](#)” on page 2).
- Store the Sample.dll file in the Configuration/External Libraries directory (see “[Integrating C functions](#)” on page 591).
- In the Flash authoring environment, select Commands > Sample. The trace statement in the JSFL file sends the results of the function defined in Sample.dll to the Output panel.

The rest of this section discusses the development of the sample. In this case, the DLL contains only one function, which adds two numbers. The C code is shown in the following example:

```
// Source code in C
// Save the DLL or shared library with the name "Sample".
#include <windows.h>
#include <stdlib.h>

#include "mm_jsapi.h"

// A sample function
// Every implementation of a JavaScript function must have this signature.
JSBool computeSum(JSContext *cx, JSObject *obj, unsigned int argc, jsval *argv, jsval *rval)
{
    long a, b, sum;

    // Make sure the right number of arguments were passed in.
    if (argc != 2)
        return JS_FALSE;

    // Convert the two arguments from jsvals to longs.
    if (JS_ValueToInteger(cx, argv[0], &a) == JS_FALSE ||
        JS_ValueToInteger(cx, argv[1], &b) == JS_FALSE)
        return JS_FALSE;

    /* Perform the actual work. */
    sum = a + b;

    /* Package the return value as a jsval. */
    *rval = JS_IntegerToValue(sum);

    /* Indicate success. */
    return JS_TRUE;
}
```

After writing this code, build the DLL file or shared library, and store it in the appropriate Configuration/External Libraries directory (see “[Integrating C functions](#)” on page 591). Then create a JSFL file with the following code, and store it in the Configuration/Commands directory (see “[Saving JSFL files](#)” on page 2).

```
// JSFL file to run C function defined above.
var a = 5;
var b = 10;
var sum = Sample.computeSum(a, b);
fl.trace("The sum of " + a + " and " + b + " is " + sum );
```

To run the function defined in the DLL, select Commands > Sample in the Flash authoring environment.

## Data types

The JavaScript interpreter defines the data types described in this section.

### **typedef struct JSContext JSContext**

A pointer to this opaque data type passes to the C-level function. Some functions in the API accept this pointer as one of their arguments.

### **typedef struct JSObject JSObject**

A pointer to this opaque data type passes to the C-level function. This data type represents an object, which might be an array object or some other object type.

### **typedef struct jsval jsval**

An opaque data structure that can contain an integer, or a pointer to a float, string, or object. Some functions in the API can read the values of function arguments by reading the contents of a `jsval` structure, and some can be used to write the function's return value by writing a `jsval` structure.

### **typedef enum { JS\_FALSE = 0, JS\_TRUE = 1 } JSBool**

A simple data type that stores a Boolean value.

## **The C-level API**

The C-level extensibility API consists of the `JSBool (*JSNative)` function signature and the following functions:

- `JSBool JS_DefineFunction()`
- `unsigned short *JS_ValueToString()`
- `JSBool JS_ValueToInteger()`
- `JSBool JS_ValueToDouble()`
- `JSBool JS_ValueToBoolean()`
- `JSBool JS_ValueToObject()`
- `JSBool JS_StringToValue()`
- `JSBool JS_DoubleToValue()`
- `JSVal JS_BooleanToValue()`
- `JSVal JS_BytesToValue()`
- `JSVal JS_IntegerToValue()`
- `JSVal JS_ObjectToValue()`
- `unsigned short *JS_ObjectType()`
- `JSObject *JS_NewArrayObject()`
- `long JS_GetArrayLength()`
- `JSBool JS_GetElement()`
- `JSBool JS_SetElement()`
- `JSBool JS_ExecuteScript()`

*Note:* Flash Professional CC runs on 64-bit operating systems only. All C extensions for this release must be built (or rebuilt) for 64 bit support.

```
typedef JSBool (*JSNative)(JSContext *cx, JSObject *obj, unsigned int argc,
jsval *argv, jsval *rval)
```

#### Description

Method; describes C-level implementations of JavaScript functions in the following situations:

- The *cx* pointer is a pointer to an opaque `JSContext` structure, which must be passed to some of the functions in the JavaScript API. This variable holds the interpreter's execution context.
- The *obj* pointer is a pointer to the object in whose context the script executes. While the script is running, the `this` keyword is equal to this object.
- The *argc* integer is the number of arguments being passed to the function.
- The *argv* pointer is a pointer to an array of `jsval` structures. The array is `argc` elements in length.
- The *rval* pointer is a pointer to a single `jsval` structure. The function's return value should be written to `*rval`.

The function returns `JS_TRUE` if successful; `JS_FALSE` otherwise. If the function returns `JS_FALSE`, the current script stops executing and an error message appears.

## **JSBool JS\_DefineFunction()**

#### Usage

```
JSBool JS_DefineFunction(unsigned short *name, JSNative call, unsigned int nargs)
```

#### Description

Method; registers a C-level function with the JavaScript interpreter in Flash. After the `JS_DefineFunction()` function registers the C-level function that you specify in the *call* argument, you can invoke it in a JavaScript script by referring to it with the name that you specify in the *name* argument. The *name* argument is case-sensitive.

Typically, this function is called from the `MM_Init()` function, which Flash calls during startup.

#### Arguments

```
unsigned short *name, JSNative call, unsigned int nargs
```

- The *name* argument is the name of the function as it is exposed to JavaScript.
- The *call* argument is a pointer to a C-level function. The function must return a `JSBool`, which indicates success or failure.
- The *nargs* argument is the number of arguments that the function expects to receive.

#### Returns

A Boolean value: `JS_TRUE` indicates success; `JS_FALSE` indicates failure.

## **unsigned short \*JS\_ValueToString()**

#### Usage

```
unsigned short *JS_ValueToString(JSContext *cx, jsval v, unsigned int *pLength)
```

**Description**

Method; extracts a function argument from a `jsval` structure, converts it to a string, if possible, and passes the converted value back to the caller.

**Note:** Do not modify the returned buffer pointer or you might corrupt the data structures of the JavaScript interpreter. To change the string, you must copy the characters into another buffer and create a new JavaScript string.

**Arguments**

```
JSContext *cx, jsval v, unsigned int *pLength
```

- The `cx` argument is the opaque `JSContext` pointer that passes to the JavaScript function.
- The `v` argument is the `jsval` structure from which the string is to be extracted.
- The `pLength` argument is a pointer to an unsigned integer. This function sets `*pLength` equal to the length of the string in bytes.

**Returns**

A pointer that points to a null-terminated string if successful or to a `null` value on failure. The calling routine must not free this string when it finishes.

## **JSBool JS\_ValueToInteger()**

**Usage**

```
JSBool JS_ValueToInteger(JSContext *cx, jsval v, long *lp);
```

**Description**

Method; extracts a function argument from a `jsval` structure, converts it to an integer (if possible), and passes the converted value back to the caller.

**Arguments**

```
JSContext *cx, jsval v, long *lp
```

- The `cx` argument is the opaque `JSContext` pointer that passes to the JavaScript function.
- The `v` argument is the `jsval` structure from which the integer is to be extracted.
- The `lp` argument is a pointer to a 4-byte integer. This function stores the converted value in `*lp`.

**Returns**

A Boolean value: `JS_TRUE` indicates success; `JS_FALSE` indicates failure.

## **JSBool JS\_ValueToDouble()**

**Usage**

```
JSBool JS_ValueToDouble(JSContext *cx, jsval v, double *dp);
```

**Description**

Method; extracts a function argument from a `jsval` structure, converts it to a double (if possible), and passes the converted value back to the caller.

**Arguments**

```
JSContext *cx, jsval v, double *dp
```

- The *cx* argument is the opaque `JSContext` pointer that passed to the JavaScript function.
- The *v* argument is the `jsval` structure from which the double is to be extracted.
- The *dp* argument is a pointer to an 8-byte double. This function stores the converted value in `*dp`.

**Returns**

A Boolean value: `JS_TRUE` indicates success; `JS_FALSE` indicates failure.

## **JSBool JS\_ValueToBoolean()**

**Usage**

```
JSBool JS_ValueToBoolean(JSContext *cx, jsval v, JSBool *bp);
```

**Description**

Method; extracts a function argument from a `jsval` structure, converts it to a Boolean value (if possible), and passes the converted value back to the caller.

**Arguments**

```
JSContext *cx, jsval v, JSBool *bp
```

- The *cx* argument is the opaque `JSContext` pointer that passes to the JavaScript function.
- The *v* argument is the `jsval` structure from which the Boolean value is to be extracted.
- The *bp* argument is a pointer to a `JSBool` Boolean value. This function stores the converted value in `*bp`.

**Returns**

A Boolean value: `JS_TRUE` indicates success; `JS_FALSE` indicates failure.

## **JSBool JS\_ValueToObject()**

**Usage**

```
JSBool JS_ValueToObject(JSContext *cx, jsval v, JSObject **op);
```

**Description**

Method; extracts a function argument from a `jsval` structure, converts it to an object (if possible), and passes the converted value back to the caller. If the object is an array, use `JS_GetArrayLength()` and `JS_GetElement()` to read its contents.

**Arguments**

```
JSContext *cx, jsval v, JSObject **op
```

- The *cx* argument is the opaque `JSContext` pointer that passes to the JavaScript function.
- The *v* argument is the `jsval` structure from which the object is to be extracted.
- The *op* argument is a pointer to a `JSObject` pointer. This function stores the converted value in `*op`.

**Returns**

A Boolean value: `JS_TRUE` indicates success; `JS_FALSE` indicates failure.

## **JSBool JS\_StringToValue()**

**Usage**

```
JSBool JS_StringToValue(JSContext *cx, unsigned short *bytes, uint sz, jsval *vp);
```

**Description**

Method; stores a string return value in a `jsval` structure. It allocates a new JavaScript string object.

**Arguments**

```
JSContext *cx, unsigned short *bytes, size_tsz, jsval *vp
```

- The `cx` argument is the opaque `JSContext` pointer that passes to the JavaScript function.
- The `bytes` argument is the string to be stored in the `jsval` structure. The string data is copied, so the caller should free the string when it is not needed. If the string size is not specified (see the `sz` argument), the string must be null-terminated.
- The `sz` argument is the size of the string, in bytes. If `sz` is 0, the length of the null-terminated string is computed automatically.
- The `vp` argument is a pointer to the `jsval` structure into which the contents of the string should be copied.

**Returns**

A Boolean value: `JS_TRUE` indicates success; `JS_FALSE` indicates failure.

## **JSBool JS\_DoubleToValue()**

**Usage**

```
JSBool JS_DoubleToValue(JSContext *cx, double dv, jsval *vp);
```

**Description**

Method; stores a floating-point number return value in a `jsval` structure.

**Arguments**

```
JSContext *cx, double dv, jsval *vp
```

- The `cx` argument is the opaque `JSContext` pointer that passes to the JavaScript function.
- The `dv` argument is an 8-byte floating-point number.
- The `vp` argument is a pointer to the `jsval` structure into which the contents of the double should be copied.

**Returns**

A Boolean value: `JS_TRUE` indicates success; `JS_FALSE` indicates failure.

## JSVal JS\_BooleanToValue()

### Usage

```
jsval JS_BooleanToValue(JSBool bv);
```

### Description

Method; stores a Boolean return value in a `jsval` structure.

### Arguments

`JSBool bv`

- The `bv` argument is a Boolean value: `JS_TRUE` indicates success; `JS_FALSE` indicates failure.

### Returns

A `JSVal` structure that contains the Boolean value that passes to the function as an argument.

## JSVal JS\_BytesToValue()

### Usage

```
JSBool JS_BytesToValue(JSContext *cx, unsigned short *bytes, uint sz, jsval *vp);
```

### Description

Method; converts bytes to a JavaScript value.

### Arguments

`JSContext *cx, unsigned short *bytes, uint sz, jsval *vp`

- The `cx` argument is the JavaScript context.
- The `bytes` argument is the string of bytes to convert to a JavaScript object.
- The `sz` argument is the number of bytes to be converted.
- The `vp` argument is the JavaScript value.

### Returns

A Boolean value: `JS_TRUE` indicates success; `JS_FALSE` indicates failure.

## JSVal JS\_IntegerToValue()

### Usage

```
jsval JS_IntegerToValue(long lv);
```

### Description

Method; converts a long integer value to `JSVal` structure.

### Arguments

`lv`

The `lv` argument is the long integer value that you want to convert to a `jsval` structure.

**Returns**

A `JSVal` structure that contains the integer that passed to the function as an argument.

## **JSVal JS\_ObjectToValue()**

**Usage**

```
jsval JS_ObjectToValue(JSObject *obj);
```

**Description**

Method; stores an object return value in a `JSVal`. Use `JS_NewArrayObject()` to create an array object; use `JS_SetElement()` to define its contents.

**Arguments**

`JSObject *obj`

The `obj` argument is a pointer to the `JSObject` object that you want to convert to a `JSVal` structure.

**Returns**

A `JSVal` structure that contains the object that you passed to the function as an argument.

## **unsigned short \*JS\_ObjectType()**

**Usage**

```
unsigned short *JS_ObjectType(JSObject *obj);
```

**Description**

Method; given an object reference, returns the class name of the object. For example, if the object is a DOM object, the function returns "Document". If the object is a node in the document, the function returns "Element". For an array object, the function returns "Array".

*Note:* Do not modify the returned buffer pointer, or you might corrupt the data structures of the JavaScript interpreter.

**Arguments**

`JSObject *obj`

Typically, this argument is passed in and converted using the `JS_ValueToObject()` function.

**Returns**

A pointer to a null-terminated string. The caller should not free this string when it finishes.

## **JSObject \*JS\_NewArrayObject()**

**Usage**

```
JSObject *JS_NewArrayObject(JSContext *cx, unsigned int length [, jsval *v])
```

**Description**

Method; creates a new object that contains an array of `JSVals`.

**Arguments**

```
JSContext *cx, unsigned int length, jsval *v
```

- The *cx* argument is the opaque `JSContext` pointer that passes to the JavaScript function.
- The *length* argument is the number of elements that the array can hold.
- The *v* argument is an optional pointer to the `jsvals` to be stored in the array. If the return value is not `null`, *v* is an array that contains *length* elements. If the return value is `null`, the initial content of the array object is undefined and can be set using the `JS_SetElement()` function.

**Returns**

A pointer to a new array object or the value `null` upon failure.

## **long JS\_GetArrayLength()**

**Usage**

```
long JS_GetArrayLength(JSContext *cx, JSObject *obj)
```

**Description**

Method; given a pointer to an array object, gets the number of elements in the array.

**Arguments**

```
JSContext *cx, JSObject *obj
```

- The *cx* argument is the opaque `JSContext` pointer that passes to the JavaScript function.
- The *obj* argument is a pointer to an array object.

**Returns**

The number of elements in the array or -1 upon failure.

## **JSBool JS\_GetElement()**

**Usage**

```
JSBool JS_GetElement(JSContext *cx, JSObject *obj, jsint idx, jsval *vp)
```

**Description**

Method; reads a single element of an array object.

**Arguments**

```
JSContext *cx, JSObject *obj, jsint idx, jsval *vp
```

- The *cx* argument is the opaque `JSContext` pointer that passes to the JavaScript function.
- The *obj* argument is a pointer to an array object.
- The *idx* argument is an integer index into the array. The first element is index 0, and the last element is index (`length` - 1).

- The *vp* argument is a pointer to a `jsval` where the contents of the `jsval` structure in the array should be copied.

**Returns**

A Boolean value: `JS_TRUE` indicates success; `JS_FALSE` indicates failure.

## **JSBool JS\_SetElement()**

**Usage**

```
JSBool JS_SetElement(JSContext *cx, JSObject *obj, jsint idx, jsval *vp)
```

**Description**

Method; writes a single element of an array object.

**Arguments**

```
JSContext *cx, JSObject *obj, jsint idx, jsval *vp
```

- The *cx* argument is the opaque `JSContext` pointer that passes to the JavaScript function.
- The *obj* argument is a pointer to an array object.
- The *idx* argument is an integer index into the array. The first element is index 0, and the last element is index (`length - 1`).
- The *vp* argument is a pointer to a `jsval` structure whose contents should be copied to the `jsval` in the array.

**Returns**

A Boolean value: `JS_TRUE` indicates success; `JS_FALSE` indicates failure.

## **JSBool JS\_ExecuteScript()**

**Usage**

```
JS_ExecuteScript (JSContext *cx, JSObject *obj, unsigned short *script, unsigned int sz, jsval *rval)
```

**Description**

Method; compiles and executes a JavaScript string. If the script generates a return value, it returns in `*rval`.

**Arguments**

```
JSContext *cx, JSObject *obj, unsigned short *script, unsigned int sz, jsval *rval
```

- The *cx* argument is the opaque `JSContext` pointer that passes to the JavaScript function.
- The *obj* argument is a pointer to the object in whose context the script executes. While the script is running, the `this` keyword is equal to this object. Usually this is the `JSObject` pointer that passes to the JavaScript function.
- The *script* argument is a string that contains JavaScript code. If the string size is not specified (see the *sz* argument), the string must be null-terminated.

- The *sz* argument is the size of the string. If *sz* is 0, the length of the null-terminated string is computed automatically.
- The *rval* argument is a pointer to a single `jsval` structure. The function's return value is stored in `*rval`.

**Returns**

A Boolean value: `JS_TRUE` indicates success; `JS_FALSE` indicates failure.